Towards Efficient Risk Quantification - Using GPUs and Variance Reduction Technique

Shih Hau Tan

Towards Efficient Risk Quantification

- Using GPUs and Variance Reduction Technique

Shih-Hau Tan

Erasmus Mundus MathMods Programme

Université Nice Sophia Antipolis

Supervisors

Dr. Mireille Bossy

Prof. Françoise Baude

A thesis submitted for the degree of Master of Science
September, 2013

Résumé

Value-at-Risk (VaR) nous donne de renseignements sur le risque total dans le commerce lorsque nous devons faire la gestion des risques. La demande du calcul rapide de la VaR se développe parce que les établissements financiers et les entreprises veulent mesurer le risque en temps réel; et récemment de nombreux chercheurs explorent le potentiel du calcul à haute performance pour le faire.

Nous introduisons deux possibilités provenant de mathématiques et GPU computing pour faire face à ce problème. Nous l'avons également mis en œuvre avec des exemples afin de comparer les résultats, pour voir combien d'accélération nous pouvons gagner. Enfin, nous discutons d'autres approches qui peuvent être les futurs travaux.

Remerciements

Je tiens tout à remercier sincèrement mes superviseurs Dr Mireille Bossy et Professeur Françoise Baude pour leurs conseils sur cette thèse, l'enthousiasme en discussion lors de la réunion, et beaucoup de soutien pour mon stage.

Je tiens également à remercier Professeur François Delarue pour son enseignement des finances stochastique, ainsi que Dr Etienne Tanré pour son enseignement sur finance computationelle, qui me donnent les connaissances fondamentales pour entamer ce projet.

Par ailleurs, je tiens à remercier le doctorant Michaël Benguigui dans l'INRIA qui m'a aidé en matière de tarification option américaine. Et je tiens à remercier l'ingénieur logiciel Lung-Sheng Chien pour d'avoir la gentillesse de répondre à toutes mes questions sur le GPU Computing. Je tiens également à remercier le doctorant Guo-Jhen Wu à l'Université Brown pour ses suggestions sur le calcul stochastique.

Je suis reconnaissant aux équipes TOSCA et OASIS à l'INRIA de m'avoir donné la chance de faire mon stage.

Enfin, je tiens à remercier mon père, ma mère et mon frère, pour leur précieux soutien et d'encouragement pendant mes études en Europe.

Contents

Chapter 1

Introduction

In financial market, investors develop their own strategies to achieve benefits on investment of stocks, futures, options, and so on. On the other hand, companies measure risk in order to avoid loss of every trading. Modeling the financial instruments, computing the price and analyzing the sensitivity contributes the development of financial mathematics.

In 1973, Fisher Black and Myron Scholes first gave an analytic ways to solve problem on option pricing in [1]. They derived equation so called Black-Scholes equation now to understand how to calculate the price of vanilla option. Robert C. Merton expanded their result to Black-Scholes Model in [12] which is widely used today. However, there is an limitation of analytic solution on option pricing when we discuss more complicated financial products, like portfolio, exotic option. Therefore, numerical computation plays an important role for people who want to investigate more on this topic. In chapter 2, we will give a review of the computation of option price.

Research Problem

Once we have method to compute the price of portfolio, we may be interested in how much risk we have to afford on each trading. It turns to the problem of calculating the Value-at-Risk (which is so called VaR). VaR is a value which gives us the information on the loss of the investment with a confidence in the next time. We can define the VaR with α confidence as the minimum value x which satisfies

$$\mathbb{P}(L > x) = 1 - \alpha$$

where L is the loss function which can be defined as $L = -\Delta V = V(S,t) - V(S + \Delta S, t + \Delta t)$. The value V is the price of portfolio, and S is underlying assets. α is a confident level which satisfies $0 \leq \alpha \leq 1$.

The computation of VaR relies on the value of loss from portfolio prices (see [2], [3]), so it will take much time if the price is computationally expensive. This is the difficulty of VaR computation and usually can be seen in problems with large size, like basket option. Then there is no doubt that getting the exact VaR

value in a fast and possible way becomes a crucial problem. The main research on this thesis is trying to find an efficient way to do the computation, especially for complicated financial products, like option book.

Methodology

As we mention, we are trying to deal with problem with big size, so the common and efficient way is to use Monte Carlo method especially it is easily to dig out possibility for parallel computing. How we use Monte Carlo method on VaR computation will be introduced in chapter 3.

Another questions may be asked is that we can expect we will have to spend a lot of time on simulation since we need enough samples to do Monte Carlo method in order to achieve good accuracy. Therefore how to reduce the time to be acceptable is another topic we will discuss. Paul Glasserman gave a series of methods ([4], [5]) on using variance reduction technique for VaR computation. After that, many algorithms are created in order to solve the time-consuming problem. We choose two methods called Importance Sampling method and Importance Splitting method to see how to imply them into our problem. More details will be given in chapter 4.

With the help of recent evolution of hardware and architecture, the time-consuming problem seems can also be solved by doing computation on graphics processing units, in other word, using GPU computing. GPUs nowadays are widely used in various area doing high performance computing, and it speeds up dramatically when there are many possibilities to do parallel computing. Hence exploring the opportunity to use GPUs is another approach to deal with the problem with massive computing. We will provide our strategy on using GPUs in chapter 6.

Contributions

In this thesis, we provide two main contributions on VaR computation as following:

- mathematical approaches from importance sampling and importance splitting method

We borrow ideas from Computational Physics (see [6], [7]) to study how to use the importance splitting method to calculate VaR. This method was decided for rare event first, and we try to apply it to our problem with a provided search algorithm. By using such method, we can avoid simulating too much samples and do the computation more efficiently. We compare the result with the importance sampling method and analyze the risks on using both methods. More details can be found in chapter 4.

- a framework on calculating VaR using GPUs

Since one of the reasons we choose Monte Carlo method to do simulation is that there is a potential to do parallel computing, so we develop a framework for the VaR computation on option book. Also we try to investigate the possibilities to use GPUs in the whole algorithm and implement it with CUDA programming. These will be discussed in details in chapter 6.

Future works

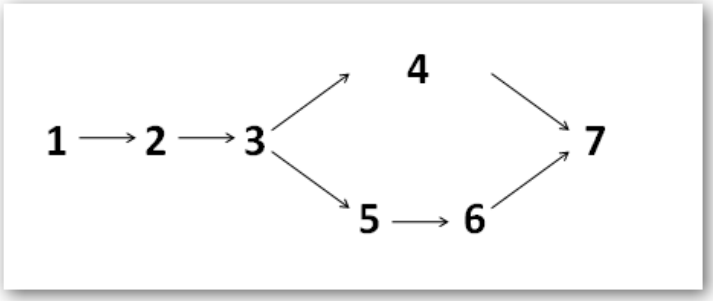
Finally we give some approaches which are related to this thesis. Most of them are not yet finished works during the internship. We hope to finish them in the future.

a. doing importance sampling method with GPUs
b. VaR for American option
c. interpolation method
d. stochastic approximation method
e. Quasi Monte Carlo method

We will address details in chapter 7.

Organization of Thesis

This thesis is organized as follows. In chapter 2, we give a short review on basic knowledge of option pricing and the example we are going to calculate. Chapter 3 states the methods on VaR computation and the difficulty we have to solve. The answer will be given in chapter 4, which we try to study Importance Sampling and Importance Splitting methods. Both methods address useful mathematical approaches, and we are going to see how to apply it to our problem. Because one of the advantage using Monte Carlo method is we can do parallel computing. Hence in chapter 5, we give an introduction on GPU computing and how to use it. Chapter 6 then will give another answer on solving the difficulty of VaR computation by providing some parallel strategies and implementation. Finally we give some conclusions and other approaches which might be the future works in chapter 7. The order of each chapter is designed as following:



Chapter 2

Review of Option Pricing

As one of the most typical derivatives, options have already been well-known and traded for many centuries. An option is a contract which gives its owner the right to buy or sell the underlying asset at a fixed price at any time before a given date. Two basic forms of options are *call* which allows the owner to buy and *put* which allows the owner to sell. Usually people hold an option for speculation or hedging.

For example. The price of jet fuel is important for airlines since high price of jet fuel always influences the benefits. However the price is usually hard to predictable. A way to avoid the loss of profits is to buy a call option of the jet fuel. Then if the price of jet fuel increases and causes deficits of business, we can exercise our option to achieve the profits and to balance the loss.

There are various kinds of options related to different markets (see [8]). Deciding a fair price for options is important because in some cases we allow people to exercise options immediately after buying it (this is so called American option). Hence in such case if the price of options is lower than the profits after exercising it, then people have a chance to make profits without any risk which is called arbitrage.

In this chapter, we review the basic theory of option pricing on European and American option from the books [9], [10]. One can check more details for deep contents from them. We will also emphasize on describing the basket options which we choose as example in the numerical results.

2.1 European Option

Consider an option with S_T which is the price of underlying asset at maturity time T and strike price K. An European style option allows people to exercise at time T. Then the profit for call and put option will be $S_T - K$ and $K - S_T$. In fact, we can choose not to exercise it if we know the profit is negative. Hence here we define the payoff function for call is $\Phi(S_T, T) = max\{S_T - K, 0\}$ and for put is $\Phi(S_T, T) = max\{K - S_T, 0\}$.

2.1.1 Black-Scholes Model

Before going to the Black-Scholes Model, we first need to understand some assumptions for the financial market. We assume that the price of underlying asset follows a geometrical Brownian motion with constant drift rate and volatility and without paying a dividend. Furthermore, there are no transaction fees to pay and there is no arbitrage opportunity and we are allowed to buy, sell stocks and lend or borrow cash. Then the price of asset can be modelled by

$$dS = \mu S dt + \sigma S dW_t \tag{2.1}$$

or in integration form

$$S_t = S_0 + \int_0^t \mu S_u du + \int_0^t \sigma S_u dW_u$$

where S is price of asset, t is the time we observe, μ is the drift rate, σ is the volatility and W_t is a one-dimensional Brownian motion.

Since we are interested in the price of option. We define $V(S_T, T)$ to be the option price at a known maturity. By Ito's lemma, we have

$$dV = (\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t})dt + \sigma S \frac{\partial V}{\partial S} dW_t \tag{2.2}$$

Now we consider a portfolio Π to be long an option and short an asset with quantity $\frac{\partial V}{\partial S}$, it means

$$\Pi = V - \frac{\partial V}{\partial S} S$$

which implies

$$d\Pi = dV - \frac{\partial V}{\partial S} dS$$

substituting it to equation (2.2), we will obtain

$$d\Pi = (\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})dt$$

In order to achieve no arbitrage opportunity, we know the price of the portfolio must be equal to the price of the portfolio which is invested with a risk-free interest rate r. It means

$$(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})dt = r\Pi dt$$

Finally we put $\Pi = V - \frac{\partial V}{\partial S}$ into the above equation then we will get the so called Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{2.3}$$

with boundary conditions which can be decided from the payoff function.

2.1.2 Analytic Solution

For simpler case, there exists a closed form solution for the price of European call and put option. Consider $C(S_t, t)$ and $P(S_t, t)$ prices of call and put option on non-dividend paying asset at time t. The closed form solution is as following:

$$C(S_t,t) = S_0N(d_1) - Ke^{-r(T-t)}N(d_2), \quad P(S_t,t) = Ke^{-r(T-t)}N(-d_2) - S_0N(-d_1)$$

where

$$d_1 = \frac{log(\frac{S_0}{K}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{(T-t)}}, \quad d_2 = \frac{log(\frac{S_0}{K}) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{(T-t)}}$$

and

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2 du}$$

More details on deriving the formula can be found in [9].

2.1.3 Numerical Methods

However, for complicated options or portfolio, we have to solve it numerically since the closed form solution doesn't exist. There are many techniques to find the price of option (see the book [11]), we give a quick review and then we will focus on Monte Carlo method in the following chapters.

Finite Difference Method

Finite difference method is a common technique on solving Ordinary or Partial Differential Equation. The main idea is to do time and space discretizations, and then construct it as a matrix form to solve the iteration or eigenvalue problem. There exist many schemes, for example, like *Euler method*, *Crank−Nicolson method*, to discretize the equation. Different scheme has its own accuracy and advantages, and we also need to consider the Courant−Friedrich−Lewy condition (CFL condition) for the convergence of the scheme.

For instance, if now we choose Implicit Euler scheme to solve the price of call option, then equation (2.2) can be discretized as

$$\frac{V_{i,j} - V_{i,j-1}}{\Delta t} + \frac{1}{2}\sigma^2(ih)^2 \frac{V_{i+1,j-1} - 2V_{i,j-1} + V_{i-1,j-1}}{h^2} + r(ih)\frac{V_{i+1,j-1} - V_{i-1,j-1}}{2h} - rV_{i,j-1} = 0$$

where $V_{i,j} = V(ih, j\Delta t)$ and boundary conditions are

$$V(S_T, T) = max\{S_T - K, 0\},\ V(0,t) = 0,\ \lim_{S\to\infty} V(S,t) = S$$

We can observe that if now we consider an option which depends on many underlying assets $S_1, ..., S_N$, we will have to do many space discretizations and it turns out to be very expensive on computation.

Monte Carlo method

An alternative way without solving the Partial Differential Equation is using Monte Carlo method. Under the assumption of risk-neutrality, we know the price of option at original time is equal to the expectation of discounted payoff. It means

$$V(S_0, 0) = \mathbb{E}[e^{-rT}\Phi(S_T, T)]$$

which can be approximated by

$$V(S_0, 0) \simeq \frac{1}{NMC} \sum_{i=1}^{NMC} e^{-rT}\Phi(S_T^i, T)$$

here i means different path and NMC means number of paths to do Monte Carlo method. By Central Limit Theorem the convergent rate is $O(\frac{\sigma}{\sqrt{NMC}})$ where σ here is the standard deviation of the estimator. It shows the complexity of this estimator doesn't depend on dimension, but we need to simulate enough samples in order to get acceptable accuracy.

