MSc Mathematics in Finance

Imperial College London

Authores:

Ghali Tadlaoui

Siu Law

Lington Meng

Jonathas Castello Branco

Faycal Fassi Fehri

**USER GUIDE**

Project: Simulation Methods in Finance

European and Asian option pricing

Team 5

# Introduction

Thanks for testing out our project. This code was designed and implemented by the team 5 of the MSc in Mathematics and Finance for the Numerical Simulation Methods class. Our objective was to apply numerical and simulation methods to price and perform sensitivity analysis on European and Asian options.

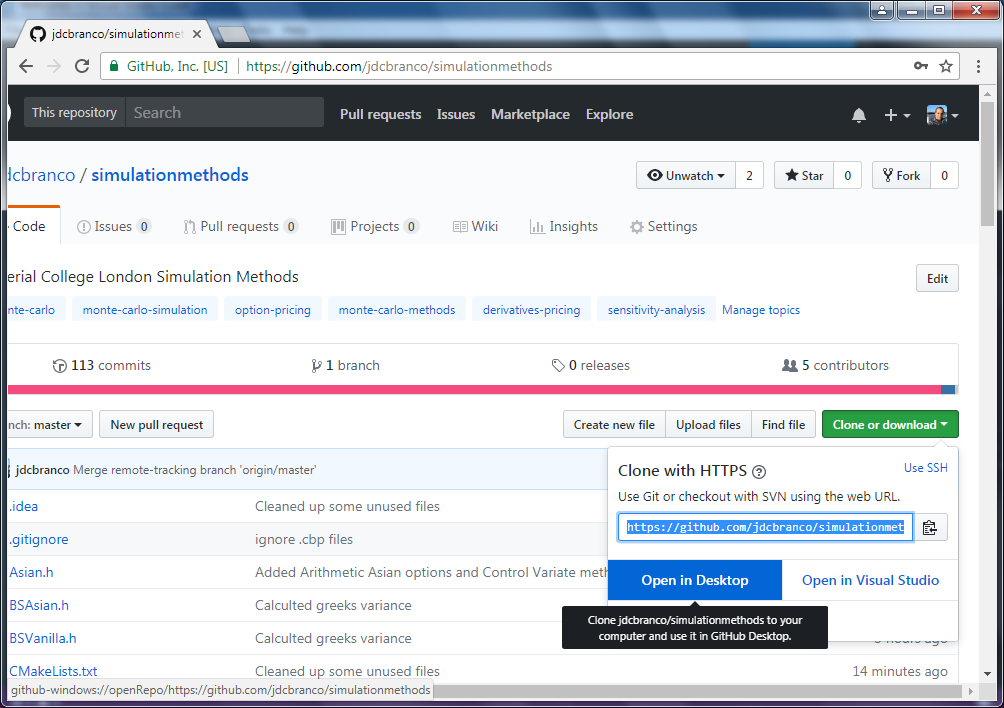
This project was done in C++ 11, so if you try to compile with your own compiler, make sure you use the –std=c++11 option. It was originally develop using CLion and Eclipse – each team member was free to use their own C++ IDE. However, we’ve included a Visual Studio solution file so you can also use Visual Studio to run our code.

Thus in order to run this project, you have a few options. First you can start extracting the zip file provided, or you can download the code directly from github.

After that, you can open it in your preferred C++ IDE – common choices being the Visual Studio 2013 (from Microsoft) or CLion 2017.3 (from Jetbrains).

