KTH Numerical Analysis
and Computer Science

A Monte Carlo Solver for Financial Problems

STEFAN THORÉN

Graduation thesis at NADA
Supervisor: Mikael Goldmann
Examiner: Stefan Arnborg

TRITA xxx yyyy-nn

Abstract

This thesis proposes a way to design software for Monte Carlo simulation that facilitates the simulation of many different kinds of stochastic processes. Monte Carlo simulation is a powerful tool that has applications in many financial contexts. One important application is the pricing of complex financial derivatives.

Pricing derivatives is a recurring problem for many financial institutions. Many different kinds of derivatives exist on the financial markets, and new kinds are introduced continually. A software for Monte Carlo simulation that is adaptable to price different derivatives could potentially save money, time and effort.

The thesis provides an introduction to Monte Carlo simulation in the financial markets. An analysis of the problem considered in the thesis project is given and a design of a Monte Carlo simulation engine is given. Finally, examples illustrating the use of the software are given.

Sammanfattning

Detta examensarbete föreslår en mjukvarudesign för Monte Carlo- simuleringar. Designen möjliggör att olika sorters stokastiska processer kan simuleras. Monte Carlo- simuleringar är ett kraftfullt verktyg med många tillämpningar i finansiella sammanhang. En viktig tillämpning är värdering av finansiella derivatinstrument.

Värdering av derivatinstrument är ett återkommande problem för finansinriktade företag. Det finns många olika typer av derivatinstrument redan idag, och antalet ökar kontinuerligt i takt med att nya finansrelaterade behov leder till nya lösningar. En mjukvara för Monte Carlo- simuleringar som lätt kan anpassas för att simulera olika typer av derivat skulle kunna spara mycket tid och pengar.

Denna uppsats ger en introduktion till hur Monte Carlo- simuleringar utförs för finansiella derivat. En analys av det problem som ligger till grund för arbetet ges och designen av den resulterande mjukvaran redogörs för. Avslutningsvis ges exempel på hur mjukvaran kan användas.

Contents

Chapter 1

Problem

1.1 Background

Monte Carlo simulation of is a method for valuing derivativs that has achieved much interest during the last decades. It is a flexible method that has proved suitable to value complex financial instruments. The method is used by financial institutions throughout the world to price mortgages, pension plans, building contracts and financial derivatives, to name a few applications.

Algorithmica Research AB is a company that develops software applications for the financial markets. The company has focused on solutions for trading and trading support. One of the products the company offers is Quantlab, a software environment that facilitates simulation and visualisation of trading scenarios. Users of the system work in workspaces. The workspaces provides visualisation of outputs from a set of simulations.

Quantlab comes in two editions, corresponding to a segmentation of the users into two groups. The first edition is aimed at *developers*. It provides an extensive functionality, and is usually used to create work spaces for traders and sales staff at financial institutions. The second edition is aimed at a group denoted *users*. Users are able to analyse the scenarios in work spaces and change input parameters. Users are typically traders and sales staff.

It is usually the case that a workspace developed by a developer is used by users in the same organisation, even if exceptions exist[1]. To keep these two user- groups apart in this text, I will in this report refer to the first group as developers, and to the second group as users, unless the intended group is obvious from the context of the text.

[1] Algorithmica Research AB has occasionally developed workspaces for clients who did not wish or could do the work themselves.

QuantLab also provides an expression language, QuantLang, which may be used to develop workspaces. For the advanced user there is a C++ API available, that is intended to provide the developer with an easily extendable programming environment. A developer may extend the QuantLab functionality by creating a dynamic link library (dll) complying with the API. QuantLab offers a developer to specify additional dlls to include.

Quantlab is a product is under constant development. One desired feature in a future version of the product would be an engine for Monte Carlo simulation of derivatives.

1.2 Purpose

The purpose of the project is to design and implement prototype software for Monte Carlo simulations of financial derivatives, adaptable to Algorithmica Research's QuantLab product.

1.3 Solution requirements

The solution was to be developed within the boundaries set by the nature of the problem and the needs of Algorithmica Research's clients.

The first requirement was that the engine was to evaluate a vast variety of different derivatives. In general, it is very hard to predict what kinds of derivatives Algorithmica's clients wish to value. It is therefore neccessary that the software allows the client to specify the conditions for the valuation herself. The conditions are introduced below.

The models used to simulate paths of the underlying are numerous. Besides the commonly available models, it is common that Algorithmica's clients develop models of their own. It not always the case that the models used by clients is publicly available. The financial markets are highly competitive and clients are protective of information of the models they use to evaluate derivatives. To avoid disclosing the models used, and to provide flexibility to use any model, the software would ideally allow the user to define the model herself.

Given a path of the underlying, the value of a derivative is defined by a pay-off rule on that path. The pay-off rule can take any form, spreading from simple rules for European call and put options to much more complex rules, such as knock-in/-out rules and different kinds of averages. Given the difficulty to predict what derivatives the client wish to model, now and in the future, pay-off rules are also to be client- defined.

The importance of variance reduction techniques to achieve result quickly is discussed in section 2.4.2. For the Monte Carlo engine to be useful, it is neccessary that the engine facilitates the use of such techniques. The software will therefore provide the user with options to apply different methods of variance reduction.

Chapter 2

Theory

This chapter contains the theory that underlies Monte Carlo simulations of financial derivatives. The chapter starts with an introduction to financial derivatives. A definition of a derivative is given, and examples of popular kinds of derivatives are given. Thereafter, models of the behaviour of underlying assets are introduced. Such models are used in Monte Carlo simulations. A few models that are interesting from a theoretical and practical point of view are presented as examples.

Last in this chapter is a description of the Monte Carlo method. The description is adjusted to describe simulation of financial derivatives. An outline of a simulation is given together with a discussion on the theoretical consequences of the method.

2.1 Derivatives

A derivative is a contract between two parties that defines the rights and obligations of the parties. The rights defined in the contract may have a value to one or both parties. That value is dependent on conditions of other assets or events. In this section an overview of what a derivative is and examples of common derivatives are given.

There exist a number of definitions of what a derivative is. In [5], a derivative is defined as:

```
An instrument whose price depends on, or is derived from,
the price of another asset.
```

Alternative definitions exist. It seems that most definitions are unanimous in that they consider a derivative to be a notation that cover a big class of financial instruments whose value depend on the value of something else. That something can be almost anything, and it is usually referred to as the "underlying asset" of the derivative. Common examples are when the instrument price depends on the price

of a stock or bond, but there has also been more exotic constructions when the price of the instrument depends on weather or the occurrence of catastrophic events such as hurricanes and earthquakes. The underlying asset need not be something that we generally regard as an asset in itself, as long as it is something that can be measured.

A derivative is a contract between two parties that defines rights and obligations of the parties. The value of the contract depends on the terms spelled out in it. Depending on the underlying asset and the terms of the contract a derivative may take on many forms. Some of the most widely spread "standard" derivatives today include:

- Options, an agreement where one party pays the other for the right to something in the future. Stock options, where one party pays for the right to buy a stock at a future date at a pre- specified price, are common.
- Swaps, an agreement between two parties to exchange something at one or more future dates. Interest swaps, where two parties swap fixed interest payments for floating interest payments on a specified amount, are common.
- Forwards, an agreement where one party agrees to buy an asset from the other party at a specified future date and price.

There exist numerous variations of financial derivatives that are traded at exchanges throughout the world[1]. These contracts are standardized and specify the conditions of the derivative. In addition, many derivatives are traded "over the counter" (OTC). These derivatives are not standardized, but are more flexible when it comes to the terms. The contracts are not registered and traded on a specific market place.

Many common contracts in varying fields of the economy that are usually not thought of as derivatives may be interpreted as such. One example is the building company that obtains the right to build houses on an acquired piece of land. The value of that right will depend on (among other things), the price of houses when the building work is done, the labour costs during the building period and the interest rates that apply during the period until all houses are built and sold. These sort of implicit derivatives are usually referred to as "Real options".