2.1.4 Basket Option

Based on the Black-Scholes model, we can get into the financial market and analyze it mathematically. But problems from financial market are not so easy to solve. There are many financial products which consist of different assets. Therefore, the multi-dimensional Black-Scholes model is developed for these problems, especially for option with basket assets.

Consider the following equation

$$dS_t^i = S_t^i r dt + S_t^i \sigma_i dW_t^i,\ i = 1, ..., d \tag{2.4}$$

where S_i is the price of ith asset, r_i, σ_i are the interest rate and volatility for asset S_i and $(W_1, ..., W_d)$ is a correlated d-dimensional Brownian motion.

We are interested in calculating the Geometric Average option which means if given a $S_t = (S_t^1, ..., S_t^d)$ satisfies the multi-dimensional Black-Scholes model, then the Geometric Average option has the payoff function as

$$\Phi(f(S_t, t)),\ f(S_t) = \prod_{i=1}^{d} (S_t^i)^{1/d}$$

Then the price of Geometric Average option can be computed as the following algorithm:

Algorithm 1: Geometric Average Option Pricing

Input: $S_0^i, r, \sigma_i, T, d, NMC$
Output: V
temp1 = 0;
for *j=1 to NMC* **do**
  temp2 = 1;
  for *i=1 to d* **do**
    Generate $S_T^i = S_0^i e^{(r-\sigma_i)T - \frac{1}{2}\sqrt{T}Z_j^i}$;
    where $Z_j^i \sim N(0,1)$;
    $temp2 = temp2 \times S_T^i$;
  end
  $S_T^i = (S_T^i)^{1/d}$;
  $V(S_T^i, T) = e^{-rT}\Phi(S_T^i, T)$;
  $temp1 = temp1 + V(S_T^i, T)$;
end
$V = \frac{temp1}{NMC}$;

2.1.5 Reduction Method

Pricing the Geometric Average option takes long time if the dimension d is high. An observation gives us an idea that if we can reduce the basket assets into one dimensional object, then we can directly use the Black-Scholes closed form formula for European option. Actually under some special cases it works and the idea is using multi-dimensional Ito's lemma.

Consider a probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t, t \geq 0), \mathbb{P})$ and a $(\mathcal{F}_t)$ -d-dimensional standard Brownian motion $W_t = (W_t^1, ..., W_t^d)$. Let $S_t^0 = e^{rt}$, and a basket option with assets $S_t = (S_t^1, ..., S_t^d)$ satisfy

$$dS_t^i = S_t^i r dt + S_t^i \sum_{k=1}^{d} \sigma_{ik} dW_t^k, \quad 1 \leq i \leq d \tag{2.5}$$

where $\sigma_{ik} \in \mathbb{R}_+$ and with a payoff function $\Phi(f(S_t))$ where

$$f : \mathbb{R}_+^d \to \mathbb{R}_+, \quad f(S_t) = \prod_{i=1}^{d} (S_t^i)^{\alpha_i}$$

and by multi-dimensional Ito's formula, we have

$$df(S_t) = \sum_{i=1}^{d} f_i(S_t) dS_t^i + \frac{1}{2} \sum_{i,j=1}^{d} f_{ij}(S_t) d < S^i, S^j >_t$$

here the first term is equal to
$\sum_{i=1}^{d} f_i(S_t) dS_t^i = \sum_{i=1}^{d} \frac{f(S_t)}{S_t^i} \alpha_i dS_t^i = f(S_t) \sum_{i=1}^{d} \frac{\alpha_i}{S_t^i} dS_t^i = f(S_t) \sum_{i=1}^{d} \frac{\alpha_i}{S_t^i} S_t^i (rdt + \sum_{k=1}^{d} \sigma_{ik} dW_t^k)$
$= f(S_t) \sum_{i=1}^{d} \alpha_i (rdt + \sum_{k=1}^{d} \sigma_{ik} dW_t^k)$

and since

$$d < S^i, S^j >_t = S_t^i S_t^j \sum_{k=1}^{d} \sigma_{ik}\sigma_{jk} dt$$

so the second term is equal to

$$\frac{1}{2} \sum_{i,j=1}^{d} f_{ij}(S_t) d < S^i, S^j >_t = \frac{1}{2}(\sum_{i,j=1, i\neq j}^{d} f(S_t)\alpha_i\alpha_j + \sum_{i,j=1, i=j}^{d} f(S_t)\alpha_i(\alpha_i - 1)) \sum_{k=1}^{d} \sigma_{ik}\sigma_{jk} dt$$

combine these two terms, we have

$$\frac{df(S_t)}{f(S_t)} = (\sum_{i=1}^{d} \alpha_i r + \frac{1}{2}(\sum_{i,j=1, i\neq j}^{d} \alpha_i\alpha_j + \sum_{i,j=1, i=j}^{d} \alpha_i(\alpha_i - 1)) \sum_{k=1}^{d} \sigma_{ik}\sigma_{jk}) dt + \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_i\sigma_{ik} dW_t^k \tag{2.6}$$

Now let $X_t = \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_i\sigma_{ik} dW_t^k$. We are going to use the following lemma to simplify the computation and get a closed form.

Lemma 2.1.1. $X_t = \beta B_t$ where B_t is a standard Brownian motion.

Proof. First we can write X_t as following:

$$X_t = \begin{pmatrix} \alpha_1 & \alpha_2 & ... & \alpha_d \end{pmatrix} \begin{pmatrix} \sigma_{11} & ... & \sigma_{d1} \\ \dots & ... & \dots \\ \sigma_{1d} & ... & \sigma_{dd} \end{pmatrix} \begin{pmatrix} W_t^1 \\ \dots \\ W_t^d \end{pmatrix}$$

and we know $(W_t^1, ..., W_t^d)$ is d-dimensional standard Brownian motion. Therefore X_t is a linear combination of Gaussian random variables which is also a Gaussian random variable.

By doing calculation, we have

$$\mathbb{E}(X_t) = \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_i\sigma_{ik}\mathbb{E}(W_t^k) = 0$$

$$\mathbb{V}(X_t) = \sum_{k=1}^{d} \mathbb{V}(\sum_{i=1}^{d} \alpha_i\sigma_{ik} W_t^k) = \sum_{k=1}^{d} \mathbb{E}((\sum_{i=1}^{d} \alpha\sigma_{ik} W_t^k)^2) = t \sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d} (\alpha_i\sigma_{ik})(\alpha_j\sigma_{jk})$$

Now since $X_t - X_s$ is also a Gaussian random variable, and we know

$$X_t - X_s = \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_i\sigma_{ik}(W_t^k - W_s^k)$$

then

$$\mathbb{E}(X_t - X_s) = \sum_{i=1}^{d}\sum_{k=1}^{d} \alpha_i\sigma_{ik}\mathbb{E}(W_t^k - W_s^k) = 0$$

and

$$\mathbb{V}(X_t - X_s) = \sum_{k=1}^{d} \mathbb{V}(\sum_{i=1}^{d} \alpha_i\sigma_{ik}(W_t^k - W_s^k)) = \sum_{k=1}^{d} \mathbb{E}((\sum_{i=1}^{d} \alpha_i\sigma_{ik}(W_t^k - W_s^k))^2)$$

$= (t+s-2t\wedge s)\sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_i\sigma_{ik})(\alpha_j\sigma_{jk}) = \mathbb{V}(X_t)+\mathbb{V}(X_s)-2\mathbb{E}(X_tX_s)$

therefore $\mathbb{E}(X_tX_s) = s\sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_i\sigma_{ik})(\alpha_j\sigma_{jk})$

and $X_t - X_s \sim N(0,\sigma^2)$, where $\sigma^2 = \sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_i\sigma_{ik})(\alpha_j\sigma_{jk})$

Finally we claim $(X_t)_{t\geq 0}$ is a Gaussian process. The continuous is clear since it is composed of Brownian motion. Also $\forall\, 0 = t_1 \leq t_2 \leq ... \leq t_n$, we can write

$$\begin{pmatrix} X_{t_1} \\ X_{t_2} \\ \dots \\ X_{t_n} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} = \begin{pmatrix} X_{t_1} \\ X_{t_2} - X_{t_1} \\ \dots \\ X_{t_n} - X_{t_{n-1}} \end{pmatrix}$$

so we can obtain $X_t = \beta B_t$ where $\beta = \sqrt{\sum_{k=1}^{d}\sum_{i=1}^{d}\sum_{j=1}^{d}(\alpha_i\sigma_{ik})(\alpha_j\sigma_{jk})}$ and B_t is a standard Brownian motion.

□

According to equation (2.6), it is clear that

$$df(S_t) = \tilde{\mu}f(S_t)dt + \tilde{\sigma}f(S_t)dX_t \tag{2.7}$$

where $\tilde{\mu} = (\sum_{i=1}^{d}\alpha_i r + \frac{1}{2}(\sum_{i,j=1,i\neq j}^{d}\alpha_i\alpha_j + \sum_{i,j=1,i=j}^{d}\alpha_i(\alpha_i-1))\sum_{k=1}^{d}\sigma_{ik}\sigma_{jk})$
and $\tilde{\sigma} = 1$. Also $dX_t = \beta dB_t$, then we have

$$\frac{df(S_t)}{f(S_t)} = \tilde{\mu}dt + \beta dB_t \tag{2.8}$$

which is the one-dimensional reduced equation. Then

$$d\log(f(S_t)) = \frac{df(S_t)}{f(S_t)} - \frac{1}{2}\frac{\beta^2 f(S_t)^2}{f(S_t)^2}dt = \tilde{\mu}dt + \beta dB_t - \frac{1}{2}\beta^2 dt$$

which implies $\log(f(S_t)) = \log(f(S_0)) + (\tilde{\mu} - \frac{1}{2}\beta^2)t + \beta B_t$

so

$$f(S_t^{f(S_0),t}) = f(S_0)e^{(\tilde{\mu}-\frac{1}{2}\beta^2)(T-t)+\beta(B_T-B_t)} \quad \forall t\in[0,T]$$

where $f(S_t^{f(S_0),t})$ means $f(S_t) = f(S_0)$.

Now consider the equation on special case $\alpha_i = \frac{1}{d}\ \forall i$, $\sigma_{ij} = 0$ if $i \neq j$ and $\sigma_{ij} = \sigma$ if $i = j$. Then equation (2.8) becomes

$$\frac{df(S_t)}{f(S_t)} = (r + \frac{\sigma^2}{2d} - \frac{\sigma^2}{2})dt + \frac{\sigma}{\sqrt{d}}dB_t$$

therefore

$$f(S_t^{f(S_0),t}) = f(S_0)e^{(r-\frac{\sigma^2}{2})(T-t)+\frac{\sigma}{\sqrt{d}}(B_T-B_t))} \quad \forall t\in[0,T]$$

With the one-dimensional reduced equation, we can directly use Black-Scholes formula to calculate the price of European call and put option.

2.2 American Option

The main difference between American and European option is that the contract of American option allows the holder to exercise it at any time before the maturity. First observation we have is that the price of American option should be large or equal to the value of payoff function, i.e $V(S_t, t) \geq \Phi(S_t, t)$. Otherwise we will have chance to do arbitrage as we mention in the beginning.

Therefore the question is when will we exercise the option after purchasing it? Generally, we exercise the American option immediately when we find $\Phi(S_t, t) > V(S_t, t)$, and keep the option when $\Phi(S_t, t) \leq V(S_t, t)$ in order to make sure we will not lose money. Clearly the boundary between to exercise and to keep is $\partial = \{S_t \mid \Phi(S_t, t) = V(S_t, t)\}$ and this is what we are going to study.

2.2.1 Partial Differential Equation

If we want to calculate the price of American option from the Black-Scholes equation, we will need to solve

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

with boundary conditions

$$V(S_T, T) = \Phi(S_T, T),\ V(S_t, t) = \Phi(S_t, t)\ \forall S_t \in \partial$$

Again if the dimension of problem is not larger than three, then using finite difference method can do the computation efficiently with good accuracy. But there is still a bottleneck on high-dimensional problem and many researchers are still trying to find better solutions.

2.2.2 Monte Carlo Method

Another way to discuss American option pricing problem is to write it as an optimal stopping time problem like

$$V(S_0, 0) = \sup_{\tau \in [0,T]} \mathbb{E}[e^{-r\tau}\Phi(S_\tau, \tau)]$$

where we can take the supremum under the optimal stopping time

$$\widehat{\tau} = \inf\{t \geq 0 \mid S_t \in \Omega\}$$

where $\Omega = \{S_t \mid \Phi(S_t, t) > V(S_t, t)\}$

Numerically we can't present the continuous time, so in order to simulate the American option price, we will use Bermudan option to approximate it. One can check [9] for more theoretical results.

The Bermudan option is given $\mathbb{T} = \{t_i \leq T \mid i = 1, ..., N\}$ and the holder is allowed to exercise the option at the time $t \in \mathbb{T}$. We can find if the opportunity times we can do exercise is more, then the price we calculate should be more

close to the true American option price. So the American option price at time t_i can be approximated by

$$V(S_{t_i}, t_i) = \sup_{\tau \in \mathbb{T}} \mathbb{E}[e^{-r\tau}\Phi(S_\tau, \tau) \mid S_{t_i}]$$

An important information we have in the beginning is that we must do the exercise of American option at the maturity time T. It means we can construct a dynamical calculation by starting at time T back to the original time. So in our approximation, consider $t_N = T$, then we will have the following equations

$$V(S_{t_N}, t_N) = \Phi(S_{t_N}, t_N)$$

$$V(S_{t_i}) = \max(\Phi(S_{t_i}, t_i), \mathbb{E}[e^{-r(t_{i+1}-t_i)}V(S_{t_{i+1}}, t_{i+1}) \mid S_{t_i}])$$

it means we should exercise the option when

$$\Phi(S_{t_i}, t_i) > \mathbb{E}[e^{-r(t_{i+1}-t_i)}V(S_{t_{i+1}}, t_{i+1}) \mid S_{t_i}]$$

and keep it when

$$\Phi(S_{t_i}, t_i) \leq \mathbb{E}[e^{-r(t_{i+1}-t_i)}V(S_{t_{i+1}}, t_{i+1}) \mid S_{t_i}]$$

therefore, the problem now becomes how to solve the conditional expectation.