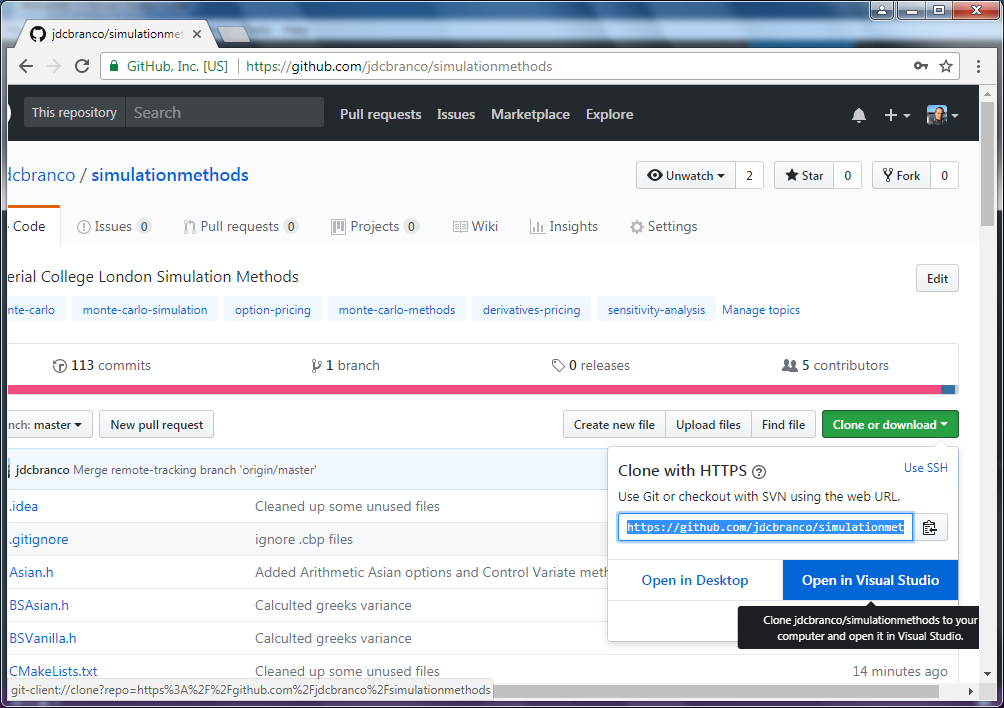
**Download using github desktop**

You may have received a zip file containing all the code. If not, you can also download it from github.com/jdcbranco/simulationsmethods



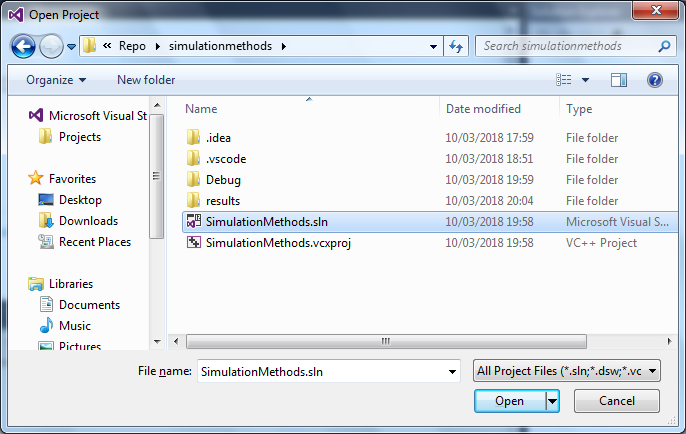
**Download using github but open directly in Visual Studio**

If you prefer to work with Visual studio, it is also possible to download and open it from the github website, by choosing the “Open in Visual Studio” choice, as demonstrated below. Note we have added support only for Visual Studio 2013 on this project.



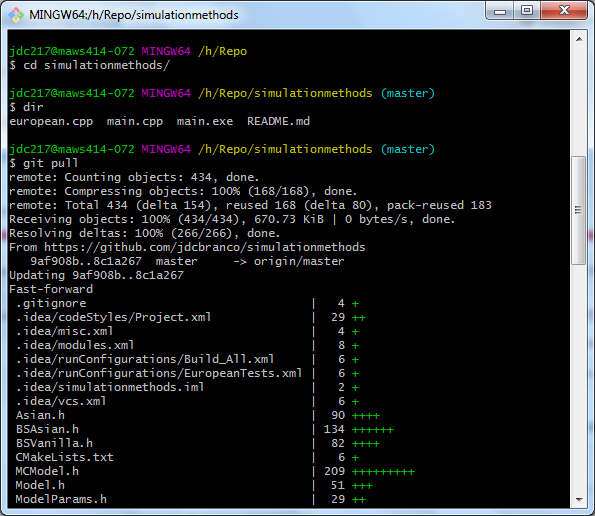
**Open the Visual Studio solution file**

Once you have downloaded and extract the project into your machine, you can head directly to the SimulationMethods.sln file, in order to open it in Visual Studio 2013.



