

**END OF STUDIES PROJECT REPORT IMAFA2 2013/2014**

**QuantLib 1.3**

**LIBRAIRIE QUANTLIB 1.3:** DESIGN AND IMPLEMENTATION OF NEW OPTIONS’ PRICING ENGINES.

|  |  |
| --- | --- |
| **Presented by:** | **Supervised by:** |
|  |  |
| Ahmed AYADI  Nathan D.KRUCK  Nolan POTIER | Mme. Anne Marie HUGUES  &  Mr. Luigi BALLABIO |
|  |  |

Index

[Chapter I Introduction 2](#_Toc377383794)

[Chapter II QuantLib Architecture 3](#_Toc377383795)

[Chapter III Barrier Options: 6](#_Toc377383796)

[I. Partial-Time Barrier Options 7](#_Toc377383797)

[Chapter IV Netography 7](#_Toc377383798)

Table of Figures

[Figure 1 Instrument Class Inheritance 3](#_Toc377384014)

[Figure 2 PricingEngine Inheritance Graph; Example Binomial Vanilla Option Engine [6] 4](#_Toc377384015)

[Figure 3 Net present value calculus Sequence Diagram [7] 5](#_Toc377384016)

[Figure 4 Barrier Options versus Vanilla Options 6](#_Toc377384017)

# Introduction

**QuantLib** is a C++ library that eases writing applications for quantitative finance. In its turn, it uses the **Boost** library **[**[1](http://www.codeproject.com/Articles/4496/An-Introduction-to-Boost)**]**. Since its publication in November 2000, it has continued to grow in popularity in the financial sector. QuantLib is an open source project, it has many advantages. First of all, it is free. It is also thoroughly tested. This is due to the collective intelligence of all users and developers. All errors are quickly detected and corrected. Finally, it is easy to extend because the source code is freely available, the developer can learn from the implementation of its functionality. QuantLib is structured into modules [[2](http://quantlib.org/reference/modules.html)]. Each module covers a distinctive aspect of all features, and contains a set of components.

Our work involves the mastery of the use of existing pricing engines, in the first place and the design and development of others, in the second. So our planning for the project will cover a learning period including documentation reading and testing of pricers, then comes the other part of our solution implementation.

# QuantLib Architecture

The QuantLib architecture has been made carefully make it easy to use and extend by adding new pricing functionalities. For this it is necessary to know the two main classes of the library.

From one hand, the abstract class « Instrument » that allows to generalize financial instruments like Vanilla Option with its different parameters (volatility, strike, maturity …). From the other, we have the « PricingEngine » abstract class that allows to calculate the instrument price.

It’s necessary to understand the use of these two classes for any given instrument. Notice that it is not always the case that a unique pricing method is used; one might want to use multiple methods for the same instrument.

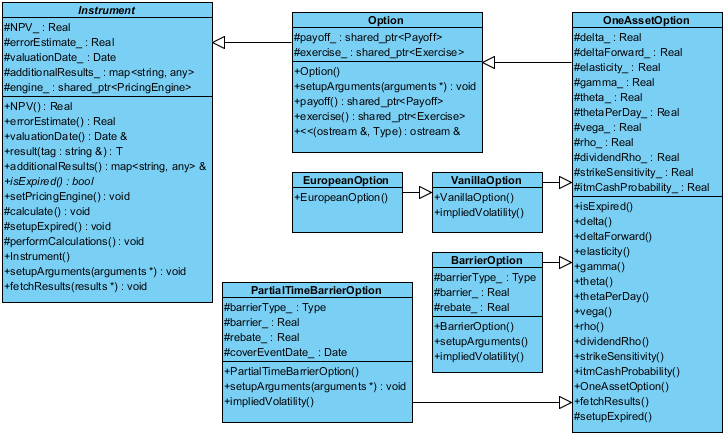


Figure 1 Instrument Class Inheritance

The figure 1-(Elaborated using UML tool Visual Paradigm)-shows a part from QuantLib Class Diagram.