Longstaff Algorithm

Longstaff and Schwartz provided a method called LSM algorithm in [13] which assumes the conditional expectation belongs to L^2 functions, and they use least square methods to approximate it. It means

$$\mathbb{E}[e^{-r(t_{i+1}-t_i)}V(S_{t_{i+1}}, t_{i+1}) \mid S_{t_i}] = \sum_{j=1}^{\infty} a_j L_j(S_{t_i})$$

where L is the basis function with coefficient a. Basis functions can be chosen like Hermite, Legendre, Chebyshev polynomials, and in [13] they also suggest Laguerre polynomial.

Picazo's Algorithm

Instead of really solving the conditional expectation, Picazo provided another approach which is called Classification Monte Carlo (CMC) algorithm in [14]. The main idea of CMC algorithm is that consider

$$F(S_{t_i}, t_i) = \Phi(S_{t_i}, t_i) - \mathbb{E}[e^{-r(t_{i+1}-t_i)}V(S_{t_{i+1}}, t_{i+1}) \mid S_{t_i}]$$

which separates the domain into two regions. One is

$$\Omega(t) = \{S_t \mid \Phi(S_t, t) > V(S_t, t)\}$$

which means we should exercise the option and the other is

$$\Upsilon(t) = \{S_t \mid \Phi(S_t, t) \leq V(S_t, t)\}$$

which means we should keep the option.

They key point is if we can find a function $\widehat{F}$ which has the same sign as F and is easier to compute or control, then we can quickly decide whether we should exercise or keep the option at time $t \in \mathbb{T}$ by looking at the function $\widehat{F}$ which shows us the characterization of exercise region and keep region.

Once we have such characterization, we start to simulate paths from original time and at each time, if the asset price is in the exercise region, then we exercise the option and start the next path, and if the asset price is in the keep region, then we keep the option and continue to the next time until it is exercised or the maturity time T.

To achieve the function $\widehat{F}$, Picazo used the idea from Machine Learning like boosting algorithm. But the method is time-consuming. In Viet Dung Doan's thesis [15], he developed a parallel version of Picazo's algorithm and implemented it with grid computing. He showed in higher dimensional problem($d > 10$) , the result is better than Longstaff algorithm. In INRIA OASIS team, Michaël Benguigui is now extending the work into a combination of GPU computing and grid computing in order to separated the works into multi-GPUs. More details can be found in [16].

Chapter 3

Value-at-Risk for Option Book

From the previous chapter we study some fundamental theories and formulas to price options. Then we may consider a question: how much risk we should take after buying an option? This comes to discuss the problem of Value-at-Risk (VaR).

VaR is a value which estimates the biggest loss of the investment with a confidence. We can define the VaR with α confidence as the minimum value x which satisfies

$$\mathbb{P}(L > x) = 1 - \alpha$$

where L is the loss function which can be defined as $L = -\Delta V = V(S,t) - V(S + \Delta S, t + \Delta t)$. The value V can be price of stocks, options, or portfolio, and S is underlying assets. α is a confident level which satisfies $0 \leq \alpha \leq 1$.

VaR gives us the information that we will have α sure that we won't lose x in the next time Δt in our investment. In financial company, they always need to predict how much money they will need to prepare to avoid bankruptcy, or which deals they can make profits with lower risk. Therefore, VaR is a good reference.

Option book is a combination of the options we want to invest. The discussion on it is more complicated than a single option. In this chapter, we discuss how to calculate VaR for Geometric Average basket option, and the difficulty we have.

3.1 Histogram Method

First we may think if we have some historical data, how can we use it to calculate VaR? The answer is given by Histogram method which is an approach to construct the distribution of the loss function by historical data without any assumption of this distribution and parameters (see [18] for more discussions). The most important thing comes to mind is that if we use these data to predict

VaR, we have assumed that the asset returns are independent and identically distributed. It is a very strong assumption and we know in real market we don't consider it usually.

3.2 Analytic Method

Another approach is trying to study the information from the loss function L. If L is in a known distribution, like Gaussian distribution, then it is not so hard to find a numerical method to compute its quantile (see [17]). But we are interested in more complicated portfolio, then in such case, some other distributions are taken into account, like Student's t-distribution and Generalized Error distribution (see [3], [18]). With the purpose of having better measures of VaR, Engel in [19] introduced the Autoregressive Conditional Heteroscedasticity (ARCH) process which is a model widely used and many researchers extend it with different explanations.

3.3 Inverse Function Method

A small technique we can consider is using inverse function if we know our loss function well. The idea is if we can write the loss function as $L = f(S_{\Delta t})$, then

$$\mathbb{P}(L > x) = \mathbb{P}(f(S_{\Delta t}) > x)$$

Now we can try to inverse f to the right hand side as

$$\mathbb{P}(f(S_{\Delta t}) > x) = \mathbb{P}(S_{\Delta t} > f^{-1}(x))$$

and since $S_{\Delta t} = S_0 e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\sqrt{\Delta t}Z}$ where $Z \sim \mathcal{N}(0,1)$, so the equation can be written as

$$\mathbb{P}(S_{\Delta t} > f^{-1}(x)) = \mathbb{P}(Z > \frac{1}{\sqrt{\Delta t}}(log(\frac{f^{-1}(x)}{S_0}) - (r - \frac{1}{2}\sigma^2)\Delta t))$$

it shows now the left hand side is Gaussian law, so we have several methods to find the quantile.

3.4 Naive Method

In order to avoid taking too much assumptions, and we want to solve the problem even if with an unclear distribution of L, we introduce a very simple method to calculate VaR which we call Naive method. The Naive method states that we first simulate N samples on the loss function L, and then do sorting on these samples to be $\{L_1, L_2, ..., L_N\}$, where $L_1 \leq L_2 \leq ... \leq L_N$. Then the VaR of V with α confidence will be $L_{\alpha \times N}$.

For example. We have the exact formula for the price of one-dimensional European option. Then first we generate a set $S_{\Delta t} = \{S^1_{\Delta t}, ..., S^N_{\Delta t}\}$, then for

each $S^i_{\Delta t}$, $i = 1, ..., N$, we can use the closed form solution to get the price of option $V(S^i_{\Delta t}, \Delta t)$. Therefore the loss function is equal to

$$L_i = V(S_0, 0) - V(S^i_{\Delta t}, \Delta t)$$

then we do the sorting on L and then VaR with α confidence is $L_{\alpha \times N}$.

3.4.1 Monte Carlo Simulation

• **single option**

But there are many cases that we don't have exact formula for pricing option. So we need to do Monte Carlo simulation. The idea of Monte Carlo simulation on VaR computation is that we first simulate paths from S_0 to $S_{\Delta t}$, then for each $S_{\Delta t}$, we use it to simulate paths to S_T and put it into payoff function for Monte Carlo method. We can see the following figure:

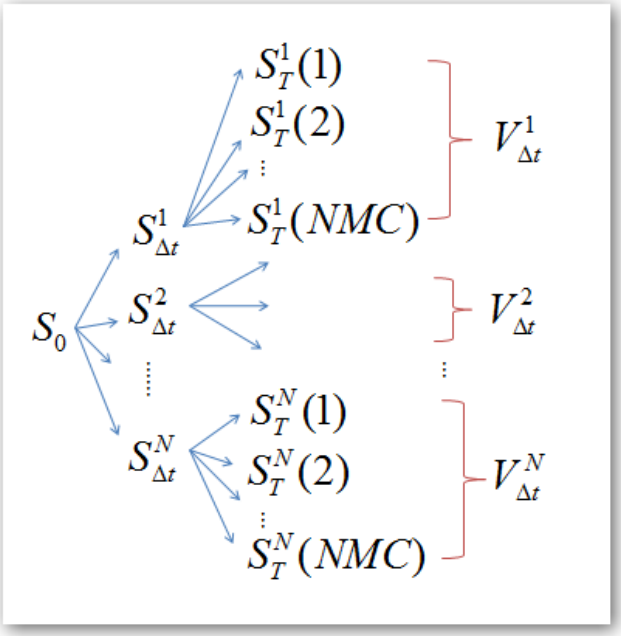


Figure 3.1: The framework to calculate VaR of option by Monte Carlo method

To demonstrate the idea easily, we create a matrix form as notation as

$$S_0 \to \begin{pmatrix} S^1_T(1) & S^1_T(2) & \dots & S^1_T(NMC) \\ S^2_T(1) & S^2_T(2) & \dots & S^2_T(NMC) \\ \dots & \dots & \dots & \dots \\ S^N_T(1) & S^N_T(2) & \dots & S^N_T(NMC) \end{pmatrix} \to \begin{pmatrix} V^1_{\Delta t} \\ V^2_{\Delta t} \\ \dots \\ V^N_{\Delta t} \end{pmatrix}$$

and then we can calculate the loss function and do sorting to get VaR

$$\begin{pmatrix} V_{\Delta t}^1 \\ V_{\Delta t}^2 \\ \dots \\ V_{\Delta t}^N \end{pmatrix} \to \begin{pmatrix} L_1 \\ L_2 \\ \dots \\ L_N \end{pmatrix} \to VaR$$

• **basket option**

Now we are going to extend the work to basket option. Consider in basket option, we have $S = (S^1, ..., S^d)$ d assets. It means the we will have to do the framework in Figure 3.1 for d times. Actually we first need to simulate

$$S_0^i \to \begin{pmatrix} S_T^{i,1}(1) & S_T^{i,1}(2) & \dots & S_T^{i,1}(NMC) \\ S_T^{i,2}(1) & S_T^{i,2}(2) & \dots & S_T^{i,2}(NMC) \\ \dots & \dots & \dots & \dots \\ S_T^{i,N}(1) & S_T^{i,N}(2) & \dots & S_T^{i,N}(NMC) \end{pmatrix} = A^i \ \ \forall i = 1, ..., d.$$

then we have to do geometric average to all matrices on each entry. It means

$$S_{ij}^{GA} = \prod_{k=1}^{d} (A_{ij}^k)^{1/d} \ \ \forall i = 1, ..., N; \ j = 1, ..., NMC.$$

after getting the matrix S^{GA}, the rest calculations are same as before as

$$S^{GA} \to \begin{pmatrix} V_{\Delta t}^1 \\ V_{\Delta t}^2 \\ \dots \\ V_{\Delta t}^N \end{pmatrix} \to \begin{pmatrix} L_1 \\ L_2 \\ \dots \\ L_N \end{pmatrix} \to VaR$$

• **option book**

Eventually our main purpose is to calculate VaR for option book. The calculation is similar as for single option or basket option because we can separate the work of calculating the loss function of option book into each option. In other word, if $L^1, L^2, ..., L^m$ are loss functions of m options in a option book, then $L = \sum_{i=1}^{m} L_i$ is the loss function of option book.

3.4.2 Delta-Gamma-Theta Approach

A possible approach to sample L fast is we can try to approximate the loss function with a quadratic function. First we do Taylor expansion on $L = -\Delta V$ and obtain

$$L = -\Delta V \approx -\frac{\partial V}{\partial t}\Delta t - \sum_{i=1}^{n} \frac{\partial V}{\partial S_i}\Delta S - \frac{1}{2}\sum_{i,j=1}^{n} \frac{\partial^2 V}{\partial S_i \partial S_j}\Delta S_i \Delta S_j$$

which is equal to

$$-\Theta \times \Delta t - \Delta \times \Delta S - \frac{1}{2}\Gamma \times (\Delta S)^2$$

Here we can observe that if we know how to compute the Greeks, then what we need to do becomes only to simulate $\Delta S = S_{\Delta t} - S_0$. It reduces the complexity of the whole computation.

But there are some constrains on using this kind of approximation and we have to be careful. First the loss function should be like a monotone function. Second, the time Δt can not be large. Hence usually we use Delta-Gamma-Theta approach to calculate VaR for Δt which is small. The following is a comparison on the loss function and using Delta-Gamma-Theta approach to approximate it:

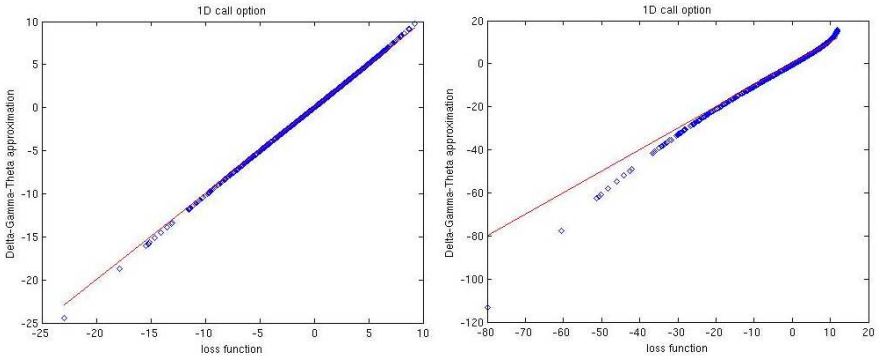


Figure 3.2: A comparison on one-dimensional call option. Left figure is with Δt = 0.08333, and right figure is with $\Delta t = 0.5$. The blue diamond is (loss, Delta-Gamma-Theta approximation) and red line is a line with slope 1 as reference

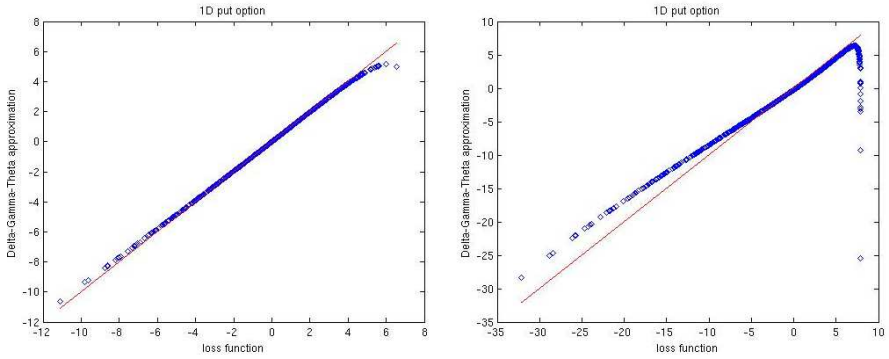


Figure 3.3: A comparison on dimensional put option. Left figure is with Δt = 0.08333, and right figure is with $\Delta t = 0.5$. The blue diamond is (loss, Delta-Gamma-Theta approximation) and red line is a line with slope 1 as reference

Greeks Computation

If we want to use Delta-Gamma-Theta approach, another thing we have to be careful is the Greeks. In general case, we don't have formula for Greeks, so we have to calculate it numerically. There are several ways to do Greeks computation, and we pick up finite difference method and Malliavin calculus

method to do a comparison. More details on deriving the formula or theoretical results can be found in [11], [20].