**Refreshing your files**

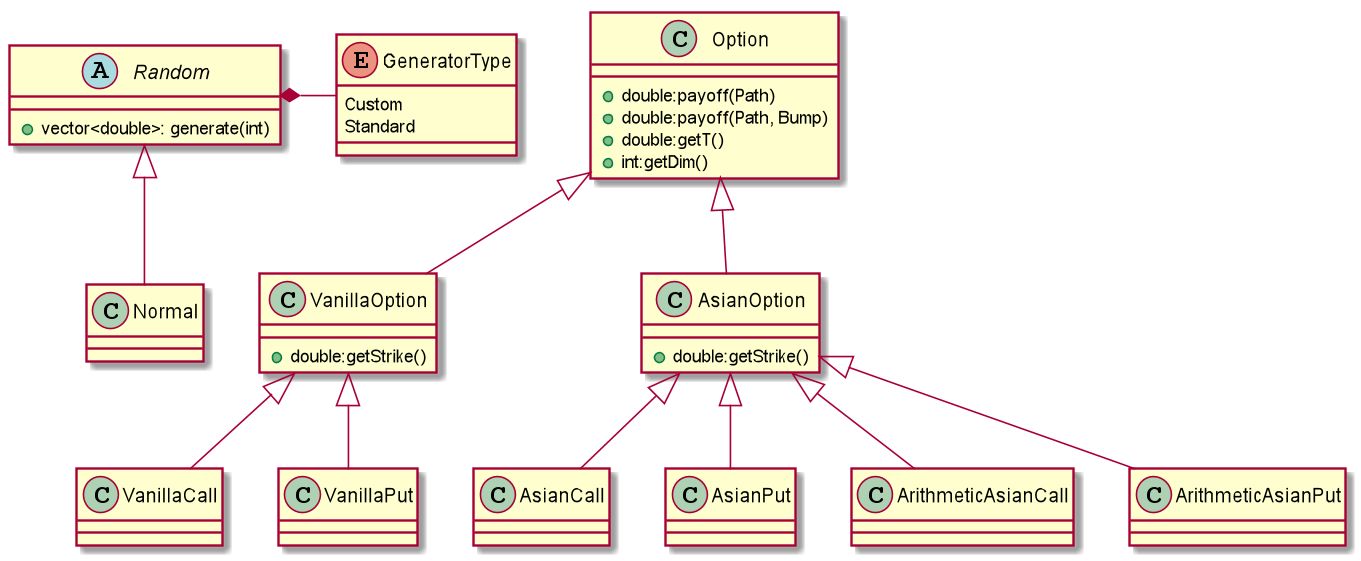
If you used git to download our code, you can run “git pull” in order to refresh and download the latest version.



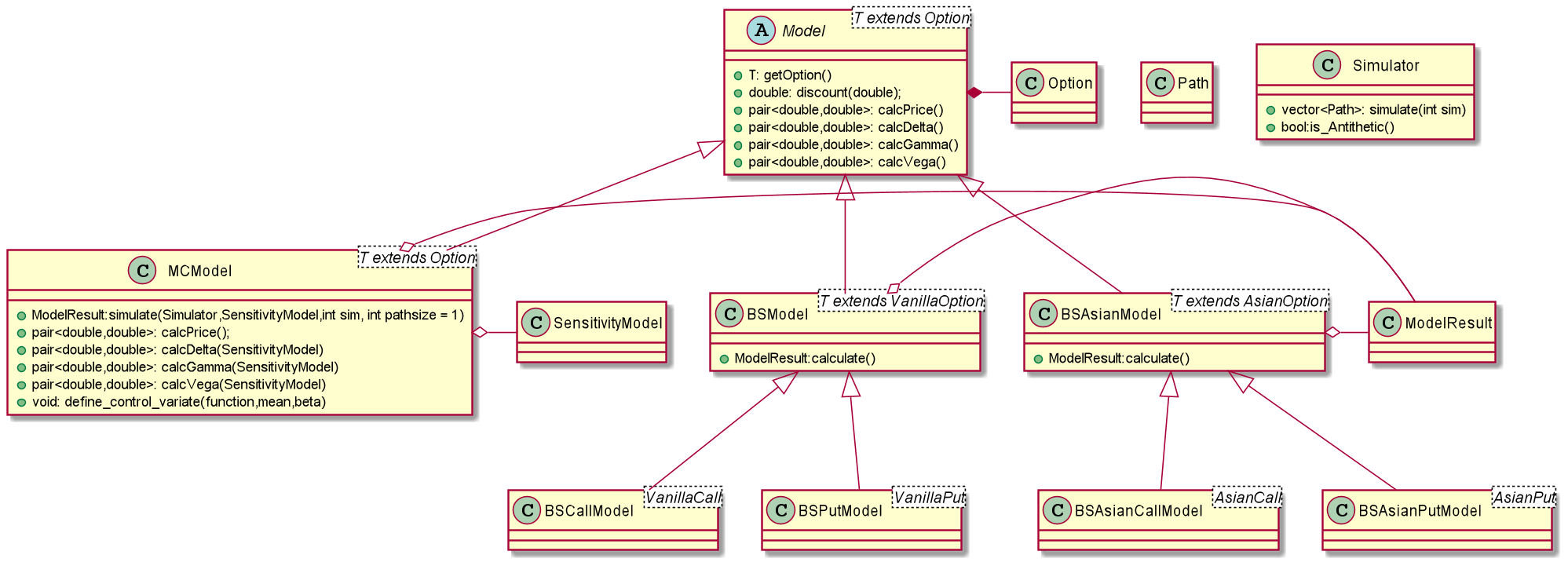
# Design

It’s worth the time to explain a bit about the design of our project so you can have an easier time understanding what the code does:

