QuantLib Project 3: Add Greeks to binomial tree engines.

Chemsi LAHLOU, Massinissa MENICHE,

Loïc LEMAIRE, Vincent LE GUERNEVÉ

Introduction:

For this project we want to improve the calculation of some Greeks (Delta and Gamma). Currently QuantLib library uses in the file binomialengine.hpp a first step tree with two nodes to calculate Delta and a second step tree with 3 nodes to calculate Gama.

So, we want to modify BinomialTree class to have at time $t = 0$ (no step) 3 nodes. Thanks to that we will improve the calculation of Delta and Gamma in BinomialVanillaEngine class, by doing this calculation with a no step tree.

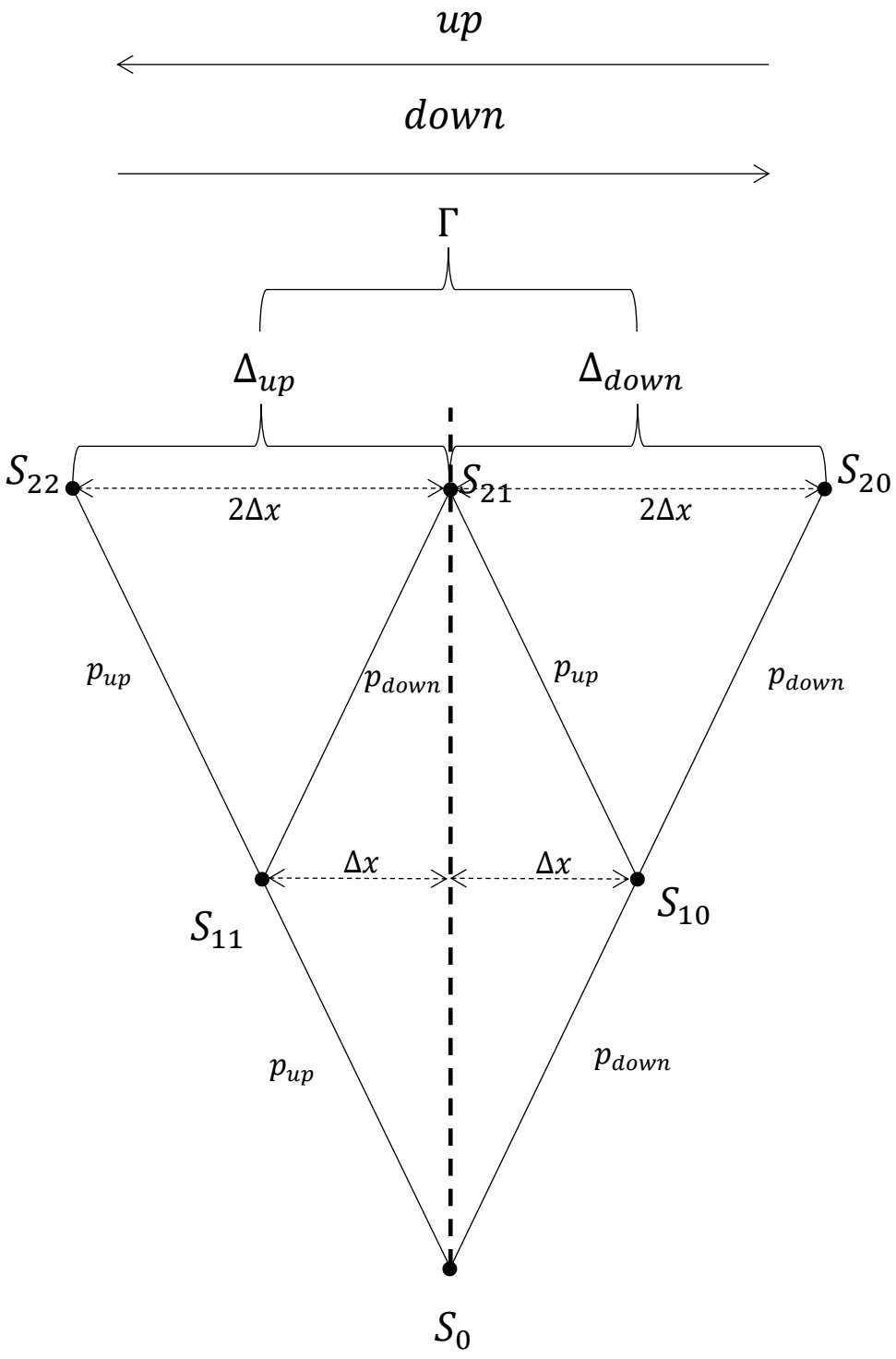
Moreover, we must consider 2 types of trees: equal jumps trees (case 1) and equal probabilities trees (case 2).