The formula for finite difference method is as following:

$$\Delta_0 = \frac{V(S+h,0) - V(S-h,0)}{2h}$$

$$\Gamma_0 = \frac{V(S+h,0) - 2V(S,0) + V(S-h,0)}{h^2}$$

$$\Theta_0 = \frac{V(S,0+\Delta t) - V(S,0-\Delta t)}{2\Delta t}$$

where we evaluate V by Monte Carlo method.

And the formula for Malliavin Calculus is as following:

$$\Delta_0 = \mathbb{E}[e^{-rT}\phi(S_T)\frac{W_T}{S\sigma T}]$$

$$\Gamma_0 = \mathbb{E}[e^{-rT}\phi(S_T)\frac{1}{S^2\sigma T}(\frac{W_T^2}{\sigma T} - W_T - \frac{1}{\sigma})]$$

where we evaluate the expectation by Monte Carlo method.

We now give an example to show the numerical results for both methods:

Example 3.4.1. *Calculate Greeks for one-dimensional call option with parameter $S_0 = 100, K = 100, r = 0.04, \sigma = 0.25, t = 0, T = 1$. We know the true solutions of Greek for call option are Delta = 0.612177, Gamma = 0.015322, and we set $h = 0.01$ for finite difference method.*

NMC	Delta(FD)	Delta(Malliavin)	Gamma(FD)	Gamma(Malliavin)
10^4	0.615564	0.600828	0.001226	0.014418
10^5	0.612398	0.616514	0.012577	0.015545
10^6	0.612611	0.611674	0.014915	0.015212
10^7	0.612450	0.612273	0.015109	0.015320
10^8	0.612233	0.612081	0.015279	0.015319

The time for calculating Greeks (calculating 100 times and take average):

Time (sec)	NMC = 10^6	NMC = 10^7	NMC = 10^8
finite difference method	0.0215s	0.1465s	1.6280s
Malliavin calculus	0.0073s	0.0715s	0.6855s

We compare the variance on both methods for call option in the following figures:

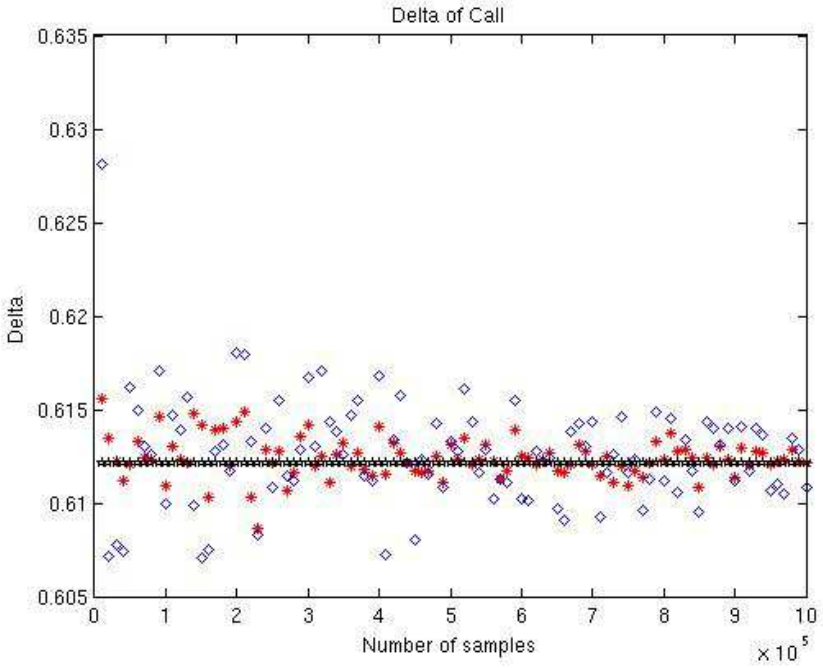


Figure 3.4: Red stars are finite difference method. Blue diamonds are Malliavin calculus.

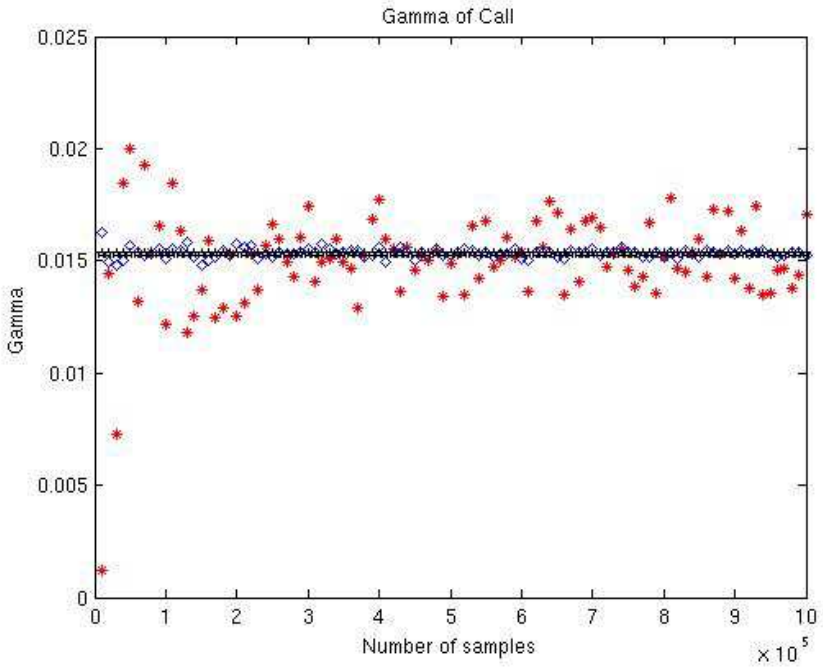


Figure 3.5: Red stars are finite difference method. Blue diamonds are Malliavin calculus.

From figure 3.2 and 3.3, we can observe the variance for Gamma calculated by finite difference method is big. The reason is we have to multiply h^{-2} which can enlarge the error from the calculation. Also the variance for Delta performed by Malliavin calculus is larger than by finite difference method. Actually there is a method in [20] to reduce the variance calculated by Malliavin calculus and the formula is

$$\Delta_0 = \mathbb{E}[e^{-rT} H_\delta(S_T) \frac{S_T}{S}] + \mathbb{E}[e^{-rT} F_\delta(S_T) \frac{W_T}{S\sigma T}]$$

where

$$\begin{aligned} H_\delta(s) &= 0 \;\; if \;\; s \leq K - \delta \\ &= \frac{s - (K - \delta)}{2\delta} \;\; if \;\; K - \delta \leq s \leq K + \delta \\ &= 1 \;\; if \;\; K + \delta \leq s \end{aligned}$$

and

$$F_\delta(t) = \{t - K\}^+ - \int_{-\infty}^{t} H_\delta(s) ds$$

Chapter 4

Importance Sampling and Importance Splitting Methods

For Monte Carlo method, usually when we have an estimator, we will find some strategies to reduce the variance since the convergent rate for estimator depends on $O(\sqrt{\frac{\sigma^2}{NMC}})$. This kind of methods called variance reduction techniques which give us a way to achieve good accuracy without simulating too much points.

So now we modify our problem a little bit to calculate the probability

$$\mathbb{P}(L > x)$$

with given L and x. Then we want to look for methods to calculate this probability efficiently.

In this chapter, we study two techniques, importance sampling and importance splitting methods, to see how to use it. Importance sampling is a useful technique to calculate the probability $\mathbb{P}(L > x)$ by changing of measure and we can do sampling on the region which we think is most important.

Another method, importance splitting method, is designed for calculating rare event (see [6], [7]). Rare event is an event which happens with very low probability. We can expect if we use Crude Monte Carlo to estimate the probability:

$$\widehat{p} = \frac{1}{NMC} \sum_{i=1}^{NMC} \mathbb{1}_{\{L>x\}}$$

the relative error will be $\frac{1}{\sqrt{NMC}} \frac{\sqrt{p-p^2}}{p}$. Then if p is small and closed to 0, we will have problem to estimate it since

$$\lim_{p\to 0} \frac{1}{\sqrt{NMC}} \frac{\sqrt{p-p^2}}{p} = \lim_{p\to 0} \frac{1}{\sqrt{NMCp}} = +\infty$$

so we have to choose other method such as importance splitting method.

Finally, our main point is to calculate VaR, so we design different search algorithms to help us to compute VaR after we use importance sampling methods and importance splitting method to evaluate the probability.

4.1 Importance Sampling Method

Importance sampling is the most typical method in variance reduction technique and there are many discussions on how to apply it to VaR computation in [3], [4], [5]. The idea of importance sampling is as following: Given X is a random variable with probability density function f, and $h : \mathbb{R} \longrightarrow \mathbb{R}$. Then

$$\mathbb{E}[h(X)] = \int h(x)f(x)dx = p$$

by Monte Carlo method, it can be estimated by $\widehat{p} = \frac{1}{NMC}\sum_{i=1}^{NMC} h(X_i)$.

Now we want to focus on some more important regions when we do the Monte Carlo method by changing of measure. We use a new estimator

$$\widehat{p_{im}} = \frac{1}{NMC}\sum_{i=1}^{NMC} h(X_i)\frac{f(X_i)}{g(X_i)}$$

where g is a new probability density function and satisfies $g(x) > 0 \quad \forall x$. It is because we have

$$\widehat{\mathbb{E}[p_{im}]} = \int h(x)\frac{f(x)}{g(x)}g(x)dx = \int h(x)f(x)dx = p$$

4.1.1 Exponential Twisting

Hence the main point is to find the change of measure. Before calculating it, first we use Delta-Gamma-Theta approach to approximate the loss function. By Taylor's expansion, we have

$$L = -\Delta V \approx -\frac{\partial V}{\partial t}\Delta t - \sum_{i=1}^{n}\frac{\partial V}{\partial S_i}\Delta S - \frac{1}{2}\sum_{i,j=1}^{n}\frac{\partial^2 V}{\partial S_i \partial S_j}\Delta S_i \Delta S_j$$

which is equal to

$$-\Theta \times \Delta t - \Delta \times \Delta S - \frac{1}{2}\Gamma \times (\Delta S)^2$$

We follow the idea from [3]. Assume that Δt is small, so $\Delta S \sim N(0, \Sigma)$, and then we can write $\Delta S = CZ, Z \sim N(0, I)$ where $CC^\top = \Sigma$. Then L becomes

$$L \approx -\Theta\Delta t - (C^\top \Delta)^\top Z - \frac{1}{2}Z^\top (C^\top \Gamma C)Z$$

We try to make the problem be easier by selecting C to be a triangular matrix. We can do it because Σ is symmetric positive definite matrix, and by

Cholesky decomposition, $\Sigma = \widehat{C}\widehat{C}^\top$ where $\widehat{C}$ is lower triangular matrix. Then, $\frac{1}{2}\widehat{C}^\top\Gamma\widehat{C}$ is symmetric and

$$\frac{1}{2}\widehat{C}^\top\Gamma\widehat{C} = U\Lambda U^\top$$

where

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & 0 & \lambda_n \end{pmatrix}$$

now let $C = \widehat{C}U$, then we have

$$CC^\top = \widehat{C}UU^\top\widehat{C}^\top = \Sigma$$

and

$$-\frac{1}{2}C^\top\Gamma C = \frac{1}{2}U^\top\widehat{C}^\top\Gamma\widehat{C}U = U^\top U\Lambda U^\top U = \Lambda$$

it means we can choose C to be triangular matrix. To simplify the notation, let $a = -\Theta\Delta t, b = -C^\top\Delta$, then

$$L \approx Q := a + b^\top Z + Z^\top\Lambda Z = a + \sum_{i=1}^{n}(b_iZ_i + \lambda_iZ_i^2) = a + \sum_{i=1}^{n}(\lambda_i(Z_i + \frac{b_i}{2\lambda_i})^2 - \frac{b_i^2}{4\lambda_i})$$

which shows that we only need to generate Z and to the calculation for Q.

Now we are going to calculate the change of measure. First we give some definitions. Let F, F_θ be cumulative functions.

Definition 4.1.1. The moment generating function of random variable X is given by $M_X(\theta) = \mathbb{E}[e^{\theta X}]$. The cumulant generating function is given by $\psi(\theta) = log(M_X(\theta))$. Let f_θ, f be density functions of F_θ and F. Then a transformation from F to F_θ given by $f_\theta(x) = e^{\theta x - \psi(\theta)}f(x)$ is called exponential twisting.