[1]Examples of such exchanges are the Chicago Board Options Exchange, CBOE [2], and Stockholmsbörsen [8]

The different contracts that may be called derivatives are too many to list. Derivatives have been written where the underlying is not a financial asset, but the magnitude of an event. Weather options may be defined so that the payoff depends on the amount of rain that has been measured on a weather station during a period of time. New derivatives are continually introduced to accommodate the needs of companies and organisations.

2.2 Value of a derivative

In section 2.1, a derivative was describes as a financial instrument that has a value that depends on the value of another asset. Since a derivative at the same time is a contract between two parties; its value is a measure of how beneficial the rights of the contract are to one of the parties. To the other party, the value of the derivative is a measure of how high the costs of fulfilling the contract terms will become. Both parties would therefore like to know the value of the contract before entering it and during its lifetime.

How do you calculate the value of a derivative? The contract terms define how the value of the derivative is determined from the underlying asset. The value is calculated from the path of values that the underlying asset has traversed during the derivative's lifetime. One way to calculate the value of the derivative is therefore to apply the rules that define the payoffs of the derivative to the path taken by the underlying. A few examples on how this may be done are given in the following subsections.

2.2.1 Call and put options

The call option is an option that has a simple rule for the value of the derivative. One usually makes a distinction between two types of call options. The *European* call option has a value that is calculated from the value of the underlying at the last day of the options lifetime. At that time, T, the value is determined to be

$$max(S_T - K, 0),$$

where S_T is the value of the underlying and K is a pre-determined *strike price*. The value of the option can never be negative.

The *American* call option can be called at any time during the option's life, in contrast to the european.

Put options are defined in a similar manner as call options. The payoff for a put option is

$$max(K - S_T, 0),$$

using the same notations as above. If S_T is the value at the last day of the option's lifetime, the option is a *European* put option. If the option can be called at any time, the option is an *American* put option.

2.2.2 Asian options

The Asian option has a payoff that depends on the price of the underlying asset during a period of time. The average price of the underlying during part of[2] the option's lifetime, S_{asian} is compared to a strike price. The value of an Asian call option is

$$max(S_{asian} - K, 0),$$

where K is the strike price of the option.

The Asian option is an example of a $path - dependent$ option. This class of options use part of, or the whole, path of the underlying during the option's lifetime to calculate a payoff. The path- dependant options are often contrasted with options that has a value that is derived from the final value of the underlying.

2.2.3 Prediction vs. After-math

The terms in a derivative contract spell out how the value of the derivative is calculated from the path of values the underlying follows during the lifetime. By using the rules, we can always calculate the value of the derivative after its lifetime has expired.

It is not sufficient to only be able to value the derivative after its lifetime has expired. The contract has a value when it is entered that corresponds to the cash flows that it is expected to cause. There is a big difference in predicting those cash flows and calculating the value of the derivative afterwards.

[2]Or all of

Any derivative has a value that can be calculated from the cash flows that it gives rise to. Using the risk- neutral valuation method proposed by Cox and Ross [3] , the value of a derivative that causes cash flows $C_1, C_2..., C_n$ at times $t_1, t_2, ..., t_n$ is

$\sum_{i=1}^{n} e^{-r_i t_i} C_i,$

where r_i is the risk- free interest rate from now to t_i.

The cash flows of a derivative may occur at several times during and after the derivative's life. The cash flows are determined by the terms in the contract and dependent on the path the underlying traverse. If we were to estimate the value of the derivative before its lifetime has expired, one valid approach would be to try to predict the path of the underlying and apply the derivative's payoff rules to that path. The cash flows resulting from the predicted path are then discounted, using appropriate interest rates, to achieve the derivatives risk- neutral expectation.

2.3 Asset Models

In section 2.1 a derivative was described as an asset with a value that could be derived from the value[3] of another asset. If we were able to predict the value of the underlying asset at all future times, we would thus be able to determine the value our derivative exactly. Unfortunately, very few assets allows us to predict their value in advance. Still, we would like to determine a fair current value of the derivative. How could we determine such a value without knowledge of the future?

One approach is to use past behaviour of the asset and assume that the asset will behave in a similar manner in the future. We are often able to go back in time and examine past behaviour[4], but what we are really interested in is to be able to say something about the asset behaviour in the future. The value of the derivative at present date is after all decided by the path we expect the underlying to traverse in the future. Many attempts have been made to develop models of the behaviour of different kinds of underlying assets. A model of the behaviour of the underlying asset is, as we will find out in section 2.4, the starting point of the Monte Carlo simulation.

Even though a derivative can be specified with almost anything as its underlying asset[5], two types of underlying assets are particularly popular on the financial markets today. Options on stocks and bonds are of great importance. Together, they constitute a big share of the option market. As a result, extensive theory on

[3]Or path of values

[4]Behaviour of prices, volatility etc

[5]For examples, see section 2.1

how to model stocks and bonds has developed. In general, the models include a random element to explain the unpredictable behaviour of asset prices.

Models of stocks and bonds often describes the price of the asset as a stochastic differential equation (SDE). With this approach, the modeller tries to describe the seemingly random movement of the asset price during a short[6] period of time. As time progresses, the asset prices will construct a path described by the SDE.

The stochastic model of an underlying asset attempts to capture the behaviour of an asset. It is of course very important that the model does this correctly, without too much error. If a model is not accurate enough, the conclusions that can be drawn from Monte Carlo simulation will be subjected to errors, since they are based on false premises. Much research has been done on how to model different types of assets. It is fair to emphasize that the models leave parameters for the user to define, and that the choice of model and parameters are highly subjective. There might be exist competing models for the same underlying.

The amount of research done is also in part a consequence of the fact that the underlying could be almost anything. It is natural to assume that different types of underlying assets exhibit different behaviour. It is for example plausible that stock and bond prices should be modelled in different ways, reflecting their different behaviour.

When a model has been defined by a stochastic differential equation, it is time to contemplate the implications of the model. We know from stochastic calculus that a function of a stochastic variable is itself stochastic[7]. Thus, the value of the derivative itself is a SDE. The following sections introduce commonly used models for stocks and bonds, which will be at the centre of this text when the Monte Carlo method is introduced.

2.3.1 Stock Models

Many models of stock behaviour can be seen as extensions of the work by Fischer Black and Myron Scholes in the 1970's. Black and Scholes are perhaps most famous for the formulas they developed for derivative pricing. It is important to remember that the formulas are consequences of an assumed model of the economy, and changes to that model affect the pricing.

Black- Scholes Model

The natural starting point when describing models of stock price behaviour is the model proposed by Fischer Black and Myron Schols in 1973[1].

[6] In a continuous model infinitesmal

[7] Ito's lemma

The Black- Scholes model assumes that there exists at least two securities, a stock and a money market account. The interest rate on the money market account is held at a fixed level and the stock price S is assumed to follow a geometric Brownian motion governed by the stochastic differential equation

$$dS = \mu S dt + \sigma S dz, z \in N(0,1)$$

The stochastic differential equation relates the change in asset price to the current asset price, a drift μ, a variance σ and the outcome of a random variable drawn from the Gaussian distribution.

The drift μ in the equation can be interpreted as the long term trend of the stock price. As $\rightarrow$ inf, μ will be the dominant factor in determining the stock price. The second term, $\sigma S dz$, may also be given an interpretation. It is the short- term departure from the long- term trend.

The Black- Scholes model makes the following assumptions on the economy:

- The stock follows a geometric Brownian motion with constant volatility.
- The stock does not generate any dividends or other cash flows[8].
- The interest rate on the money market account is known and constant.
- The option can only be exercised on the expiration date, i. e. it is European.
- Markets are efficient
- No commissions are charged.

Black and Scholes used the assumptions above to derive their famous valuation formulas for European call and put options[9].

[8]Black and Scholes were interested in valuating options. When valuation of an option is performed, this assumption can be relaxed. The requirement is that no dividend or cash flow occurs during the lifetime of the option.

[9]Please see [5] or any other introductory text on derivative valuation

2.3.2 Interest Rate Models

Even though some derivatives written on interest rate products can be analysed within the Black- Scholes framework with lognormal returns, the framework has limitations[5]. A model based on Black- Scholes does not describe how interest rates themselves change over time. Black- Scholes describe the movement of a single variable, while interest rates are described by an entire yield curve. The yield curve change over time, as a consequence that the rates that form it change.