Case 1: Equal jumps trees

We have in the case of equal jumps trees a probability p_{up} to up and p_{down} to down for each step. Obviously $p_{up} + p_{down} = 1$.

So, if the value of the underlying at time $t = 0$ is S_0, after n_{step} steps with n_{up} up the value of the underlying is:

$$S_{n_{step},n_{up}} = S_0 e^{(2n_{up} - n_{step})\Delta x}$$



<u>Figure 1</u>: Equal jumps Tree.

We can notice that in an equal jumps tree $S_0 = S_{21}$.

So, we can add 2 steps in this tree and $2\Delta t$ at maturity and consider that we start at step 2.

So, we will have a tree with the same number of steps and the same maturity and 3 points at time $t = 0$.

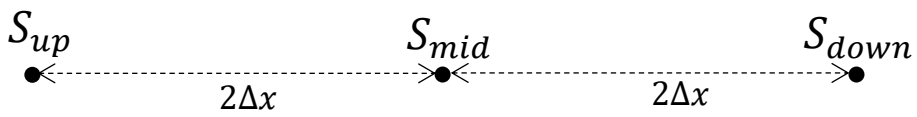


Figure 2: New tree at time $t = 0$.

After that, thanks to the method values of VanillaOption class we can calculate the payoff so the value of the option for each node. We can write them $P_{up}, P_{mid}, P_{down}$.

$$\Delta = \Delta_{up} = \frac{P_{up} - P_{mid}}{S_{up} - S_{mid}} = \frac{P_{up} - P_{mid}}{2\Delta x}$$

$$\Delta_{down} = \frac{P_{mid} - P_{down}}{S_{mid} - S_{down}} = \frac{P_{mid} - P_{down}}{2\Delta x}$$

$$\Gamma = \frac{\Delta_{up} - \Delta_{down}}{\frac{S_{up} - S_{down}}{2}} = \frac{\Delta_{up} - \Delta_{down}}{2\Delta x}$$

$$\Longrightarrow \begin{cases} \Delta = \dfrac{P_{up} - P_{mid}}{S_{up} - S_{mid}} \\ \Gamma = \dfrac{\Delta_{up} - \Delta_{down}}{\frac{S_{up} - S_{down}}{2}} \end{cases}$$

Case 2: Equal probabilities trees

We have for an equal probabilities tree after n_{step} steps and n_{up} up:

$$S_{n_{step}, n_{up}} = S_0 e^{(2n_{up} - n_{step})\_up} e^{n_{step} \times dps}, \qquad dps : \text{drift per step}$$

We can achieve the same method as in case 1. However, because of the drift per step coefficient $S_0 \neq S_{21}$.

So, we can create a new equal probabilities tree S', with the same drift per step coefficient and such that $S'_{21} = S_0$.

Now we are looking for S'_0.