Then the change of measure can be written as

$$\frac{dF(X)}{dF_\theta(X)} = e^{-\theta X + \psi(\theta)}$$

in order to get it, first we calculate the moment generating function of L

$$\mathbb{E}[e^{\theta Q}] = e^{\theta a}\prod_{i=1}^{n}\mathbb{E}[e^{\theta(\lambda_i(Z_i + \frac{b_i}{2\lambda_i})^2 - \frac{b_i^2}{4\lambda_i})}] = e^{\theta a}\prod_{i=1}^{n}e^{-\theta\frac{b_i^2}{4\lambda_i}}\mathbb{E}[e^{\theta\lambda_i(Z_i + \frac{b_i}{2\lambda_i})^2}]$$

here we use $(Z_i + \frac{b_i}{2\lambda_i})^2$ is χ^2-distribution, then

$$\mathbb{E}[e^{(\theta\lambda_i(Z_i + \frac{b_i}{2\lambda_i})^2)}] = \frac{e^{\frac{\theta\frac{b_i^2}{4\lambda_i}}{1-2\theta\lambda_i}}}{\sqrt{(1-2\theta\lambda_i)}} \quad \forall\ \theta\lambda_i < \frac{1}{2}$$

and

$$\psi(\theta) = log(\mathbb{E}[e^{\theta Q}]) = \theta a + \frac{1}{2}\sum_{i=1}^{n}(\frac{\theta^2 b_i^2}{(1-2\theta\lambda_i)} - log(1-2\theta\lambda_i))$$

therefore, the new probability measure P_θ can be defined by

$$\frac{dP}{dP_\theta} = e^{-\theta L + \psi(\theta)}$$

it shows the new estimator for $\mathbb{P}(L > x)$ is

$$\widehat{p_{im}} = \frac{1}{NMC} \sum_{i=1}^{NMC} \mathbb{1}_{\{L_i > x\}} e^{-\theta Q_i + \psi(\theta)} \tag{4.1}$$

where under the new measure P_θ, $Z \sim N(\mu(\theta), \Sigma(\theta))$, $\mu_i(\theta) = \frac{\theta b_i}{1-2\lambda_i\theta}$ and $\Sigma(\theta)$ is a diagonal matrix with entries $\sigma_{ii}^2(\theta) = \frac{1}{1-2\lambda_i\theta}$

Finally, the choice of θ is also important. We want to find θ to make $-\theta x + \psi(\theta)$ be minimum. Because we use Delta-Gamma-Theta approach to approximate the loss function, in other word, we use a quadratic function to approximate it, then $\psi(\theta)$ is a convex function. Then if we try to minimize $-\theta x + \psi(\theta)$, it is same to find condition that $-x + \psi'(\theta) = 0$ which is an optimization problem.

4.1.2 From Importance Sampling to VaR

So far, the importance sampling gives us a way to calculate the probability

$$\mathbb{P}(L > x)$$

with given L and x and without sampling too much points. However, our main point is to find x such that $\mathbb{P}(L > x) = 1 - \alpha$ with a given α. Hence once we want to use the importance sampling method, we need to do a for loop for x outside the computation. The algorithm is as following:

Algorithm 2: Importance Sampling Method for Calculating VaR

Input: $S_0, r, \sigma, K, TOL, NMC, a, b$
Output: VaR
calculate Δ, Γ, Θ of loss function with $Z \approx (0, \Sigma)$ and solve the optimization problem for θ
for $x \in [a, b]$ **do**
  sum = 0;
  for $i = 1 : NMC$ **do**
    (1)simulate $Z_i \sim N(\mu(\theta), \Sigma(\theta))$, and then calculate $e^{-\theta Q_i + \psi(\theta)}$
    (2)simulate L_i
    (3)evaluate sum = sum + $e^{-\theta Q_i + \psi(\theta)} \mathbb{1}(L_i > x) + a$
  end
  $\mathbb{P}(L > x) = \frac{sum}{NMC}$
  if $|\mathbb{P}(Y < f(x)) - (1 - \alpha)| < TOL$ **then**
    VaR = x;
    break;
  end
end

Example 4.1.1. *Calculate probability for one-dimensional call and put option with parameter* $S_0 = 100, K = 100, r = 0.04, \sigma = 0.25, \Delta t = 0.0833, T = 1$

For call option, we know VaR with 95% confidence is equal to 6.2752. So we take x = 6.2752 and L is the loss function of call option to estimate $\mathbb{P}(L > x) = \alpha$ and we can expect α should be equal to 0.05. We compare the result with and without importance sampling method.

call option

NMC	α	Variance	α with IP	Variance
10^2	0.0400	17.5352	0.0506	3.8301
10^3	0.0470	21.8970	0.0506	3.9554
10^4	0.0469	20.1830	0.0502	4.0584
10^5	0.0511	20.2098	0.0500	3.9653
10^6	0.0502	20.2848	0.0500	3.9661

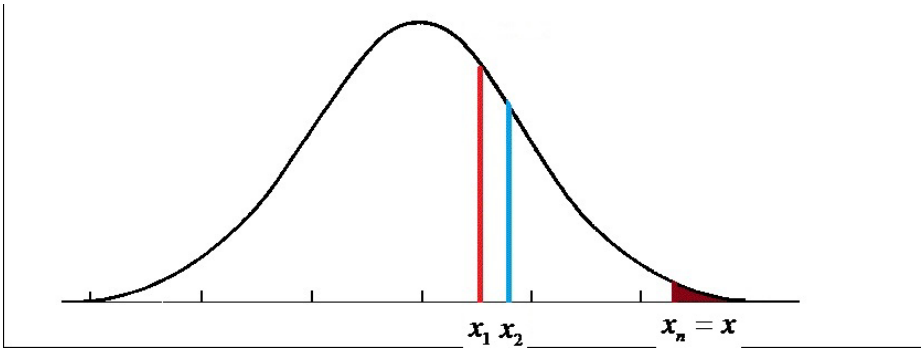
similarly for put option, we know VaR with 95% confidence is equal to 4.0694. So we take x = 4.0694 and L is the loss function of put option to estimate $\mathbb{P}(L > x) = \alpha$ and we can expect α should be equal to 0.05. We compare the result with and without importance sampling method.

put option

NMC	α	Variance	α with IP	Variance
10^2	0.0100	8.5454	0.0396	0.5852
10^3	0.0380	9.1216	0.0510	0.6452
10^4	0.0488	8.0039	0.0499	0.6050
10^5	0.0506	7.9979	0.0498	0.5939
10^6	0.0504	8.0564	0.0498	0.5932

4.2 Importance Splitting Method

As we mention in the beginning, importance splitting method is designed to calculate the probability of rare event. The idea of importance splitting method is the following picture:



Consider $A_k = \{y|L(y) > x_k\}$ and define $\mathbb{R} = A_0 \supset A_1 \supset ... \supset A_{n-1} \supset A_n = A$ where $A = \{y|L(y) > x\}$ and we have $\mathbb{P}(L(Y) > x) = \mathbb{P}(Y \in A)$. The

probability $\mathbb{P}(Y \in A)$ can be written in

$$\mathbb{P}(Y \in A) = \prod_{k=1}^{n} \mathbb{P}(Y \in A_k | Y \in A_{k-1})$$

Now the problem is how to choose A_k? In [6], [7], there are two possibilities. One is called fixed level method which sets up the level from A_1 to A_n uniformly. The other is called adaptive method in which A_k will converge to A_n in an adaptive way. We will focus on using adaptive method and using the following algorithm from [6]:

Algorithm 3: Importance Splitting for Calculating the Probability

Input: $S_0, r, \sigma, K, N_k, NMC, x$
Output: $\mathbb{P}(L > x)$
k = 0;
(1)Generate $L_1^{(k)}, L_2^{(k)} ..., L_{N_k}^{(k)}$ from $f_k(L)$;
(2)calculate the quantile $q_\alpha^{(k)}$;
(3)$A_{k+1} = \{X \in \mathbb{R} \mid L(X) > q_\alpha^{(k)}\}$;
if $q_\alpha^{(k)} < x$ **then**
  k = k + 1;
  go back to (2);
end
$\mathbb{P} = (1-\alpha)^k \times \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbb{1}_{L(X_i^{(k)}) > x}$

The algorithm is easy to implement, however, using f_k which is restricted density on A_k to generate L is a problem if we don't know the exact law of it. Generally, many papers suggest to use Metropolis-Hastings algorithm (see [6], [7], [31]) to deal with it. And if we know $X \sim \mathcal{N}(0_d, I)$ is Gaussian, then we have the formula

$$\frac{X + c\mathcal{N}(0_d, I)}{\sqrt{1+c^2}}$$

4.2.1 From Importance Splitting to VaR

Again, the importance splitting provides a way to calculate the probability

$$\mathbb{P}(L > x)$$

with given L and x. To find x, we still need to do a for loop for x outside the computation. The algorithm is as following:

Algorithm 4: Importance Splitting Method for VaR

Input: $S_0, r, \sigma, K, N_k, NMC, x, a, b, TOL$
Output: $\mathbb{P}(L > x)$
for $x \in [a, b]$ **do**
  k = 0;
  (1)Generate $L_1^{(k)}, L_2^{(k)}..., L_{N_k}^{(k)}$ from $f_k(L)$;
  (2)calculate the quantile $q_\alpha^{(k)}$;
  (3)$A_{k+1} = \{X \in \mathbb{R} \mid L(X) > q_\alpha^{(k)}\}$;
  if $q_\alpha^{(k)} < x$ **then**
    k = k + 1;
    go back to (2);
  end
  $\mathbb{P} = (1-\alpha)^k \times \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbb{1}_{L(X_i^{(k)})>x}$;
  if $|\mathbb{P} - (1-\alpha)| < TOL$ **then**
    x = VaR ;
    break ;
  end
end

4.3 Search Algorithm

Finally we give methods which can help us to find VaR quickly from the given α and L.

4.3.1 Normal Search

A very naive way to do it is to test all the points in a given interval $[a, b]$ like the following algorithm:

Algorithm 5: Naive Search Algorithm

Input: a, b, TOL, NMC
Output: VaR
for $x \in [a, b]$ **do**
  Calculate the loss function;
  $\mathbb{P}(L > x) = \frac{1}{NMC} \Sigma_{j=1}^{n} \mathbb{1}_{\{L_i > x\}}$;
  if $|\mathbb{P}(L > x) - 0.05| < TOL$ **then**
    VaR = x;
    break;
  end
end

4.3.2 Bisection Search

However, if the VaR we are going to search is close to b, then we will waste much time on looking for it. In order to avoid it, we use the idea of bisection method to help on searching VaR.

Algorithm 6: Bisection Search Algorithm

Input: a, b, n, TOL, NMC
Output: VaR
for $i = 1 : n$ **do**
  Ave = $\frac{a+b}{2}$;
  Calculate the Loss function;
  $\mathbb{P}(L > ave) = \frac{1}{N}\Sigma_{j=1}^{n}\mathbb{1}_{\{L_i > ave\}}$;
  if $|\mathbb{P}(L > ave) - 0.05| < TOL$ **then**
    VaR = ave;
    break;
    else
      if $\mathbb{P}(L > a) \times \mathbb{P}(L > ave) > 0$ **then**
        a = ave;
        else
          b = ave;
        end
      end
    end
  end
end

4.3.3 Newton's Search

Now if we consider $\mathbb{P}(L > x) = f(x)$. Then the problem to find VaR becomes to solve $f(x) - (1 - \alpha) = 0$. Since $f(x)$ is a monotone decreasing function, we can use Newton's method to help us solving this equation.

Algorithm 7: Newton's Search Algorithm

Input: a, n, TOL, NMC, α
Output: VaR
$z_0 = a$;
for $i = 1 : n$ **do**
  $f(x) = \mathbb{P}(L > z_0)$ - (1 - α);
  $z_i = z_{i-1} - \frac{f(z_{i-1})}{f'(z_{i-1})}$;
  if $|z_i - z_{i-1}| < TOL$ **then**
    VaR = z_i;
    break;
  end
end

Here is a simple comparison for calculating VaR with importance sampling method and different search algorithms. We search from a = 5.5 to b = 10, TOL = 10^{-4}. For normal search we discretize [a,b] with grid 0.0001. For Newton's search we set h = 0.01 for finite difference method to calculate the derivative. We calculate VaR for one-dimensional call option with different NMC and the time is calculated 10 times and take average:

NMC	Naive Search	bisection Search	Newton's Search
10^5	204.5832s	0.6286s	1.0363s
10^6	1557.0482s	3.5954s	1.0741s
10^7	—	34.1494s	8.4561s

There are still some variance reduction techniques discussed in [3], [4], [5] which can be used also in calculating VaR. In [6], [7], there are details for theoretical results on using importance splitting method when the probability we concern is very small. However, more or less in these methods we have to give assumptions on parameters or functions in the beginning. In general we want a method without too many restrictions when we use it. Hence in the next chapter we are going to discuss how to do it.

Chapter 5

Background of GPU Computing

So far we have tried several methods which are from mathematics to help reducing the time of computation. In fact, we choose Monte Carlo method for our main simulation, so there is a potential that we can combine with parallel computing. Then ths question now is how can we implement our idea into programming? There are several ways to do that, and here we choose GPU computing.

GPU computing becomes popular nowadays because GPUs are powerful tools to implement parallel computing. If we know which part in our algorithms can be done independently, then we can assign threads and blocks from GPU to do that. In order to have a background of how to program it, in this chapter, we give a short introduction from [21], [22], [23] on what is GPU computing, how can we start to program it and the reason why we choose to use GPUs rather than CPUs for our topic.

5.1 Overview of GPUs

Graphics processing units(GPUs) are devices which are designed to to some basic operations to CPUs in our machine, like manipulating the memory to create the image on screen. Today we can see the high definition video in most games, and it is not surprising that they are using such devices in the game console like PlayStaion 3 and Xbox 360 in order to have higher visual quality.

With a potential to do parallel computing on GPUs, NVIDIA released CUDA in November 2006, which is a platform that allows developers to use the parallel compute engine in NVIDIA GPUs to solve many computational problems, which need to be done in long time, in a more efficient way. Developers can use the most standard programming language, C language, to get into the GPU computing, and CUDA is also designed to support other languages like FORTRAN, JAVA, DirectCompute, OpenACC.