In the following subsections, common models of interest rates are introduced. Interest rate models are often more complicated than the Black- Scholes world, even when the extensions of the model included.

The Vasicek Model

The Vasicek model is a one- factor model of the short interest rate. That the model is one-factor means that it only accounts for one source of uncertainty. That it models the short rate is another way of saying that the rate modelled is the rate that prevails for an infinitesimally short period of time. A rate defined this way is sometimes referred to as the instantaneous short rate.

The Vasicek model describes the short interest rate r as a risk- neutral process:

$$dr = a(b - r)dt + \sigma dz$$

where a, b and σ are constants.

The terms in the model can be interpreted in almost the same way as the terms in the Black- Scholes model2.3.1, with a few amendments. The first term, $a(b-r)dt$ is the dominant term as time increase. b is a level that the long- term trend of r converges to. This is easily understood if you consider what happen as time increases. If the interest rate r is above the long- time level b, $(b - r)$ will be negative. This implies that the interest is more likely to decrease in the next short time period. The argument is reversed for the case when r is smaller than b. a is a measure of how fast the convergence is. A small a means slower convergence. σdz

One- factor models such as Vasicek's can generally be written on the form

$$dr = m(r)dt + s(r)dz,$$

where the instantaneous drift m and instantaneous standard deviation s are functions of the prevailing interest rate r. Depending on the way m and s are defined, other one- factor models arise. Examples of comminly used models are Rendleman and Barter's, Cox's and Ingersoll and Ross'. The interested reader

finds an account of different one- factor models in [5]. I believe that Vasicek is a good example because of its simplicity. It is easily understood, and at the same time it encompasses the concept of mean reversion, a feature that often occurs in financial models of assets.

The interest rate models presented so far are examples of equilibrium models. Such models usually make assumptions on economic variables to derive a process for the risk-free instantaneous rate. As is emphasized in [5], the models attempts to model a process in a risk- neutral world, which is not necessarily the same as the real world.

One consequence is that the equilibrium models may not fit today's term structure. There is often a discrepancy between the model and the current term structure. The discrepancy can be minimized by a careful choice of parameters, but there are often still significant errors.

In contrast to the equilibrium models, there are models designed to be consistent with the current term structure. These models are said to be no-arbitrage models. They usually incorporate a time dependency in the drift of the short rate.

The Heath, Jarrow and Morton Model

In 1992, Heath, Jarrow and Morton (HJM) presented a new approach to interest rate modelling. HJM models calibrate to the initial yield curve automatically, and thus avoid the inconsistencies of equilibrium models. The model is also interesting since it implies that the process for the short- term risk- free interest rate is non-Markov. Constructing a tree for the process is therefore complicated. Monte Carlo simulation is a better way to solve problems based on the HJM framework[5].

For a detailed account of the model, the reader is recommended to read either [5] or the original text [4].

2.4 The Monte Carlo method

This section introduces the reader to the Monte Carlo method in a financial setting. The benefits and drawbacks of the method are presented, as well as techniques intended to improve the performance of the method.

The section is organized in the following manner. First, the Monte Carlo method is examined from a financial perspective. The focus is on derivative valuation, and the insecurity of the simulation result is looked into. Second, the issue of how to

achieve an accurate result with a limited amount of computing resources is discussed. Finally, different techniques to achieve more accurate results are introduced to the reader.

2.4.1 Monte Carlo in financial markets

The Monte Carlo method was first used to value derivatives by P.P Boyle in 1977 [7]. The method is flexible in the sense that it can be applied to differently stated problems and it has the benefit of being well suited to deal with multiple random factors. This often comes in handy. One example is when the underlying asset is a basket of stocks or when volatility or interest rates are random.

The Monte Carlo method is also useful when derivatives are path- dependent. It allows for the model of the underlying asset to include jump processes, a feature that enables more realistic modelling of asset price paths.

The major drawback of Monte Carlo simulation is that it may require a significant amount of computer time to obtain a reliable result. The efficiency can often be improved by using so-called variance reduction techniques, an issue that is discussed in section 2.4.2.

As stated in section 2.1, the value of an option is the risk-neutral expectation of its discounted payoffs. We are thus interested in finding that expectation for the option at hand. It is sometimes possible to find an explicit formula for the expectation of the derivative by analytical means. One example of such a case is the value of the European call option on a stock, which can be obtained by using Black- Scholes formula. In many cases however, there is no explicit solution to the problem. In those cases it makes sense to look for other ways to value the derivative. One approach is to derive a value for the risk- neutral expectation by Monte Carlo simulation.

Monte Carlo simulation is a method that aims to obtain statistical estimates, primarily of the risk- neutral expectation of a derivative. I would like to emphasize that the term *statistics* may be ambivalent. The term notates any statistical measure that may be obtained from a Monte Carlo simulation. The risk- neutral expectation of the derivative is one example of such a measure. When the term *statistics* is used, it includes the risk- neutral expectation.

So far, this text has introduced a number of different kinds of derivatives. The derivatives exhibit many different properties, but they have in common that their value can be derived from the path of the underlying asset. Common models of the behaviour of the underlying were introduced in section 2.3. The models have in common that the behaviour of the underlying asset can be described by a stochastic

differential equation. Now the time has come to explain how to perform a Monte Carlo simulation to achieve the derivative's risk- neutral expectation.

The idea of Monte Carlo simulation is simple, yet elegant. Its foundation is the central limit theorem, which is applied to a large number of derivative values. These are obtained by first simulating the path of the underlying, using the SDE, and then applying the pricing rule for the derivative to that path. Let us first examine how this is done in more detail, and thereafter discuss the theoretical implications of the method.

A Monte Carlo simulation can be viewed as consisting of three phases. In the first phase, the conditions for the simulation are defined. In the second phase, the actual simulation is performed. In the third phase, the resulting statistics are calculated. Below is an account of the three phases, merged together to give a better overview of the steps involved in a simulation.

1. Before the simulation

- A model of the behaviour of the underlying is defined [10]
- The derivative's payoff is defined.

2. During simulation, repeat

- Simulate a path of the underlying over time according to the given model.
- Calculate the value of the option, given the simulated path of the underlying.

Continue the simulation until sufficiently many simulations have been performed. We will later discuss what is meant by "sufficiently many".

3. Compile statistics from the simulated paths

- Take the average of the estimate as an approximation of the options expected payoff.
- Discount the expected payoff with an appropriate interest rate to achieve the risk- neutral expected pay- off.

As mentioned, the foundation of the Monte Carlo method is the central limit theorem. The value of the derivative that result from each simulation can be seen as a draw from a distribution of possible outcomes of the derivative. The distribution has a mean m and variance σ. If the random number generator used is good, the draws from the distribution does not show any pair wise correlation. That is, the outcomes of the simulations of the derivative are not correlated. We apply the

[10] The model may be one of the models described in section 2.3, but not necessarily.

central limit theorem to the averaged sum of the estimates. If we have many estimates of the option value, the central limit theorem tells us that that if we take the average of the estimates, it will converge to the expected value.

For now, let us assume that we somehow have access to a Monte Carlo simulator that allows us to perform a simulation as described above. If we apply the rules of the derivative to a path taken by the underlying asset, we get n cash flows $C_1, C_2, ..., C_n$ at times $t_1, t_2, ..., t_n$[11]. We discount the cash flows, using the appropriate interest rates, and obtain the discounted cash flow of the option. Let s_i denote the discounted cash flow of the option received for the ith path. We have that

$s_i = \sum_{k=1}^{n} e^{-r_k t_k} C_k$

In our Monte Carlo simulation, we perform n simulations and average the results. Denote that average of the estimates $\hat{S}$. For the average, we have that

$\hat{S} = \frac{1}{n} \sum_{i=1}^{n} S_i$

The central limit theorem states that $\hat{S}$ will converge to the true expected value $E(s)$ as $n \to \infty$.

It is important to realize that $\hat{S}$ only is an approximation of $E(s)$ for any finite n. The central limit theorem states that the averaged mean $\hat{S}$ exhibits a standard error of size

$\frac{\sigma}{\sqrt{n}}$.

