Fix oscillations in binomial tree engines

Contributors

Thanks to the following people who have contributed to this project :

- Corentine Nicolas
- Lola Spasojevic
- Paul Vissac
- Moad Yakoubi

Abstract

It has been noticed that the number of steps in a binomial tree affects the values of the option price and that the option value oscillates between two different values depending on whether the time step is even or odd. That is, Chung and Shackleton proposed a solution to this problem consisting on setting the option value at the penultimate nodes to the theoritical Black & Scholes value, considering that by definition there is no exercise of the option between the ultimate and the penultimate nodes of the binomial tree.

Implementation

The implemention consists on modifying the ***BinomialVanillaEngine*** in order to trigger the pricing scheme mentioned above. We choosed to add a boolean attribute to the class constructor named ***oscillations*** so that the pricing scheme is triggered if it is TRUE and is not triggered on the other case.

To do so, we modified the header file ***binomialengine.hpp***. We first roll back the option to the penultimate node (timeSteps – 1). We define also a variable of type Time named ***dt*** that represents the time to maturity of the option at this specific time step. We go then into a for loop with the goal of modifying the option value at each single node of the penultimate step. We retrieve first the underlying spot value at each node using ***underlying*** method of the lattice object before calling the ***BlackCalculator*** class for which we give as an input the option type (American put), the strike, the forward spot value, the standard deviation and finally the continuous discount factor.

Therefore, the last step of the process consist on attributing to each value of the option at the step just before maturity the maximum between analytical Black & Scholes value, and the payoff associated to exercising the option at the node, knowing that the option type is American and can be consequently exercised at every time step by the holder.

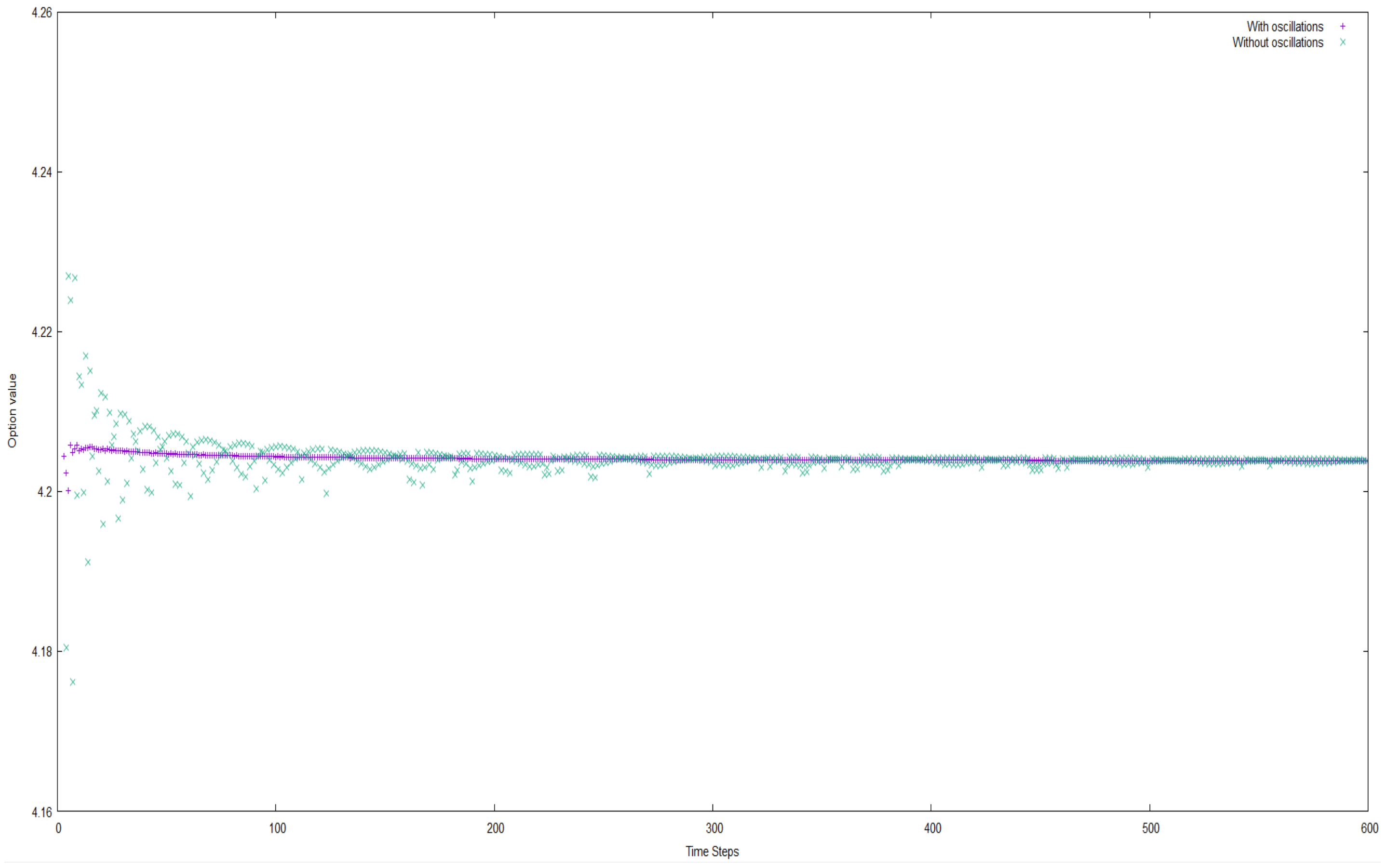
We call in the ***main.cpp*** file the implemented pricing scheme and the adjusted BinomialVanillaEngine class. We compare this behaviour too by setting the ***oscillations*** parameter to FALSE and therefore by not triggering the pricing mechanism. To be more specific, we instantiated a ***map*** variable that we call summary table in order to store the option Net Present Value at each time step and print it in the output shell. We also save the NPV values of the options in a 2 separate text files in order to plot their evolutions using ***gnuplot*** tool. Finally, we return as well the time duration of pricing the option at each step.

Results

The algorithm results shows a clear stabilization of the option value with the implemented scheme. We use for our simulation a range of ***timeSteps*** between 3 and 600 and the option value remains the same and converges to a stable value of 4.20382 after time step 591 for both odd and even time steps. For the second pricing scheme, which doesn't consist on modifying the option values at the penultimate step, we clearly notice the fluctuations of the option values for all simulated steps.

However, some criticism that could be given to our implementation is that the model and the scheme gives especially at the early time steps a lower present value than with a classical binomial tree.

Hereafter a ***gnuplot*** of the output simulation of the implementation for different time steps.



Conclusion

In this project, we have implemented a new pricing scheme and checked the behaviour of the binomial engine and its influence on the option prices. We have got familiar too with a new programming language that is ***C++*** and have been able to apply the different concepts seen in the financial theory of order to modify the ***BinomialVanillaEngine*** class.