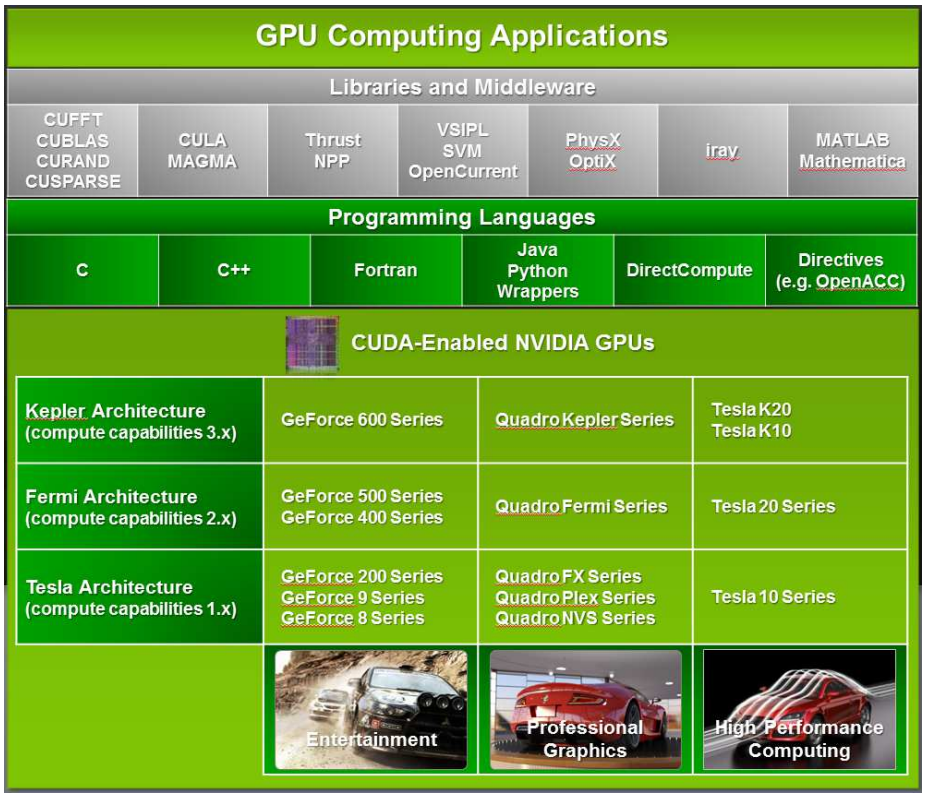


Figure 5.1: Figure from [21] shows the supported languages and applications by CUDA.

5.2 CPUs and GPUs

Now we may ask ourselves: when should we use CUDA programming? Should we do every tasks on GPUs instead of CPUs? The answer is no. Usually CPUs are specialized used to compute problems which is complex. But if for problem which is computed intensively and can be expressed as data-parallel computation, GPUs then become a better choice.

The reason is that in GPUs, there are more transistors which are used to data processing instead of data caching and flow control. Then we can imagine, when we execute a program, it is run on many data elements in parallel without too much flow control since the program is executed in each data element, and we don't need to care about the latency of memory access in doing data caches since we are doing the calculation in many data elements simultaneously.

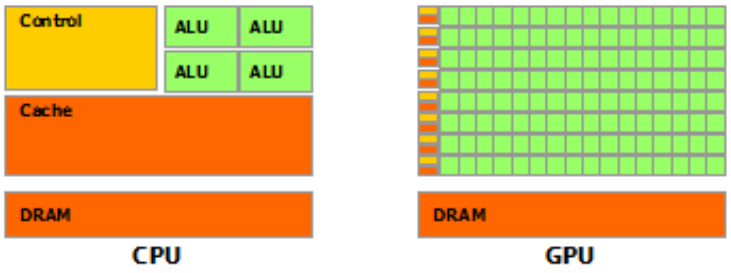


Figure 5.2: Figure from [21] shows the transistors for data processing

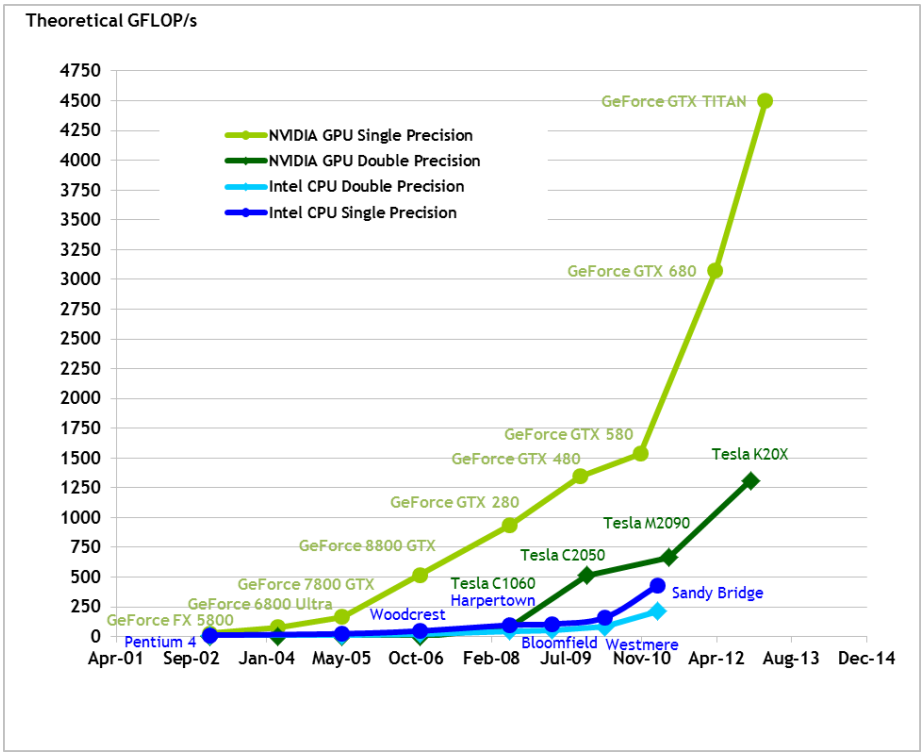


Figure 5.3: Figure from [21] shows the difference of computation bound between GPUs and CPUs

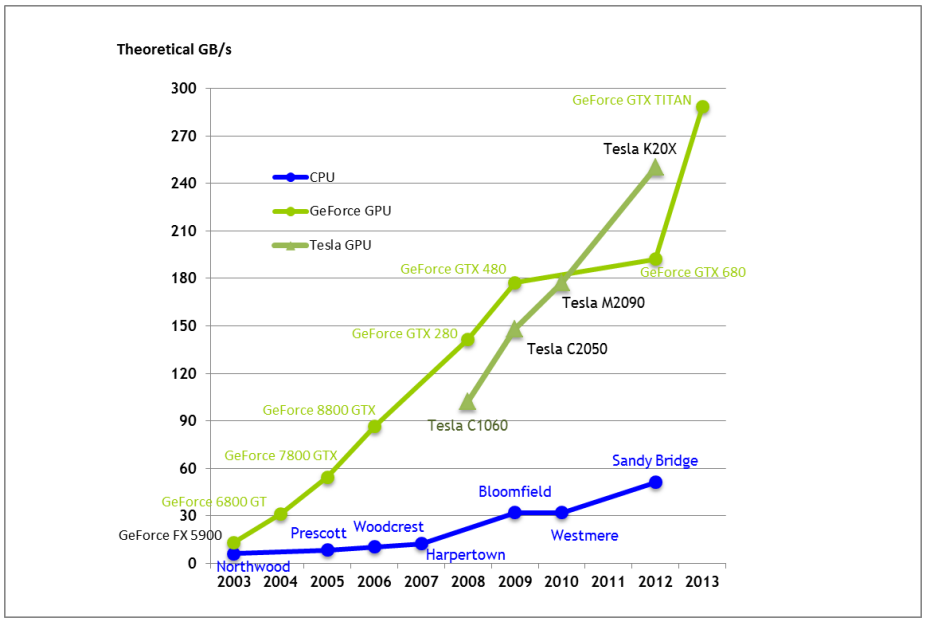


Figure 5.4: Figure from [21] shows the difference of memory bound between GPUs and CPUs

5.3 Threads, Grids and Blocks

As we mention, CUDA programming can be done in C language. There are two ways to do that. The first way is using the library or API released in CUDA SDK. There are many useful libraries to solve mathematical problem like numerical linear algebra, or to do random number generation. We will give an introduction to the libraries we select to use in the next chapter.

The second way is to write a *kernel function* which will be executed in parallel in different threads when we call it. Thread is the most fundamental element in GPUs, A block is arranged by individual threads, and grid contains several blocks.

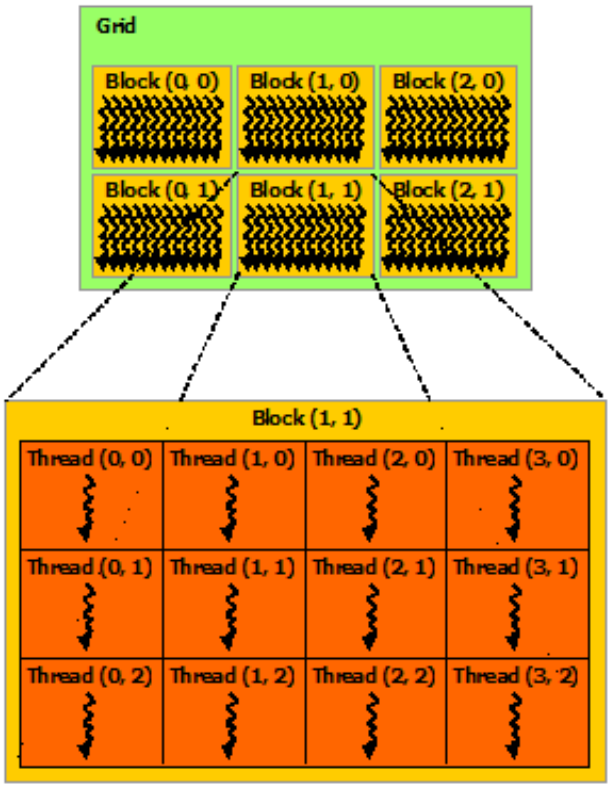


Figure 5.5: Figure from [21] shows the relations of thread, grid, and block

But, the number of threads per block is limited. It means we can not assign too much threads to afford the parallel jobs because threads and blocks stay on the same processor and share the memory of the core. Usually there are 1024 threads in one block on GPUs we use now. A good news is that the kernel function can be executed by multiple equally-shaped thread blocks, so it means the total numbers of threads we can assign to work each time is number of thread per block $\times$ number of block. Each block in a grid can be defined as one-dimensional, two-dimensional, or three-dimensional index.

There are many kinds of memories in GPUs. Two of the most important memories are global memory which is accessible to all host systems, and shared memory which can be accessed in each processor. Threads can be communicated in the same block by using shared memory, and also can access in global memory. So select a good size of dimensions for grid and blocks is important since shared memory is faster than global memory. The memory allocation is a crucial problem we need to face to do GPU computing. In our numerical case, we will try everything on global memory first to be the benchmark and see how much improvement we can do.

5.4 GPUs on VaR Computation

To finish this chapter, we have to answer a question: why we choose to use GPUs to calculate VaR. The answer can be found in figure 5.3 and 5.4 which are theoretical results from NVIDIA. Figure 5.3 shows us that if the main job of our problem is computation, then GPUs will have around 5 times better on GFLOP/s than CPUs. Figure 5.4 gives another information that if now the main task is dealing with memory on data transmitting, then GPUs will have around 4 times better on GB/s than CPUs. Therefore, we can first check the works in our VaR computation belong to which group, and then to see theoretically how much better we can achieve.

Fortunately, our VaR computation can be separated into four parts as following:

1. Random Number Generation

2. Geometric Average

3. Monte Carlo Reduction

4. Sorting

and for 2, 3, 4 all belong to the group to deal with memory. 1 can both be in group of computational bound and memory bound since it depends on the algorithm and implementation we choose. Then apparently, we will have better performance to do computation on GPUs than CPUs. More details are going to be discussed in the following chapter.

Chapter 6

Parallel Strategy

In this chapter, we are going to see how to use GPU computing to calculate VaR for option book. Generally the option book contains European and American options. We choose European basket options to be our numerical experiment in order to get a benchmark and to see how to extend the result for the general case.

We choose the Naive method with Monte Carlo approach introduced in chapter 3 because there are many calculations which can be implemented with parallel computing. As we discuss in chapter 5, the whole computation can be separated into four parts. We are going to see how to implement these parts in GPU computing.

6.1 Random Number Generation

First of all, when we do some stochastic approximations with Monte Carlo method, we need to generate random numbers with good quality and many algorithms and generators are developed to provide the user good psuedorandom numbers. A very famous library for people to test whether the generator they develop or use is good is called *TestU01* (see [24], [25]). We can choose the generator which passes the examination of *TestU01* for our problem.

To develop a random number generators in parallel is a big task, so we don't give details on the implementation. Since there exists a library called *CURAND* provided by CUDA which gives us different choices on generators and they all pass the *TestU01*, and we can choose to generate the random numbers on CPUs or GPUs. Hence we can use the library directly. One can check more details in [26]. Here is an example to use Host API to generate normal random numbers on device:

```
curandGenerator_t gen;
float *devData;
cudaMalloc((void**)&devData, N*sizeof(float));
curandCreateGenerator(&gen,CURAND_RNG_PSEUDO_MTGP32);
curandSetPseudoRandomGeneratorSeed(gen,1234ULL);
```

```
curandGenerateNormal(gen,devData,N,mean,variance);
```

Actually for psuedorandom numbers, *CURAND* provides three different generators. They are *XORWOW* which is implemented by XORWOW algorithm, *MRG32k3a* which is a member of the Combined Multiple Recursive family developed by Pierre L'Ecuyer, and *MTGP32* is a member of the Mersenne Twister family developed by Makoto Matsumoto and Takuji Nishimura. We give a comparison on using these three generators to calculate the price of the European Geometric Average basket call and put option with 100 independent assets $S_0^i = 100$, $i = 1, ..., 100$ and parameters $K = 100$, $r = 0$, $\sigma = 0.2$, $T = 1$.