The standard error is a measure of the insecurity in the estimate of the derivative's value. From the size of it we can draw two conclusions. First, we can improve the accuracy of our simulation by performing more simulations. Second, since the error decreases as $O(\frac{1}{\sqrt{n}})$, it is possible that many simulations are needed to provide high accuracy.

Every simulation estimate is a result of a number of path constructions and evaluations of those paths. The time required to complete the simulation is very much dependant on the number of paths we simulate. Time is often a scarce resource in the context of Monte Carlo simulations. If possible, we would like to achieve a result faster, given a level of accuracy. We are interested in reducing the variance of m, while at the same time not changing m itself.

Techniques developed to improve the performance of a Monte Carlo simulation by reducing variance are referred to as "variance reduction techniques". Some of

[11] This is an extension of the argument on page 82 in Implementing derivatives models

them are general enough to be applied to almost any given problem, while other needs to be adjusted to the problem at hand. The rest of this section introduces some common variance reduction techniques and explains why a reduced variance is obtained.

2.4.2 Variance reduction techniques

This section presents three variance reduction techniques that are often used. The presentation of the variance reduction techniques only scratch the surface of the work done in this field. For the interested reader, there is plenty of litterature and studies on variance reduction. Peter Jäckel's *Monte Carlo methods in Finance* [6] is a standard text. Section 2.4.2, 2.4.2 and 2.4.2 all draw on the material presented in that book.

Antithetic variates

The random numbers used to drive the stochastic process symbolizing the path of an asset are often drawn from distributions that are symmetric. Examples of such distributions are the normal and the uniform distribution.

Assume that we draw a vector of random numbers, z_i from a symmetric distribution. If each element in z_i is mirrored in an imaginary mirror placed at the distributions mean we get $\hat{z}_i$. $\hat{z}_i$ is also a vector of random numbers drawn from the distribution. How can we utilize this observation?

The trick is to reuse the vector of numbers drawn to construct a new path. That is, for every vector of drawn numbers, two paths are constructed. One path arises from using the numbers in z_i and one from using the numbers in $\hat{z}_i$. It is often more time consuming to generate two vectors of random numbers than to generate on and mirror it. By reusing work already done, we benefit. Let us discuss the theoretical consequences of using antithetic variates.

At first, it may seem that the technique violates the foundation of Monte Carlo-simulation. Clearly, z_i and $\hat{z}_i$ are not independent! Remember that we use the central limit theorem when we state that the averaged value of the simulations converges to the true value of the options. We need to deal with this, and the way we do it is by not viewing the paths generated by z_i and $\hat{z}_i$ and individual samples. Instead, we average the results obtained by using the vectors to form pair wise averages. If we denote the pairwise averages $\hat{v_i}$, we have that

$$\hat{v_i} = \frac{v(z_i)+v(\hat{z}_i)}{2}$$

The averages are i.i.d, and the central limit theorem can be applied to them. It is clear that the expectation of $\hat{v_i}$ is m, since

$E(\hat{v_i}) = E(\frac{v(z_i)+v(\hat{z_i})}{2}) = \frac{E(v(z_i))+E(v(\hat{z_i}))}{2} = \frac{m+m}{2} = m$

Computing an average $\hat{v_i}$ requires one vector of random numbers, a mirroring and two calculations of the v function. If variance is to be reduced, it is necessary that

$V[\frac{1}{2}(v_i + \hat{v_i})] < \frac{1}{2}V[v_i]$

This is a simple way to state that since the averaged pair requires two calculations, the extra computations must be reflected in a decreased variance if the technique is to improve performance. The statement is equivalent to saying that the covariance of v_i and $\hat{v_i}$ is less than zero,

$Cov[v_i, \hat{v_i}] < 0$

Control variates

Sometimes, the problems we attempt to solve by Monte Carlo- simulations either have approximate solutions, or are similar to other problems, for which closed form solutions exist. When we apply the method of control variates, we use the approximate solution to our problem to reduce the variance in our estimate of the expected value.

Let us see how this is done. The payoff function that is used to describe the derivative payoff is denoted $v(u)$, where u is the vector of random numbers underlying the stochastic process. In addition, we have the related function $g(u)$, whose expected value is exactly known. The ordinary Monte Carlo estimator $\hat{v} = \frac{1}{n}\sum_{i=1}^{n} v(u_i)$ is an estimate of the expected value of the derivative payoff.
Since we have that

$$E[\frac{1}{n}\sum_{i=1}^{n} v(u_i)] = E[\frac{1}{n}\sum_{i=1}^{n} v(u_i)] + \beta(E[g(u)] - \frac{1}{n}\sum_{i=1}^{n} g(u_i)) \tag{2.1}$$

we can replace $\hat{v}$ by the control variate estimator $v_{\hat{CV}}$

$v_{\hat{CV}} = \frac{1}{n}\sum_{i=1}^{n}[v(u_i) + \beta(E[g] - g(U_i)))]$

But why is $v_{\hat{CV}}$ a better estimator than the original $\hat{v}$?

To begin with, the expectations of the estimates are identical according to equation 2.1. But why do we achieve a smaller variance if we use $v_{\hat{CV}}$ rather than $\hat{v}$?

The answer is that every simulation that overestimates $v(u_i)$, also is likely to overestimate $g(u_i)$. Therefore, the two functions will often partly cancel out the error in the simulation when $v_{\hat{CV}}$ is used as the estimator.

How do we find the value for β? The optimal choice [6] is to set

$\beta* = \frac{Cov[v,g]}{V[g]}$

$Cov[v, g]$ and $V[g]$ may be hard to determine exactly. We can however, find an estimate for $\beta*$ within the scope of the simulation itself, or by first performing a trial simulation with a fewer number of paths to obtain an estimate.

The control variate method is a very popular variance reduction technique. It is often the case that significant performance improvements can be achieved by applying a well designed control variate to a problem. One drawback of the method is that it requires that a control variate is well suited for the problem at hand.

Low discrepancy numbers

When a Monte Carlo- simulation is performed, one important question that needs to be answered is how to generate the random numbers used to simulate a path of asset prices.

Generally, the objective of the proposed generators has been to give a recipe for how to generate numbers that satisfy criteria fulfilled by truly random numbers. The generators have therefore aimed to provide numbers that are as "random" as possible.

Using truly random number is the logical starting- point for a Monte Carlo simulation. If there is a serial correlation between the drawn numbers, a mispricing of the derivative is likely to arise[6].

It may seem counterintuitive that many studies have found that random numbers are not always the best choice for the Monte Carlo- method. Quasi- random numbers are also known as low- discrepancy series. They are provided by generators that are designed to provide numbers that appear random in the sense that they fill the unit cube uniformly. These generators differ from the normal random generators since the numbers they provide do not cluster. They fill the room uniformly and evenly.

Low- discrepancy numbers do not aim to be serially uncorrelated. Instead, they aim to distribute the drawn number sequences evenly by taking the points in the domain that has already been sampled into account.

Consider for example the geometric Brownian motion, where paths are built using numbers drawn from the standard normal distribution. Any correlation between the numbers drawn to produce one path would lead to a regularity or bias in the paths. A derivative value on such a path would likely be mispriced.

However, if we interpret our function as a function of a vector of draws from the normal distribution, there is no reason why such underlying vectors cannot be serially correlated. Jäckel speculates that the reason implementations often have used serially uncorrelated numbers is that the same random number generator has been used to produced one- and more-dimensional random vectors. There are random number generators that take into account the previously drawn number vectors. The next drawn vector is correlated to the old in a way that distributes them to avoid the clusters and gaps of vectors that are pseudo- random.

Chapter 3

Analysis

In this section an analysis of the problem is given. It provides a foundation for the decisions taken in the design and implementation of the project Monte Carlo solver. The analysis is divided into four subsections. Every section corresponds to a topic that was analysed as part of a pre- design phase. Most conclusions were made prior to the design and implementation phases, although some were results of insights achieved in that work.

