Quantlib Project

Malo Le Goff, Clément André, Alexandre Giraud, Minh Tri Truong

March 2022

Contents

1 Introduction

In Monte Carlo engines, repeated calls to the process methods may cause a performance hit; especially when the process is an instance of the Generalized-BlackScholesProcess class, whose methods in turn make expensive method calls to the contained term structures. We'll see in this report how to increase the performance of the engine at the expense of some accuracy. After that, we'll apply the same modifications to an engine with a path-dependent payoff used for pricing Asian options.

2 European Option

The initial code used a Black and Sholes process combined with a Monte Carlo engine (from the MCEuropeanEngine class) to price an European option. As said in the introduction, the methods of the GeneralizedBlackScholesProcess class call term structures, which is quite expensive in terms of computation. The project's idea was to replace these term structures by simpler constant values. To do so, we've proceeded in 2 steps :

- Writing a ConstantBlackSholesProcess class
- Overriding the pathGenerator method inherited from the MCVanillaEngine class

2.1 Writing the constantBlackSholesProcess class

We make our constantBlackScholesProcess class inherits from the StochasticProcess1D class. Now we'll have to define what are the attributes of this class and which method does it need to override.

As precised in the text of the project, we only needed 4 constant parameters that we defined as parameters of our new constantBlackSholesProcess class :

- The underlying value
- The risk free rate
- The volatility
- The dividends

Now that we defined the attributes of our class, we must define what methods we have to override. Note that the function used to generate the path in the Monte Carlo method is the evolve() function, itself calling the apply() function itself calling the expectation() and stdDeviation() functions as we see in the file stochasticProcess.cpp. But the implementation of the evolve() function doesn't have to change. The expectation() and stdDeviation don't have to change either since they simply call the diffusion() and drift() functions. So we're left with 4 functions from the stochasticProcess class to override :

- x0()
- drift()
- diffusion()
- apply()

To override these functions, we copied their implementation in the blackscholesprocess.cpp class and tweaked them so they don't use complex termStructures but only the constant parameters we defined as the attributes.

2.2 Modification of the European MC Engine class

Now we must take care of the Monte Carlo engine used. The first modification is to add a boolean attribute that can decide whether or not the MC engine is going to run with the TermStructures parameters or is gonna extract and use constant parameters. The setting of such parameter is done with the method withConstantParameters() based on the same models of the other methods of the MakeMCEuropeanEngine class (withSamples(), withSeed(), ...)

Now the main part : overriding the pathGenerator method. Following the hint, we've overriden the pathGenerator method so when the method withConstantParameter(true) is called on the engine, it triggers the MC engine with constant parameters, i.e. with an instance of the ConstantBlackScholesProcess instead of a classic BlackScholesProcess. To do this, we extracted values from the termStructures of the BlackSholes process at a specific sampling time and use them as constant values to build our ConstantBlackScholesProcess.

2.3 Results

At first, we had no difference in NPV computed between the 2 processes although the method with the constantBlackScholes process was about 5 times faster. So we've decided to play around with parameters that should play a role in the accuracy of the calculation : the number of samples and the number of time steps.

Here is the evolution of the NPV of the european option priced using a classic BlackScholes process with non-constant parameters. Note that the rows corresponds to the different values of the number of samples used while the columns are the different values of the number of steps :

Nb_samples\Nb_steps	1	10	100	1000
10	3.4611632	6.0313977	4.2004066	4.6900114
100	4.8191836	4.1325136	4.3201488	4.1333739
1000	4.1937652	4.2230632	4.1816590	4.1118851
10 000	4.1829192	4.1751640	4.1508623	4.1727826
100 000	4.1716673	4.1635568	4.1703565	4.1744857

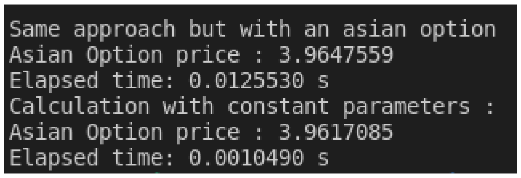
For the constantBlackScholes process :

Nb_samples\Nb_steps	1	10	100	1000
10	3.4611632	6.0313977	4.2004067	4.6900114
100	4.8191837	4.1325136	4.3201488	4.1333739
1000	4.1937652	4.2230632	4.1816590	4.1118851
10 000	4.1829192	4.1751640	4.1508623	4.1727826
100 000	4.1716673	4.1635568	4.1703566	4.1744857

So we see that, even if there are some slight differences for some combinations of number of samples and steps (colored in red), an European option's NPV is not very sensitive to the use of constant parameters instead of complex termStructures. But is it the case for an Asian option ?

3 Asian Option

The method is the same with few differences like replacing the MCVanillaEngine with the MCDiscreteAveragingAsianEngine class. The results though are quite different from the European option's results. It appears that the Asian option is much more sensitive to a call to constantParameters than the European option is. Indeed, we did not need to play around with the parameters to observe a difference in value :



```
Same approach but with an asian option
Asian Option price : 3.9647559
Elapsed time: 0.0125530 s
Calculation with constant parameters :
Asian Option price : 3.9617085
Elapsed time: 0.0010490 s
```

The method using constant parameters is about 10 times faster and the NPV is also quite different. It's logical that this kind of option is more sensitive because the option's pricing depends on the path generated. Indeed, as the constantBlackScholes process is used to generate the paths, the effects of using constant parameters (instead of using more precise term structures) on the pricing is more important than for the previous option whose payoff depends only on the value at maturity, so independent from the path