XORWOW

GA Call (95% CI)	Reduced Call	Absolute Error	Relative Error
0.168257 (0.001039)	0.16777	4.8×10^{-4}	2.89×10^{-3}
GA Put (95% CI)	Reduced Put	Absolute Error	Relative Eror
2.125461 (0.003309)	2.12855	3.08×10^{-3}	1.45×10^{-3}

MRG32K3A

GA Call (95% CI)	Reduced Call	Absolute Error	Relative Error
0.168203 (0.001039)	0.16777	4.3×10^{-4}	2.57×10^{-3}
GA Put (95% CI)	Reduced Put	Absolute Error	Relative Eror
2.129877 (0.003309)	2.12855	1.32×10^{-3}	6.2×10^{-4}

MTGP32

GA Call (95% CI)	Reduced Call	Absolute Error	Relative Error
0.16796 (0.001037)	0.16777	1.9×10^{-4}	1.13×10^{-3}
GA Put (95% CI)	Reduced Put	Absolute Error	Relative Eror
2.127392 (0.003308)	2.12855	1.15×10^{-3}	5.4×10^{-4}

Basically they have similar accuracy, and it seems *MTGP32* is better. So we will choose *MTGP32* for the following experiment.

6.2 Geometric Average

The second step is to do the geometric average. In other word, we need to do the calculation introduced in chapter 3 as

$$S_0^i \to \begin{pmatrix} S_T^{i,1}(1) & S_T^{i,1}(2) & \dots & S_T^{i,1}(NMC) \\ S_T^{i,2}(1) & S_T^{i,2}(2) & \dots & S_T^{i,2}(NMC) \\ \dots & \dots & \dots & \dots \\ S_T^{i,N}(1) & S_T^{i,N}(2) & \dots & S_T^{i,N}(NMC) \end{pmatrix} = A^i \ \ \forall i = 1, ..., d.$$

and

$$S_{ij}^{GA} = \prod_{k=1}^{d} (A_{ij}^k)^{1/d} \ \ \forall i = 1, ..., N; \ j = 1, ..., NMC.$$

The sequential algorithm to do it is calculating one by one as the following figure:

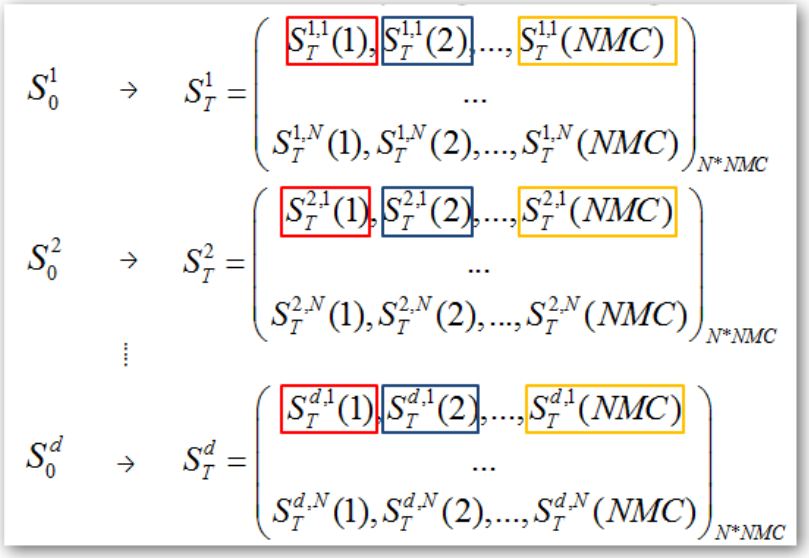


Figure 6.1: A basic sequential computation for geometric average step. We do geometric average on red blocks and it will become S_{11}^{GA}, and we do geometric average on blue blocks and it will becomes S_{12}^{GA}, and so on.

Now the idea to do GPU computing here is to assign threads to do the geometric average on each row simultaneously since all samples are independent. The idea is as following:

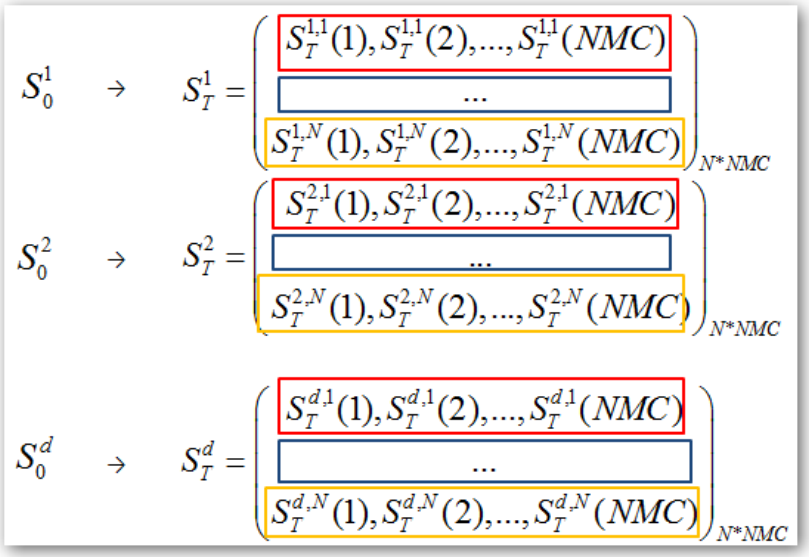


Figure 6.2: The parallel idea of computation for geometric average step. We do geometric average on each row.

An example kernel code is as following:

```
__global__ void Geo(int N_row, float *devDataMC, float *S_t, float *PayoffCall){
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    float S_T;
    float da_pro, dev_da_geo;
    while (tid < NMC){
        da_pro = 1.0;
        for (int N_stock = 0 ; N_stock < d ; N_stock++)
        {
            S_T = S_t[N_row + N_stock*N] * exp((r - 0.5f * sigma *sigma )*(T-t)
            + sigma*sqrt(T-t)*devDataMC[tid + N_stock*NMC]);
            da_pro = da_pro * S_T;
        }
        dev_da_geo = powf(da_pro,(g));          //  Geometric   Average
        //  Payoff function of Call
        dev_da_geo = dev_da_geo - K;
        if (dev_da_geo < 0.0f)
        {
            dev_da_geo = 0.0f;
        }
        PayoffCall[tid] = exp(-r*(T-t))*dev_da_geo;
        tid += blockDim.x * gridDim.x;
    }
}
```

Figure 6.3: The kernel code to do geometric average

6.3 Monte Carlo Reduction

After getting the matrix S_{ij}^{GA}, we will put it into the payoff function as $\Phi(S_{ij}^{GA}, T)$ and then do Monte Carlo method to get our $V(S_{\Delta t}, \Delta t)$ to calculate the loss function as follows:

$$S^{GA} \rightarrow \begin{pmatrix} V_{\Delta t}^1 \\ V_{\Delta t}^2 \\ \cdots \\ V_{\Delta t}^N \end{pmatrix} \rightarrow \begin{pmatrix} L_1 \\ L_2 \\ \cdots \\ L_N \end{pmatrix}$$

Then the third step is mainly about doing Monte Carlo method. The idea of Monte Carlo method is using the sample mean to approximate the expectation. So there are many ways to do reduction of all the samples to achieve the mean in parallel. A most common way is using binary operation to reduce all inputs into a single number. We can find the idea as the picture:

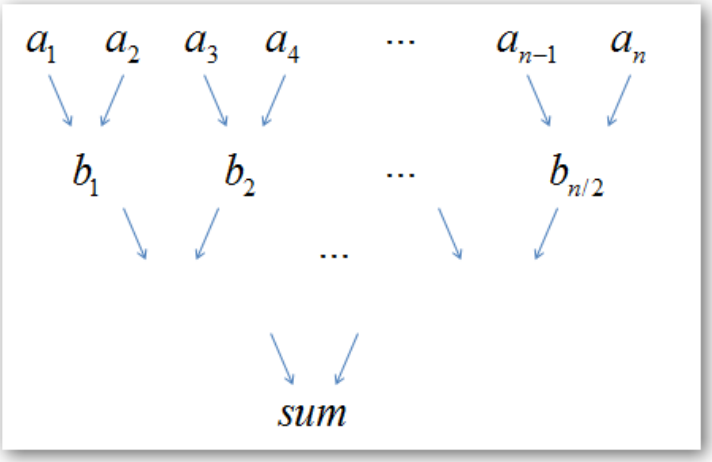


Figure 6.4: The idea of binary reduction. Each summation on the same level can be done independently.

Since this method is common so there exist many libraries for it. A useful library called *Thrust* in CUDA provides the binary reduction for a vector (see [27]). Then once we have all our samples, we can put it into an vector then just called *Thrust* to do reduction parallel. An example to use vector sum in *Thrust* is as follows

```
thrust::device_ptr<float> dev_ptr(PayoffCall);
float sum = thrust::reduce(dev_ptr,dev_ptr + NMC, (float) 0, thrust::plus<float>());
```

6.4 Sorting

Finally, we will have N samples $Loss = \{L_1, ..., L_N\}$ to do sorting and pick up the one $L_{N\times\alpha}$ to be VaR. Because sorting is also a fundamental problem for data structure, so there exist also many methods and libraries to use. We again choose library *Thrust* to do the sorting on a vector (see [27]). Here is an example to use Thrust to do sorting:

```
thrust::sort(Loss,Loss + N);
```

6.5 Numerical Result

In this numerical experiment, we take European basket call option with five assets for example. Consider $S = (S^1, S^2, S^3, S^4, S^5)$ are independent with $S_0^i = 100$, $i = 1, ..., 5$ and parameters $K = 100$, $r = 0.04$, $\sigma_i = 0.25$, $T = 1$, $\Delta t = 0.0833$.

First we compare the result with the VaR calculated by its reduced equation which we can directly use Black-Scholes formula to calculate the price of option.

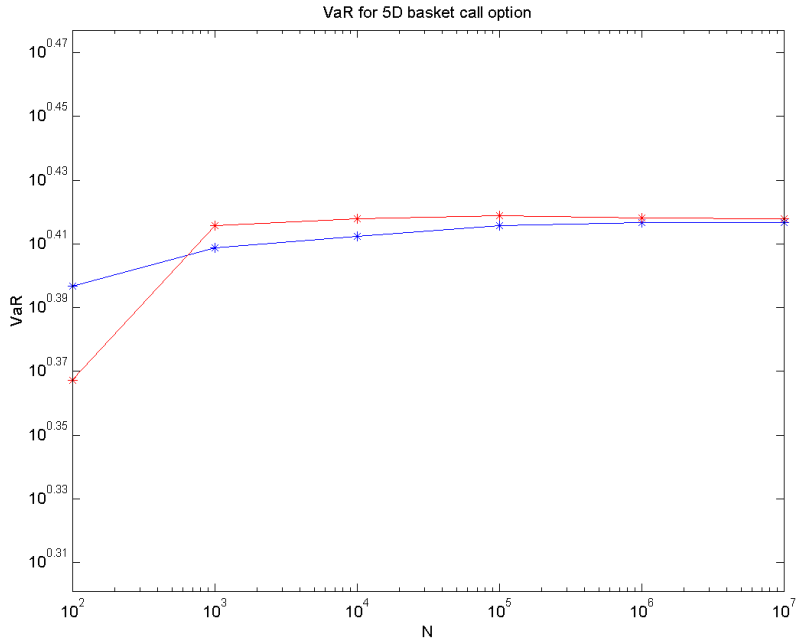


Figure 6.5: Red line is VaR for basket option, and blue line VaR for its reduced equation.

And we compare the computational time by CPUs (Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz) and GPUs (GeForce GTX 680):

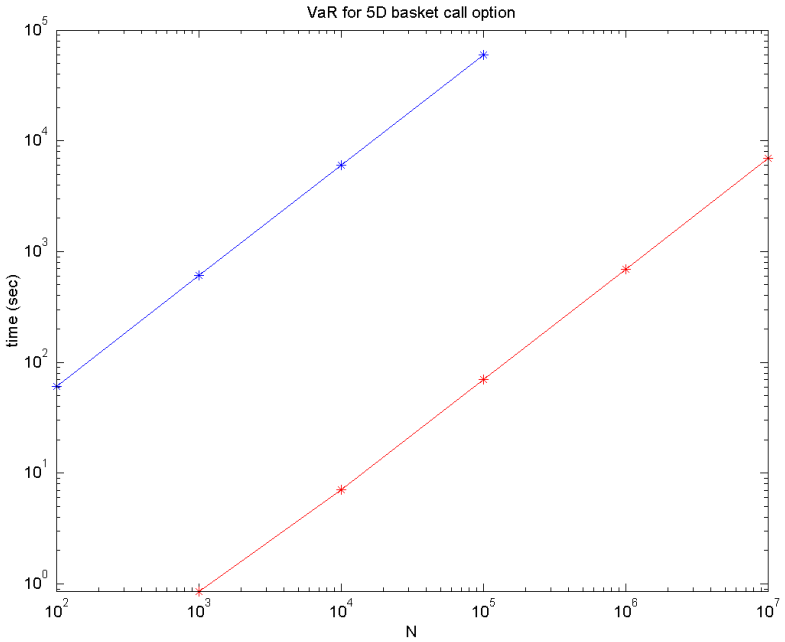


Figure 6.6: Red line is the time for computing VaR by GPUs, and blue line is the time for computing by CPUs.

We get large speed-up on this example. One main reason is that in the algorithm of using GPU computing, we do every computation in parallel. But for the code using CPUs, we didn't calculate it in different cores but only use one. Theoretical result tells us if we have optimal codes on both using CPUs and GPUs, then using GPUs will have 4 or 5 times better on performance than CPUs.