3.1 The user

The users of the Monte Carlo solver will primarily be QuantLab developers. Given that the developers model scenarios in QuantLang and C++, they are assumed to possess good programming skills. The users will probably value a flexibility in the system design higher than an environment where much of the functionality is fixed.

3.2 Implementation language

The possibility to include dll- files in the QuantLab development editions made C++ a good candidate for implementation language. The C++ language is a widely accepted programming language within and outside the financial industry. One determining reason C++ was chosen is that Algorithmica's clients are familiar with the language.

3.3 Flexibility

The project goal is a flexible Monte Carlo solver. I believe that a discussion on what it means for a solver to be flexible is motivated.

In this project, flexibility came to mean two things. First, it means the degrees of freedom a user has to define what simulation the solver performs. A solver that

allows the user to determine the properties of the simulation is preferred to one that does not.

A flexible Monte Carlo solver would allow its user to define the following properties of the simulation:

- The underlying asset.
- The derivative.
- What, if any, variance reduction techniques that should apply.
- What results of the simulation that are of interest.

The identified properties are discussed in the following sections.

3.3.1 Underlying Asset

The number of different assets used as underlying today is very large, and can only be expected to grow as new derivative products are introduced. In addition, there are often many different models that attempt to model the behaviour of a single asset. In a Monte Carlo simulation, it is the model of the behaviour of the underlying asset that is simulated. It is important to realise that different result may arise from the use of different models. Considering the plentyness of models of underlying behaviour, it is not a very good idea to specify the models that may be used in the simulation. A better approach is to find a good way to allow the user to specify her own model of the underlying.

When it comes to models of stocks and bonds, these often describe the behaviour with a SDE. Since stocks and bond are the most important underlying assets for Algorithmica, a user should be able to define the behaviour of the underlying as an SDE. If possible, behaviour defined in other ways should be allowed.

3.3.2 The derivative

When we perform a Monte Carlo simulation to value a derivative, we are interested in how to calculate its payoff from the path of the underlying. The payoff is a function of the path of the underlying over a defined time period. How the payoff-function is defined is decided either by the parties who sign the contract[1] or by a standard contract[2]. It would not be a good idea to tie the engine to certain types of payoff rules, considering the many alternatives. The client should therefore define the payoff rule.

[1] In the case of a OTC derivative
[2] In the case of a exchange- traded derivative

3.3.3 Simulation variance reduction techniques

Variance reduction techniques are of great importance for reducing simulation time in Monte Carlo simulations of financial derivatives[3].

The techniques considered in this paper, introduced in section 2.4.2, are dependant on the simulated derivative. The anti- thetic variate and control variate techniques both require knowledge of the valuation problem. When it comes to low-discrepancy numbers, several number generators are conceivable.

Since the user is responsible for defining both the model used for the underlying asset, and the rules for derivative valuation, it is reasonable that the user should also be responsible for choosing variance reduction techniques.

3.3.4 Simulation results

There are many conceivable results that can be derived the simulation of paths. The primary result is the risk- neutral expectation of the derivatives value. That is the statistical measure usually considered as the result of a simulation. However, there are other statistics that could be derived from the the paths that may be valuable. One example is the variance of the estimates of the derivative value. A high variance indicates a greater insecurity in the estimate.

It is possible that a user would like to value more than one derivative at the same time. If the underlying asset follows the same model, and the generated paths were reused, the computing resources needed to generate paths could be saved.

Not all potentially useful statistics derive from the derivative(s). The user may be interested in statistics of the simulated paths that do not affect the value of the derivative but still are considered useful. Such statistics could include the number of times the underlying asset crossed a barrier, or statistics on the underlying assets value at a certain time during the simulation.

As with the model used to describe the behaviour of the underlying asset, and the payoff- rule of the derivative, it is hard to give a complete description of what statistics may be of interest to the user. It is therefore reasonable to let the user define the statistics herself, if possible.

3.4 Functionality

Prior research on Monte Carlo simulation often has been focused on variance reduction techniques rather than design of a reusable solver. Although this study

[3]See section 2.4.2

has a different focus, there is still much to learn from the way the solvers in other studies have been implemented. I found that there existed common features in the implementations that I could use as a template for the backbone of my own solver. By *template* I mean a set of basic functionality. I do not make any assumptions on how the functionality was made available by the system, or how it was implemented. The template only defined what information was necessary to carry out the simulation and the basic steps that a simulation consists of.

A Monte Carlo application typically performs a number of tasks[4]. In order to perform the tasks, it needs some information on the problem to be solved. At least, the following information is needed:

- The time period over which the simulation is to take place.
- How the underlying asset behaves over that time period.
- How to calculate the derivative's payoff, given the path of the underlying asset.
- When to stop the simulation[5]

In the project work, the functionality defined by the template became a skeleton for the solver. The skeleton roughly defined tasks that were to be performed by the solver. The tasks are:

- Construct paths of the underlying assets properties over the defined time period.
- Value the derivative on the paths.
- Return the results of the simulation to the user.

In a way, the level of generality offered by a Monte Carlo solver could be defined as the possibility for a user to adjust how to provide the necessary information and how the solver performs those tasks.

The following sections present the design and implementation of the Monte Carlo engine.

[4]Which are reviewed in section 2.4.1
[5]Such as the number of simulations to be performed

Chapter 4

Results

This chapter presents the Monte Carlo solver. The section consists of two subsections. In section 4.1 is a high-level discussion on the solver. The section discusses how the solver uses the concepts of nodes, path and valuation. Section 4.2 presents the implemented solver. It gives a description of the solver's classes and how the classes collaborate during a simulation.

4.1 Concept overview

This section provides the reader with an overview of how the solver relates to the concepts of nodes, paths and valuation. The high-level discussion is meant to help the reader understand how the implemented classes in section 4.2 work together.

Nodes and Paths
The solver regards a path as a set of nodes that are defined at times in the simulation time period. The time over which the underlying is to be simulated is considered discrete. That means that for every member of the set[1], there is a node in the path of the underlying asset.

A node represents a step in a simulated path. The path is constructed by using the time series that describe the behaviour of the underlying. In its simplest form, the node holds the price of the asset at that time. It may, however, contain more complex information, such as the instantaneous volatility of the asset. A node keeps information needed for two purposes:

- During a simulation, it stores the information needed to build subsequent nodes in the path.
- It stores the information needed to value the derivative.

[1] corresponding to a point in time

It may of course be very difficult to predict what information a user of the system would like to store in the nodes. The information is dependent on the type of time series involved in the simulation. One possibility is that the node will contain information for more than one time series. That way, more than one time series could progress simultaneously.

Allowing several processes to progress simultaneously has its benefits. It means that derivatives with more complex underlying assets and pay-off functions can be simulated. For example, the underlying could be defined as a basket of stocks or bonds, and the pay-off function could include discounting by a rate that itself follows a stochastic differential equation.

Path development
Now that we have discussed what a node is and what information it may hold, we are ready to examine how to build a path of nodes. What we want to achieve is a path of nodes that contain the data that resulted from applying the SDEs that define the time series to an initial state.

Prior to building a path, an initial state must be defined. It is from this initial state the path is developed. In the design, the initial state is a node. From that node, the start node, subsequent nodes in the path are built by a step-by-step process. I believe that the easiest way to explain path progression is to understand how the second node in the path is constructed from the first. The third node is then built from the second in a similar way.

Assume that the user has defined the initial node and that we are about to construct the second node in the path. The first node contain start values for the time series involved in the simulation. When we traverse the path from the first node to the second, we apply the SDEs that define the time series to the data in the first node. This is done for every time series. When completed, we have calculated the data in the second node. The computations that are carried out depends on what the underlying asset is and how the user has defined its behaviour.

Path evaluation
When a path has been built, it is used to evaluate one or more derivatives. Once built, we store the nodes that we need to evaluate the derivative. Evaluation of the derivative can be done either for every path directly after it has been built, or for all the paths together after the path simulation stage is done. Each choice has its benefits and drawbacks.

If the evaluation is to be done after every path simulation, the results need to be stored until all the paths have been simulated and evaluated. An average can then be taken over the samples.