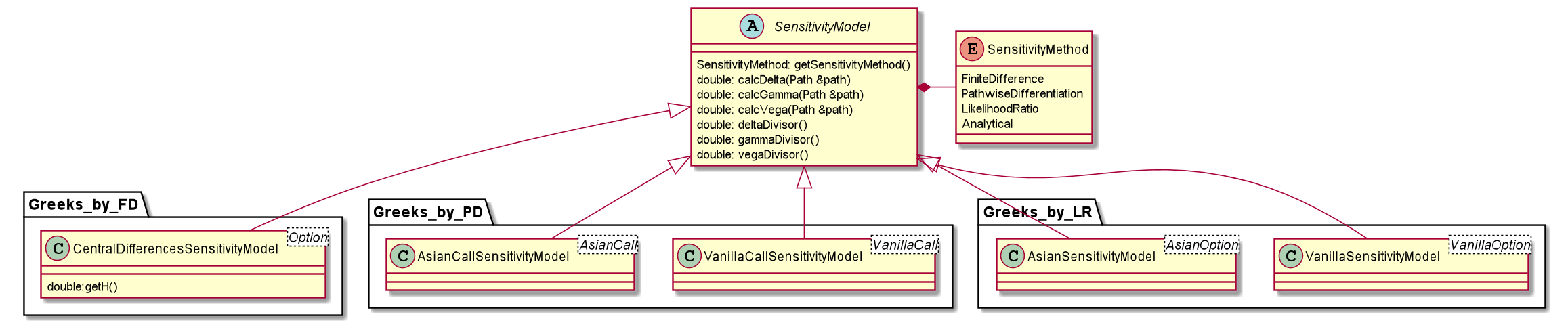
All the option classes implement the payoff function, and are organized as follows:



The main workhorse of the project is the Model class - from which the MCModel (short for Monte Carlo Model) and the Black Scholes models derive from, as follows.



In our Monte Carlo simulation, the sensitivities calculations are done on the SensitivityModel class, per Path, and are averaged inside MCModel class. The Central Differences method implementation works for any Option, but we also implemented methods that aim to reduce the variance of the sensitivities - the Pathwise differentiation and the Likelihood Ratio methods. Since those methods involve analytical closed form formulas of some degree that rely on the specific option in question, we provided only those for the (European) VanillaCall and the (Geometric Average) AsianCall



In this work we haven't implemented support for American style options - only European style options with Fixed strike.

# The Project Entry Point: main.cpp

The first place to look is the main() function inside the main.cpp file. It’s the main entry point for the application and contains all the high level logic of the application. You can easily change the option pricing parameters:

**double** s0 = 100.0;  
**double** strike = 100;  
**double** sigma = 0.4;  
**double** r = 0.05;

Our project assumes a constant interest rate of 5%, and constant volatility of 40%. The options (both the European and the Asian) have the same strike: 100, and the underlying is considered to have the same price, but the values themselves can be played upon.

An Option object needs to be declared before it can be used:

VanillaCall vanillaCall(strike, 1.0);

The Asian options require an additional parameter, which is the number of prices used in the average for the payoff function:

**double** asian\_dim = 100; *//Number of prices considered in the average*AsianCall asianCall(strike, 1.0, asian\_dim);

After that, we need to create a Model object. Two analytical models are provided:

BSCallModel bsModel(vanillaCall, s0, sigma, r);

And

BSAsianCallModel bsAsianModel(asianCall, s0, sigma, r, asian\_dim);

A more generic, Monte Carlo model, which can price any option (under the limitations and assumptions of our project), can also be created as follows:

MCModel<VanillaCall> mcModel(vanillaCall, s0, sigma, r, 0.005, ***Explicit***);

MCModel<AsianCall> asianMcModel(asianCall, s0, sigma, r, 0.005, ***ExplicitGeometricAverage***);

The MCModel objects above are instantiated with the usual parameters, plus a parameter that tells the model what SDE solver to use to simulate the path: the Explicit means we use the geometric brownian motion explicit solution to generate a price path, and the ExplicitGeometricAverage means that we use a explicit solution to sample the Geometric average directly without having to sample prices. This can be done because the Geometric average follows a lognormal distribution, just like the prices, but with different mean and variance.

Sensitivity calculation (greeks) is performed by subclasses of the SennsitivityModel. An example follows below:

Greeks\_by\_FD::CentralDifferencesSensitivityModel<VanillaCall> vanilla\_call\_fd(mcModel,0.005);

You can run a simulation by using the following two lines of code

Simulator simulator(normal,**true**); *//True=Antithetic*

ModelResult mcModelResult = mcModel.simulate(simulator,vanilla\_call\_fd,i);

# Concluding remarks

We are happy for you interest in our project. We tried to experiment with new design and performed code refactoring a few times in order to improve the code maintainability and flexibility. We would like to hear further from you – any ideas, suggestions or questions are all very welcome. Please reach us out at our github project like: <http://github.com/jdcbranco/simulationmethods>