Another numerical experiment is now we take same European basket call option with five assets but with correlation matrix $\rho_{ij} = 1$ if $i = j$ and $\rho_{ij} = 0.5$ if $i \neq j$. Consider same parameters $S = (S^1, S^2, S^3, S^4, S^5)$ are independent with $S_0^i = 100$, $i = 1, ..., 5$ and $K = 100$, $r = 0.04$, $\sigma_i = 0.25$, $T = 1$, $\Delta t = 0.0833$. Again we compare the VaR with its reduced equation and also the time for whole computation between GPUs and CPUs:

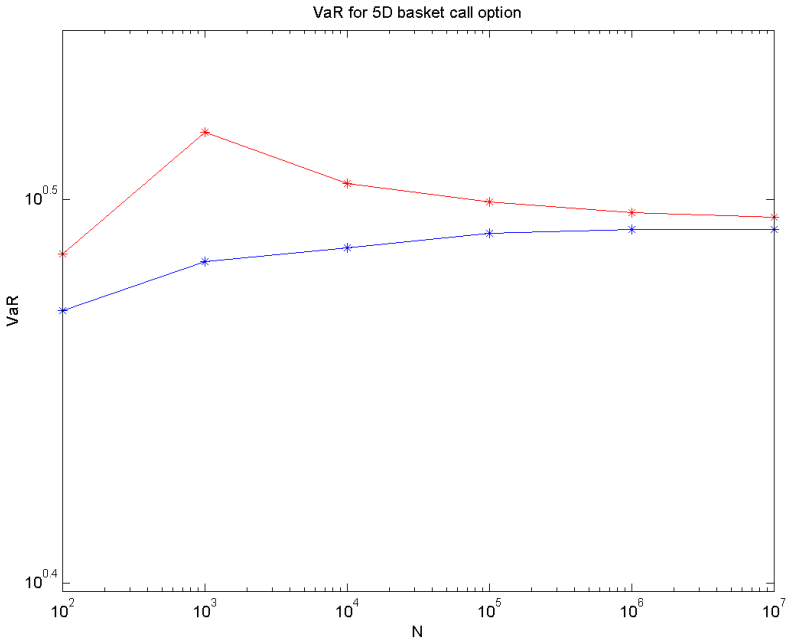


Figure 6.7: Red line is VaR for basket option, and blue line VaR for its reduced equation.

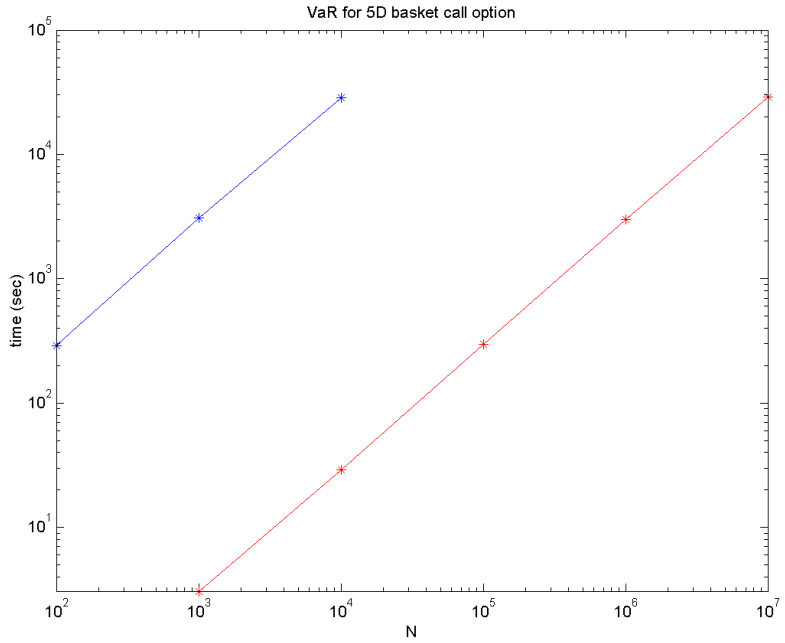


Figure 6.8: Red line is the time for computing VaR by GPUs, and blue line is the time for computing by CPUs.

We can find we still get large speed-up on the case with correlation matrix. Then it means if now we want to calculate VaR for general European option book, using GPUs will achieve better performance than CPUs.

There are still some improvements we can do on this algorithm. First we can think about how to use local memory which is faster than global memory to our problem. In such case the memory allocation problem will be considered seriously. Second, instead of doing the Geometric Average step row by row, we can even try to implement it by doing it with matrix by matrix. We can test which size of matrix we use will get the best performance on the Geometric Average step, and we can choose this size to do our experiment.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

During the previous chapters, we have recalled the fundamental theory of option pricing, and the way to compute VaR numerically. However, we bump into a problem that for complicated options or portfolio, the whole computational time is large and which can't be accepted in research and industry.

A first way we study is using importance sampling and splitting method to do some improvements. Importance sampling is an useful method to do variance reduction for calculating the integration, which we can get good accuracy without simulating too much points. It reduces the time to compute VaR since we usually need lots of samples on it. Importance splitting method is another approach to calculate the probability more adaptive, especially for rare event. We get positive effects by using these methods on assisting the computation of VaR together with the search algorithm.

Instead of digging out method from mathematics, we also try to find useful tool from computer science. It is GPU computing which gives us a large acceleration of the computation. In order to do computing on GPUs, we study some basic strategies we can have in our algorithm to do parallel computing, and try to find some existing libraries to use directly.

Both approaches from mathematics and computer science help to solve the problem on VaR computation and are easy to implement. However, there are still some methods which are worthwhile to discuss and haven't been finished during the internship. We show them here as future works.

7.2 Future work

7.2.1 Doing Importance Sampling Method with GPUs

We didn't discuss in chapter 4 for the topic that whether we can do GPU computing for importance sampling method. Actually there are some possibilities. First is when we use Delta-Gamma-Theta approach, we will need to calculate the Greeks and it can be done in parallel. Also there exist some matrix multiplications in the algorithm and it is a typical topic to think how to use GPUs on it. Finally we will need to solve an optimization problem for θ and it is interesting to dig out the possibility to do parallel computing in this part.

7.2.2 VaR for American option

Since Michaël Benguigui in OASIS team of INRIA is now doing American option pricing with grid and GPU computing using Picazo's algorithm. Then it is important to think how to combine the framework and idea of his job with VaR computation for calculating VaR for option book.

7.2.3 Interpolation Method

A bottleneck for applying Picazo's algorithm of American option pricing to VaR computation is time-consuming. For example, we compare Picazo's algorithm, Longstaff's algorithm, and binomial tree method on one-dimensional American Call option pricing. We choose parameters as $S_0 = 100, r = 0.4, \sigma = 0.25$, nbDeltaT = 20, nbMC = 10^6, deltaT = 0.05, and parameters for Picazo's algorithm are numTraingingInstances = 500, numIterations = 150, nbcont = 10000. For Longstaff algorithm we choose basis function as Laguerre polynomial (dimension 3). For binomial tree method, we choose N = 2048 which is the number of leaves. Also, we know in one-dimensional case without paying dividend, the price of European call option is equal to American call option. The result is as following:

	Call Price	Time (sec)
European B-S formula	33.4410	$<$ 1s
Picazo's algorithm	33.4592 (CI: 0.034)	CS: 47.4448s, Pricing: 0.9893s
Longstaff's algorithm	33.4269	6.546s
Binomial Tree algorithm	33.4416	13.2061s

here CS means classification. For Picazo's algorithm we use the program from Michaël Benguigui with 2 GPUs, and Longstaff's algorithm we use the code from Premia (see [28]), and we implement Binomial Tree method on MATLAB 2012.

We can image if now we choose Picazo's algorithm to simulate the loss function of American option, then even if we only need 10^4 points, we will spend 138 hours for calculating only one VaR. Clearly it doesn't work practically. So we are looking for a method that we don't need to simulate too much samples in the beginning.

An idea comes to mind is as following. If now we are looking for the 95% VaR for V, then first we generate n samples for the loss function and sort it to be $L_1, ..., L_n$. We pick up points from L_{n*a} to L_{n*b} and use these points to do the interpolation of the loss function in a small region where a and b should not be too far from 95%. Once we have the interpolation, we can use it to calculating the required samples instead of generating it from the original formula. We give an example with piecewise cubic Hermite interpolation as following:

Example 7.2.1. *Calculate the VaR for one-dimensional call option with parameter* $S_0 = 100, K = 100, r = 0.04, \sigma = 0.25, \Delta t = 0.0833, T = 1$ *and for* $\frac{S_0}{K} = 0.9, 1.0, 1.1$

We pick up $a = 88\%$ and $b = 96\%$.

method	0.9	1.0	1.1
Analytic solution	3.9225	6.2752	8.7233
Naive method (10^6 samples)	3.9194	6.2700	8.7152
Naive method (10^4 samples)	3.8941	6.2244	8.6459
Naive Interpolation method (10^4 samples)	3.9239	6.2779	8.7274

clearly it helps on VaR computation. But in fact this is not a general result, and we still need to investigate on how to do it in more complicated case.

7.2.4 Stochastic Approximation Method

O. Bardou in [29] developed a method by using stochastic approximation to calculate VaR. The main idea is to write VaR as an expectation, and then apply optimization method like stochastic gradient descent to approximate the expectation. An interesting thing is that they had same problem of time-consuming, and they also chose importance sampling method for help. However, there is no comparison or running time in [29], so it is interesting to compare their methods with methods introduced in this thesis.

7.2.5 Quasi Monte Carlo Method

Using Quasi Monte Carlo method to accelerate calculating expectation is common and many people usually use it. Since in our problem, we also need to evaluate many expectations, so naturally we wonder to know whether Quasi Monte Carlo method helps to do VaR computation. Actually in [30], there are results showing that Quasi Monte Carlo method is better on evaluating the price of basket option than Monte Carlo method. But they only discussed European case. In American option pricing, we will need to do some dynamical computations which are not just calculating expectation. Therefore it is worthwhile to test whether Quasi Monte Carlo method also works on such case and see how to apply it to calculating VaR for option book.

Bibliography

[1] F. Black, M. Scholes. The Pricing of Options and Corporate Liabilities. Journal of Political Economy, pages 637-654, 1973.

[2] H. Gifford. Fong, Kai-Ching. Lin. A New Analytical Approach to Value at Risk. The Journal of Portfolio Management, Vol. 25, No. 5: pages 88-97, 1999.

[3] P. Glasserman, Monte Carlo Methods in Financial Engineering. Springer, 2003.

[4] P. Glasserman, P. Heidelberger, P. Shahabuddin. Variance reduction techniques for estimating value-at-risk. Management Science, 46(10): pages 1349-1364, 2000.

[5] P. Glasserman, P. Heidelberger, P. Shahabuddin. Portfolio value-at-risk with heavy-tailed risk factors. Mathematical Finance, 12(3): pages 239-269, 2002.

[6] J. Morio, R. Pastel, F. Le Gland. An overview of importance splitting for rare event simulation. European Journal of Physics, pages 1295-1303, 2010.

[7] F. Cérou, P. Del Moral, T. Furon, A. Guyader. Sequential Monte Carlo for rare event estimation. Statistics and Computing, Volume 22, Issue 3, pages 795-808, 2012.

[8] John C. Cox, Mark Rubinstein. Options Markets. Prentice Hall, 1985.

[9] Steven E. Shreve. Stochastic Calculus for Finance II: Continuous-Time Models. Springer, 2004.

[10] Damien Lamberton, Bernard Lapeyre. Introduction to Stochastic Calculus Applied to Finance. Chapman and Hall/CRC, 2007.

[11] Yves Achdou, Olivier Pironneau. Computational Methods for Option Pricing. Society for Industrial and Applied Mathematic, 2005.

[12] R.C. Merton. Theory of rational option pricing. The Bell Journal of Economics and Management Science, pages 141V183, 1973.

[13] F.A. Longstaff and E.S. Schwartz. Valuing American options by simulation: a simple least-squares approach. Review of Financial Studies, 2001.

[14] J.A. Picazo. American Option Pricing: A Classification-Monte Carlo (CMC) Approach. Monte Carlo and Quasi-Monte Carlo Methods 2000: Proceedings of a Conference Held at Hong Kong Baptist University, Hong Kong SAR, China, pages 422-433, 2002.

[15] Viet Dung Doan. Grid Computing for Monte Carlo Method based intensive calculations in financial derivative pricing applications. PhD thesis. University of Nice Sophia-Antipolis, 2010.

[16] Michaël Benguigui, Françoise Baude. Towards parallel and distributed computing on GPU for American basket option pricing. Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference.

[17] Duffie, Darrel, Jun Pan. An Overview of Value at Risk. Journal of Derivatives, 4, No.3, 1997.

[18] Irina N. K, Svetlozar T. R. Value at Risk: Recent Advances. Chapman and Hall / CRC, pages 801-858, 2000.

[19] Robert F. Engel. Autoregressive Conditional Heteroskedasticity with Estimates of United Kingdom Inflation. Econometrica, Vol. 50, No.4, pages 987-1007, 1982.

[20] Eric Fournié, John-Michel Lasry, Jérôme Lebuchoux, Pierre-Louis Lions, Nizar Touzi. Applications of Malliavin calculus to Monte Carlo methods in finance, Finance and Stochastics, pages 391-412, 1999.

[21] CUDA C Programming Guide, 2013.

[22] Jason Sanders, Edward Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 2010.

[23] Shane Cook. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. Morgan Kaufmann, 2012.

[24] P. L'Ecuyer, R. Simard. TestU01: A C Library for Empirical Testing of Random Number Generators ACM Transactions on Mathematical Software, Vol. 33, article 22, 2007.

[25] http://www.iro.umontreal.ca/ simardr/testu01/tu01.html

[26] CURAND Library Programming Guilde, 2013.

[27] http://thrust.github.io/

[28] https://www.rocq.inria.fr/mathfi/Premia/

[29] O. Bardou, N. Frikha, G. Pagès. Computing VaR and CVaR using Stochastic Approximation and Adaptive Unconstrained Importance Sampling. Monte Carlo Methods and Applications, pages 173-210, 2009.

[30] Corwin Joy, Phelim P. Boyle, Ken Seng Tan. Quasi-Monte Carlo Methods in Numerical Finance. Management Science, pages 926-938, 1996.

[31] L. Tierney. Markov Chains for exploring Posterior Distributions. Annals of Statistics 22, 1701V1786, 1994.