If the evaluation is performed when all paths have been simulated, they need to be stored until evaluation can be performed. This may require a bigger memory, even though the memory demand can be decreased by only saving the nodes needed to evaluate the derivative rather than the whole path[2]. A benefit is that one function call is enough to evaluate the derivative over the paths.

Even though the amount of work that needs to be done to perform the evaluations of the paths is the same, the first approach requires more function calls but not as much memory. In the project Monte Carlo engine, the second approach has been applied. The assumption was made that in most cases few enough nodes are stored in each path to make it possible to store them until all paths have been simulated[3].

4.2 The implemented solver

This section presents the implemented Monte Carlo solver. What follows is a description of the implemented system. First, the implementation of the key conceptual parts of the solver are presented and discussed individually. After that, an account of the complete system is given, complete with sequence diagrams that show how the system parts collaborate to perform a simulation.

4.2.1 Time

Time is an essential building block of a Monte Carlo engine. It is used to define the period for which the derivative is written, as well as a parameter when cash flows are discounted. There is some controversy over the way time is to be defined when a financial model of an asset is constructed. One example is whether the value of an asset changes when the market it is trading on is closed. There is also the question of how the user of the system would like time to be presented. Is the time frame of a model supposed to be altered by the use of a calendar, or is time a number?

I believe that the most general way to describe time in this frame work is as a number. There are several benefits with this representation:

- Many of the existing models treat time as a number. A transparency is achieved when time is represented in the same way in the engine. This transparancy makes it easier for new users to adapt.

[2]This is usually a big improvement. It is common that the value of the derivative is calculated from a small subset of the nodes.

[3]Ideally, the user would be allowed to choose the approach to statistics gathering. That is currently not implemented as part of the project.

- The representation of time as a number facilitates discounting cash flows.
- If is to be presented to the user as a calendar or any other format, little adjustment needs to be done.
- It is probable that performance benefits from time being a number. Other formats are likely to result in many convertations to time as a number, as many models regard time as a number.

In the implemented solver, time is a **double**.

4.2.2 The Node

For Monte Carlo simulation, we need a path of the underlying asset to perform the valuation of the derivative. A path is a set of nodes, where every node defines one step in the path[4]. Every node contains the information necessary to:

- Build subsequent steps in the path.
- Value the derivative, once the path has been constructed.

Since the information contained in the nodes determines what time series can be modelled, and what statistics that can be gathered, it is important that the user of the system is allowed to control it.

The solver's way to deal with the problem of user- defined nodes is to allow the user to define the node as a C++ class. That way, the user has a great degree of freedom in how to specify the nodes. The user may choose to let the node be a simple container for a set of data, or a more complex class that allows for function calls and other C++ features.

The user of the system specifies a node class which, along with function pointers that define the time series and what nodes are to be sampled for statistics, is used to create a path. The class responsible for the path creation, $TProcess$, is introduced in section 4.2.3.

4.2.3 The Time Series

As stated in section 2.3, models of asset behaviour are often considered to be stochastic. Besides containing an element of randomness themselves, the models often include parameters such as interest rates or volatility which also are modelled as time- series. Therefore it is intuitive to define the behaviour of the underlying as a collection of time series. The processes are used to build a path of states over

[4]

a number of time steps. A path is built from a set of an initial node that contains start values for the processes. From the start node, the second node in the path is built by applying functions that represent the SDEs. The functions may very well depend on each other, as long as they are applied in an order that ensures that data needed as input for a function is available at the time of computation.

In the project's Monte Carlo solver, there is a class that is responsible for creating paths. The class, $TProcess$, is used to simulate paths of the underlying asset. It provides the function $OnePath$, used to simulate one path of the underlying and to store the nodes in that path that are used to value the derivative. Figure 4.1 is a UML class diagram of the $TProcess$ class.

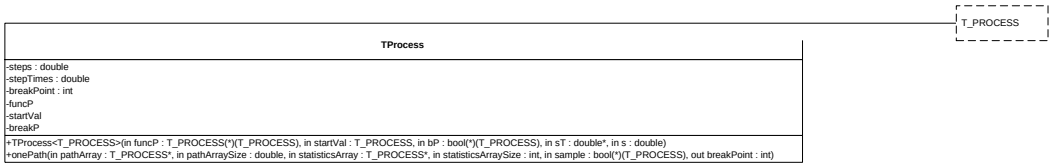

Figure 4.1. The process template class

TProcess is implemented as a template class to allow the user to call it with any class (i. e. the user- defined node class). When it is used to create a class it takes the following parameters:

- $funcP$, a function pointer to a function that takes an object of the TPROCESS template parameter and return an object of the same type. The function describes how the stochastic process develops step by step.
- $startVal$, an instance of the user- defined node class. $startVal$ is used as the initial node in the simulated paths.
- bP, a function pointer to a function that defines break- conditions. The function is called with an instance of the user defined node class and returns a bool indicating whether the simulation should continue or return with a default value.
- $stepTimes$, an array of doubles that contain the points in time that the nodes will be defined on.
- s, a double indicating the number of nodes in a path.

The TProcess template class also provides the function $OnePath$. As the function name suggests, it builds a path of nodes. $OnePath$ is responsible for the measures that needs to be taken during path construction. Besides building the

path itself it stores the nodes that are needed for statistics. OnePath takes the following parameters:

- *pathArray*, an array of nodes. The array will hold the nodes in the simulated path. For performance reasons, we reuse the same array every time we simulate a path.
- *pathArraySize*, an integer holding the size of the pathArray.
- *statisticsArray*, an array of nodes. The array will hold the nodes in the path that are used for statistics.
- *statisticsArraySize*, an integer holding the size of the statisticsArray.
- *sample*, a pointer to a function that takes a node and returns *true* if that node is to be sampled for statistics.

The *OnePath* function builds the path by repeatedly applying the function that $funcP$ refers to. After every step, the *sample* function is called on the node to determine whether the node should be part of the set on which valuation of the derivative is done.

As a result of the process, two vectors of nodes are filled. The *pathArray* vector holds all the nodes in the path, while *statisticsArray* only hold the nodes of interest for valuation. The nodes in the second vector will also be stored after every path simulation, while the nodes in first will not. The motivation of having two vectors rather than storing the whole path is that it is common that the derivative is not valued on the whole path[5]. The memory saved by storing only the nodes of interest allows the user to simulate more paths, given a fixed- size memory. The optimisation does require the user to be careful when defining the *sample* function, since the *statisticsArray* has a fixed size.

In a Monte Carlo simulation, we would like to use the results from a number of simulations for statistics. The repetitive character of the method means that we would like to store the results from every simulation in a practical manner until we are ready to draw conclusions. The class $TStatistics$, introduced in section 4.2.4 facilitates the storage of results and may be used to obtain statistical measures from collected data.

4.2.4 The Statistics

At every node in a path, there might be useful information[6]. As a consequence, this information needs to be either stored for future use or passed on to an object that

[5] It is usually valued on a small subset of the nodes in the path

[6] For example, the value of the underlying or an indicator that the simulation is to stop

knows what to do with it. In the design, there is a class, *TStatistics*, that stores sampled data for future use.

TStatistics is a template class that facilitates the storage of results from the individual simulations. It is also responsible for compiling statistics on the simulations requested by the user. The template class takes two classes as its input. The first is the user- defined node class. The second is a class that defines how the user would like to receive the statistical output.

The *TStatistics* template class takes the following parameters:

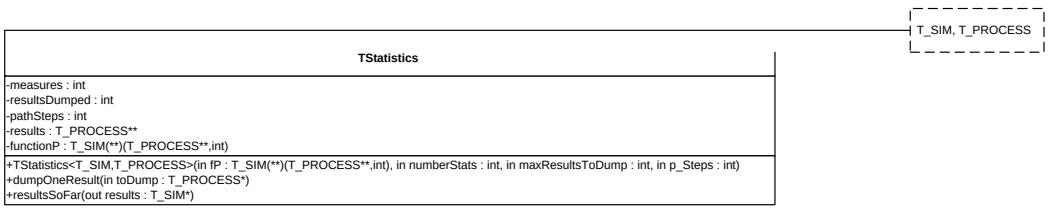

Figure 4.2. The statistics template class

- *measures*, an integer stating the number of statistics to be stored.
- *resultsDumped*, int integer holding the number of results posted to the class so far.
- *pathSteps*, an integer stating the number of nodes in a posted result
- *results*, an array of results from the simulations.
- $functionP$, an array of pointers to functions that are used to calculate statistics on the posted results.