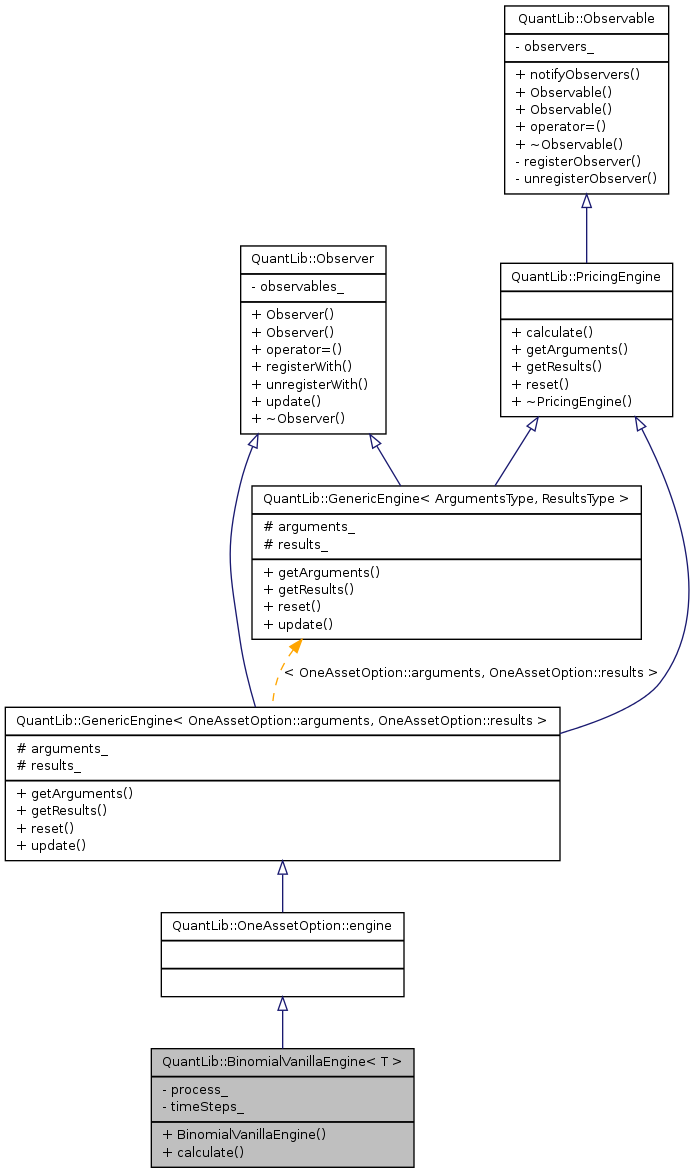


Figure 2 PricingEngine Inheritance Graph; Example Binomial Vanilla Option Engine [[6](http://quantlib.sourcearchive.com/documentation/1.1-1/classQuantLib_1_1BinomialVanillaEngine.html)]

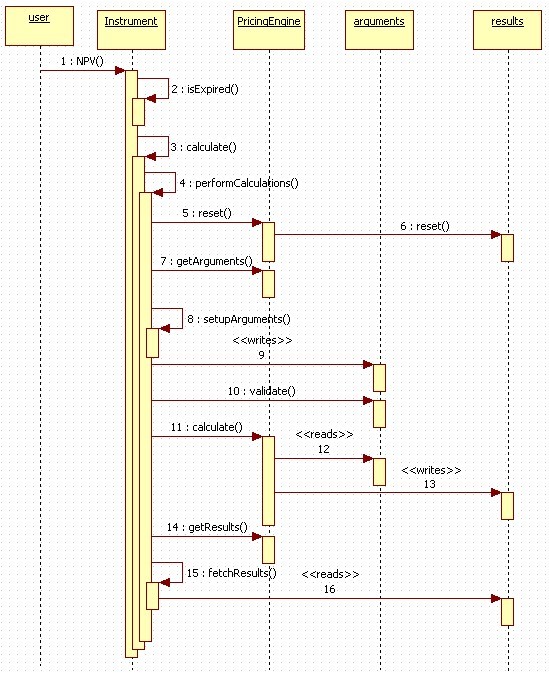


Figure 3 Net present value calculus Sequence Diagram [[7](http://quantlife.tistory.com/category/quantlib?page=2)]

The figure 3 shows the different exchanges between the objects to calculate the net present value (NPV) of an instrument.

# Barrier Options:

Barrier option is a path-dependent option where the price is based on the fluctuations in the underlying's value during all or part of the contract term. It was traded in the late 60s and it has been used to manage the risk. It comes in many flavors and forms, but its key characteristic is that this type of option is either initiated or exterminated upon reaching a certain barrier level; that is, it is either knocked in or knocked out.

“A basic American option is one type of **path dependent option**. Because it can be exercised at any time prior to expiration, its value will change as the underlying asset's value changes. An Asian option, also called an average option, is another type of path dependent option, because its payoff is based on the average price of the underlying asset during the contract term. Similarly, a barrier option would be considered a path dependent option because its value changes if the underlying asset reaches or surpasses a specified price. The lookback option and Russian option are also path-dependent options.”[[3](http://www.investopedia.com/terms/p/pathdependentoption.asp)]

In our work we consider the most basic type of barrier option; “**the single barrier**”. This option comes in 4 types where each type can be a Call or a Put option:

* Up & In
* Up & Out
* Down & In
* Down & Out

An "In" barrier option, for example, means that it becomes active once crossing the barrier level. If the underlying asset fails to cross the barrier, the Barrier Option you bought becomes worthless piece of paper upon expiration even if the underlying asset is trading above ( call option ) or below ( put option ), its strike price!