TStatistics provides two functions. The first, *dumpOneResult*, is used to build a foundation for statistics by repeatedly posting results from simulations. The second, $resultsSoFar$, is used to compile statistics on the posted results. The two functions are presented below.

The *dumpOneResult* function is used to build a collection of results from Monte Carlo simulations. It takes a vector of user- defined nodes as a parameter and stores its content for future use. As more results are stored, the matrix of user- defined nodes is growing.

The $resultsSoFar$ function applies the statistics functions in the $functionP$ array to the stored results. It calls the functions on the *results* vector, along with

the parameter $resultsSoFar$. The output of the function is a vector of the user-defined return type.

The TStatistics class is nothing but a convenient way to put together two related tasks- the storage of outcomes from the experiments and the calculation of statistics.

4.2.5 The Monte Carlo Engine

Typically, a user would not be expected to call $TProcess.onePath$ and $TStatistics.dumpOneResult$ herself. Instead, a user would define a time period, an underlying and a rule for how to value a derivative. She would also define criteria for when enough simulations have been performed. After giving the initial directions, she would expect the Monte Carlo solver to take care of the rest. So far, we have discussed the two main parts of a Monte Carlo solver. In this section, the MCEngine class is introduced. The class is responsible for the high- level tasks of a simulation.

The Monte Carlo engine is a template class that takes the user- defined node class as a parameter, along with function pointer that define the time series and how to gather and calculate desired statistics.

The MCEngine template class ties together the functionality provided by the $TProcess$ and $TStatistics$ classes. It provides the function *simulate* that carries out the whole Monte Carlo simulation and returns the requested statistics.

The MCEngine template class has no member variables. Its sole purpose is to provide the *simulate* function. The *simulate* function takes the following parameters:

- *process*, a TProcess<T_PROCESS> class, defining the stochastic process used as to model the behaviour of the underlying.
- *statistics*, a TStatistics<T_SIM,T_PROCESS> class, defining the format of the statistics to be gathered.
- $numberOfPaths$, an integer defining the number of paths to be simulated.
- $stepsInPath$, an integer defining the number of steps in ach simulated path.
- $stepsForStats$, an integer defining the number of nodes in each simulated path that are used to value the derivative.
- *sample*, a pointer to a function that is used to determine whether a node in the simulated path is to be kept for statistics.

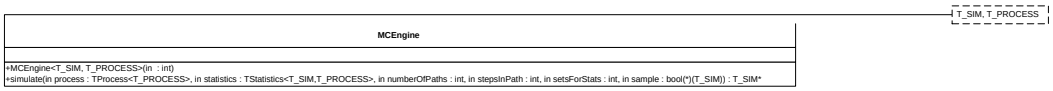

Figure 4.3. The MCEngine template class

4.2.6 A simulation sequence diagram

What happens when a simulation is performed? Figure 4.4 is a sequence diagram showing the interaction between the different parts of the system. The user starts the simulation by calling the *simulate* function of a *MCEngine* instance. It repeatedly calls for paths to be simulated and posts the nodes of interest to a *TStatistics* instance. After the paths have been simulated, *MCEngine* calls on *TStatistics* for the results of the simulation. The results are then returned to the user.

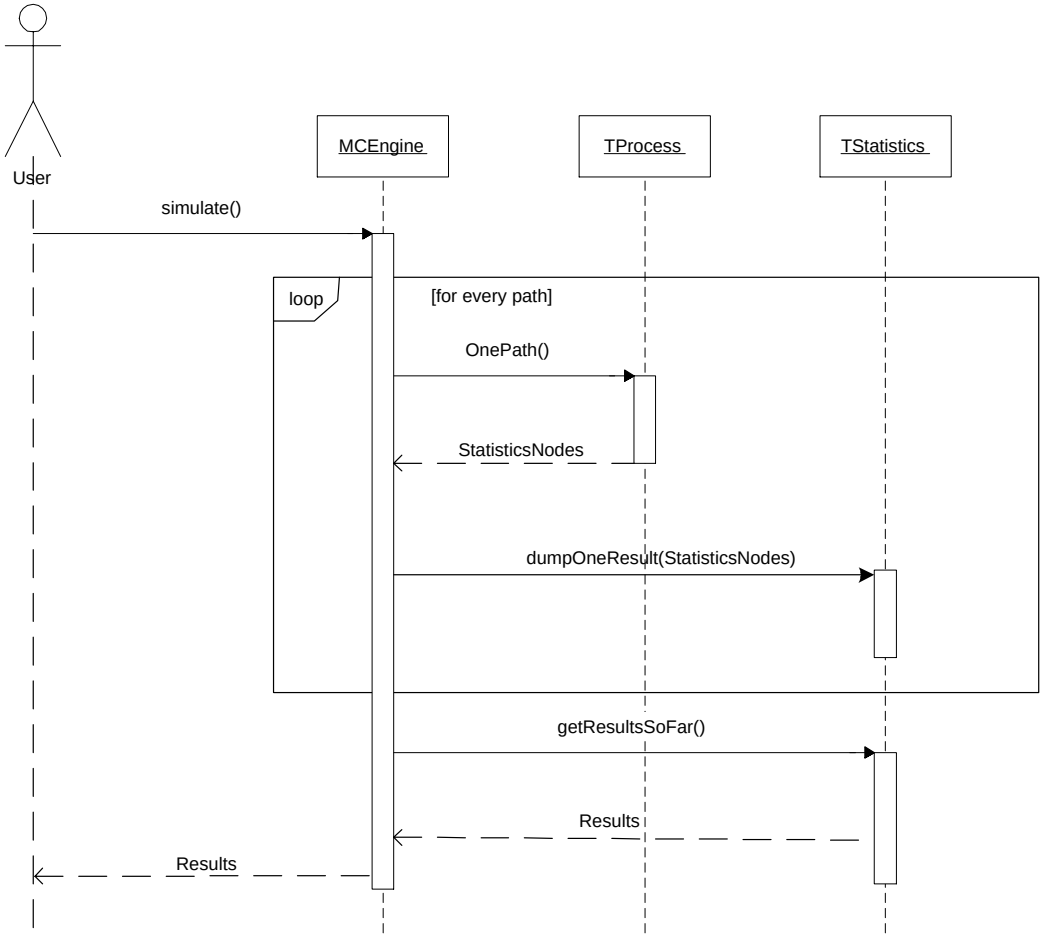

Figure 4.4. A Monte Carlo simulation

Chapter 5

Examples

This chapter contains an example of how to use the Monte Carlo solver. We will use the engine to value two derivatives on a stock. The lifetime of both derivatives is one year. The first derivative is an ordinary call option. The second derivative is an Asian call option that uses the arithmetic mean over the last three and a half months of the year. A specification of the stock model and derivatives used is given below

5.1 The problem

5.1.1 The underlying asset

The underlying is a stock that is assumed to follow a process defined by the Black-Scholes model. It is assumes to be defined by the following parameters:

- The variance $\sigma = 0.30$
- The drift $\mu = 0.05$

The model given by Black and Scholes is therefore

$$dS = 0.05Sdt + 0.30Sdz, z \in N(0, 1) \tag{5.1}$$

For simplicity, we assume that the initial price of the stock is 1 (USD, EUR or other suitable currency). The underlying is simulated over the year in 100 steps of equal size. A year is assumed to have 365 days. Thus, every step corresponds to a period of 3.65 days. Every day is counted, not only the days when the market is open for trade.

5.1.2 The call option

The call option (see section 2.1) has a strike price of 1.

5.1.3 The Asian call option

The Asian option has a strike price of 1 and takes the arithmetic average over the last 109.5 days. This corresponds to the last 30 steps of the asset's path.

5.2 Setting up the solver

This section illustrates how the input for the Monte Carlo engine is set up.

5.2.1 The node

For a Black- Scholes model, a node only needs to keep the asset price. In addition, the node class used in the simulation also keeps a variable for time. This time variable is used for sampling decisions. The node class used in this example is called *Step*. A UML class diagram of the *Step* is found in figure 5.1

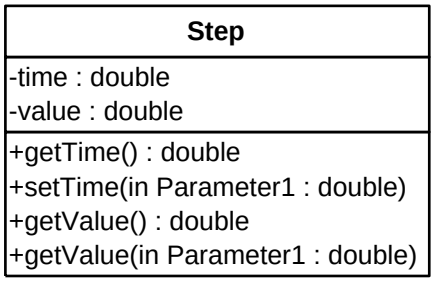

Figure 5.1. The node class

5.2.2 The underlying

The underlying is described by equation 5.1. It is passed to the Monte Carlo engine as a pointer to the function gbm:

```
Step gbm( Step prevVal){
    double dt = 0.01;
    double r = 0.05;
    double vol = 0.3;
    double t = 0;
    for(int i = 0; i < 12; i++){
```

```
t+= (double)rand()/RAND_MAX;
    }
    t = t-6;

  prevVal.setValue( prevVal.getValue()*( 1 + (r*dt + vol * t* sqrt(dt))));
   prevVal.setTime( prevVal.getTime() + dt);
   return prevVal;
}
```

5.2.3 The call option

The call option is a statistics to calculated on the sampled data from the simulations. For every simulated path, it calculates the value of the call option with strike price 1 on the last node.

```
double call(Step** results, int numberOfResults){
double temp = 0;
for(int i = 0; i<numberOfResults; i++){
    if( (results[i][STAT_STEPS-1].getValue() -1> 0))
    {
temp += ( results[i][STAT_STEPS-1].getValue()-1);
    }
}
    return temp/numberOfResults ;
}
```

We define STAT_STEPS to be the number of nodes sampled for each path[1].

5.2.4 The Asian call option

The Asian call option is also a statistics that is calculated from the gathered data. It is more complicated to calculate, as it contains a calculation of an average.

```
double call30(Step** results, int numberOfResults){
        double temp = 0;
for(int i = 0; i<numberOfResults; i++){
    double val = 0;
    for(int j = 0; j < 30; j++){
val += results[i][j].getValue();
    }
    val = val/30;
    if( val -1> 0)
    {
```

[1] In this example, STAT_STEPS will be 30.

```
temp += ( val-1);
    }
}
    return temp/numberOfResults ;
}
```

5.2.5 The sampling function

We want to sample the last 30 nodes of every path. If we sample fewer nodes, we will not have enough data to calculate the value of the derivatives. If we sample more nodes than necessary, we will store unnecessary information and waste memory. The sampling function is used when a path is built to fill the statistics- array[2]. The sampling function used in the simulation is given below:

```
bool sample30(Step s){
    double d = 0.5;
    double st = s.getTime();
    if( st >= 0.685 ){
return true;
    }
    else{
return false;
}
}
```

5.3 The results

The simulation was carried out with an increasing number of paths for both the derivatives. The results of the simulation for the call option and the Asian call option are given in diagrammes 5.2 and 5.3 respectively.

[2]See section 4.2.3

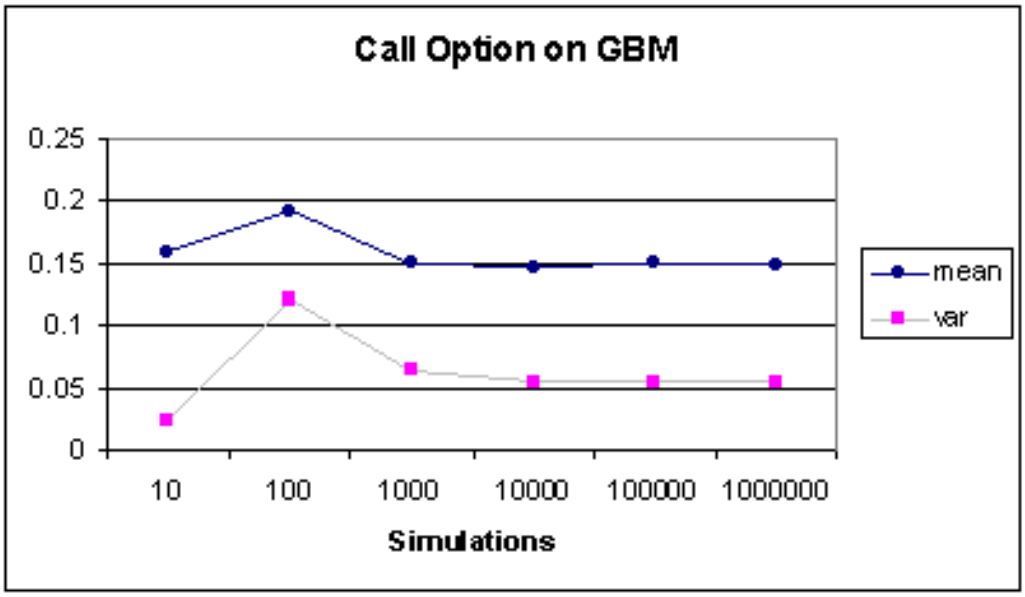

Figure 5.2. The call option

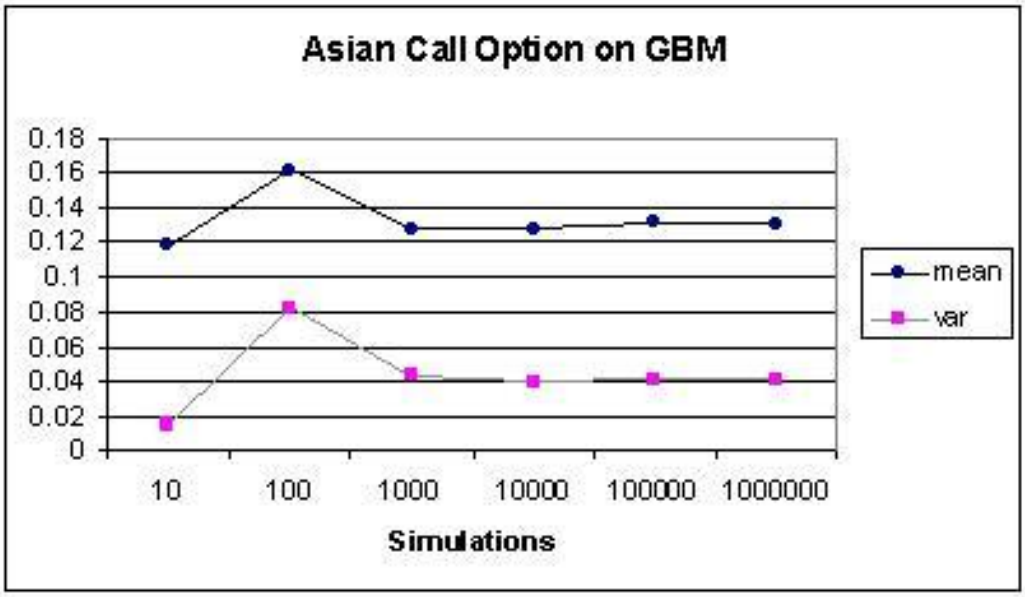

Figure 5.3. The call30

Bibliography

[1] Fischer Black and Myros Scholes. The pricing of options & corporate liabilities. *The Journal of Political Economy*, 1973.

[2] Chicago Board Options Exchange, march 2005.

[3] Cox and Ross. The valuation of options for alternative stochastic processes. *The Journal of Financial Economics*, 1976.

[4] D. Heath, R. Jarrow, and A.J Morton. Bond pricing and the term structure of interest rates: A new methodology for contingent claims valuation. *Econometrica*, 1992.

[5] John C Hull. *Options, Futures and Other Derivatives*. Prentice Hall, 2003.

[6] Peter Jäckel. *Monte Carlo methods in finance*. Wiley, 2002.

[7] Boyle P. P. Options: A monte carlo approach. *Journal of Financial Economics*, 1977.

[8] Stockholmsbörsen, march 2005.