So it’s clear, at first thoughts, that Barrier Options are more dangerous than normal plain vanilla options. Then why would investors use them? Basically because they carry a much **lower extrinsic value** than plain vanilla options. [[5](http://www.optiontradingpedia.com/barrier_options.htm)].

|  |  |  |
| --- | --- | --- |
|  | Advantage | Disadvantage |
| Knock-In Barrier Options | 1. Cheaper, resulting in higher profits 2. Ideal for speculating huge moves | 1. Higher risk of loss if underlying asset moves moderately 2. Commonly traded for forex, not stocks. |
| Knock-Out Barrier Options | 1. Cheaper, resulting in higher profits 2. Ideal for speculating small moves | 1. Higher risk of loss if underlying asset rallies 2. Commonly traded for forex, not stocks. |

Figure 4 Barrier Options versus Vanilla Options

There are various barriers types including Parisians, double barriers, and partial time barriers. Here, we discuss partial-time barrier options

## Partial-Time Barrier Options

The watching period for a barrier crossing is restricted to a fraction of the option’s lifetime. Here we have two types: **partial-time-start** and **partial-time-end**.

The monitoring period of **Partial-time-start** barrier options starts at time zero and ends at a random date before expiration. As for **partial-time-end** barrier options, they have the monitoring period start at an arbitrary date before expiration and end at expiration.

There are two classes of **partial-time-end** barrier options [[4](http://hosho.ees.hokudai.ac.jp/~kubo/Rdoc/library/fExoticOptions/html/BarrierOptions.html)]: Type **B1** is defined such that only a barrier hit or crossed causes the option to be knocked out, and hence there is no difference between up and down options. Type **B2** options are defined such that a down-and-out call is knocked out as soon as the underlying price is below the barrier. Similarly, an up-and-out call is knocked out as soon as the underlying price is above the barrier.

# Implementation

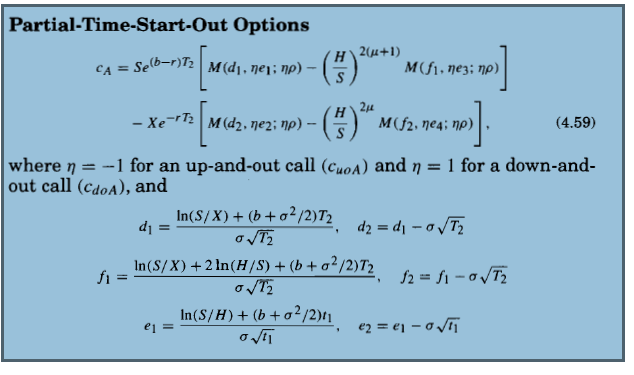
The first phase of our work, concerned the reading of the available library documentation and its installation and configuration on our computers. Also we had to read about its Barrier Options.

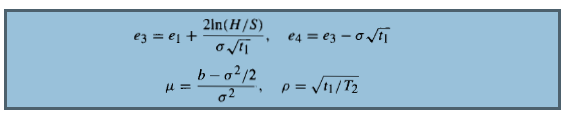
To implement the barrier option our first idea was to reuse the class “OptionBarrier” included in the library and combined it with a “VanillaOption”. But after had understood the architecture of the library it was easier to implement the solution directly with “PartialTimeBarrierOption” class inheriting from the “OneAssetOption” class. So we elaborated a class diagram to represent the new class and hence to explicit the relations with the other classes needed for the solution (see Figure 1: Instrument Class Inheritance).

Once the architecture was respected we had to implement the pricing method to the Partial-Barrier Option. For this we have used formulas of the book: "The Complete Guide To Option Pricing Formulas by Espen GaarDer Haug".

Example of formulas implemented:

|  |  |
| --- | --- |
| **S: underlying**  **H: barrier**  **X: Strike**  **T2: expiring date** | **t1: barrier window start or stop**  **b: dividend yield**  **r: interest rate**  **σ: volatility** |





# Netography

[1] Boost Library: <http://www.codeproject.com/Articles/4496/An-Introduction-to-Boost>

[2] QuantLib1.3 modules: <http://quantlib.org/reference/modules.html>

[3] <http://www.investopedia.com/terms/p/pathdependentoption.asp>

[4] <http://hosho.ees.hokudai.ac.jp/~kubo/Rdoc/library/fExoticOptions/html/BarrierOptions.html>

[5] <http://www.optiontradingpedia.com/barrier_options.htm>

[6] [http://quantlib.sourcearchive.com/documentation/1.11/classQuantLib\_1\_1BinomialVanilla Engine.html](http://quantlib.sourcearchive.com/documentation/1.11/classQuantLib_1_1BinomialVanilla%20Engine.html)

[7] <http://quantlife.tistory.com/category/quantlib?page=2>