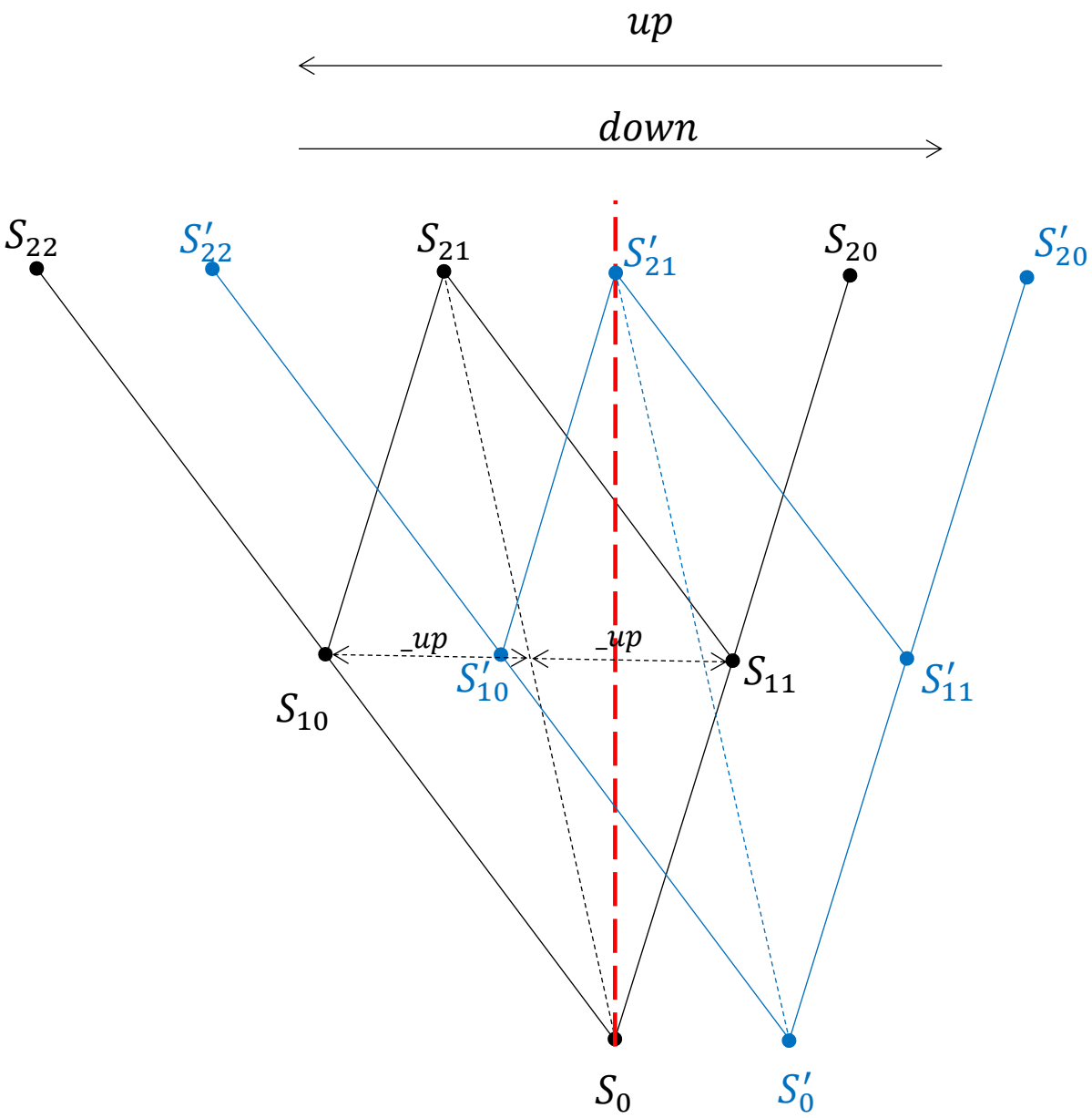


Figure 3: Equal probabilities Tree.

So $S'_{21} = S'_0 e^{(2\times 1-2)\_up} e^{2dps} = S'_0 e^{2dps} = S_0$ by construction of S'.

$\Rightarrow$ $\boxed{S'_0 = e^{-2dps} S_0}$

If we consider now S' with two more step and $2\Delta t$ longer maturity, we can proceed as in the case 1.

Main modification:

We also modify the file main.cpp to create a put option and calculate Delta and Gamma with the current method, using the actual Binomial engine and using Black and Scholes formula. We will use Black and Scholes result to compare the results between the actual binomial engine and the binomial engine with the new method.

So, we obtain:

```
Option type = Put
Maturity = January 17th, 2020
Underlying price = 334
Strike = 300
Risk-free interest rate = 0.100000 %
Dividend yield = 0.000000 %
Volatility = 20.000000 %


Delta calculated with the actual Binomial engine: -0.260452
Gamma calculated with the actual Binomial engine: 0.00486343

Black & Scholes Delta: -0.260504
Black & Scholes Gamma: 0.00486061
mmeniche@pc-sc-498:~/Bureau/Cpp/IMT2018/project3$
```

Annex: main.cpp

```
#include "binomialtree.hpp"
#include "binomialengine.hpp"
#include <ql/methods/lattices/tree.hpp>

// Added dependencies for Option creation
#include <ql/qldefines.hpp>
#ifdef BOOST_MSVC
#  include <ql/auto_link.hpp>
#endif
#include <ql/instruments/vanillaoption.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/utilities/dataformatters.hpp>

#include <ql/pricingengines/vanilla/analyticeuropeanengine.hpp>

#include <boost/timer.hpp>
#include <iomanip>
#include <iostream>

using namespace QuantLib;


int main() {

    try {
          //  Option caracteristics
          boost::timer timer;
        std::cout << std::endl;

        // set up dates
        Calendar calendar = TARGET();
        Date todaysDate(15, Jan, 2019);
        Date settlementDate(17, Jan, 2019);
        Settings::instance().evaluationDate() = todaysDate;

        // set up option characteristics
        Option::Type type(Option::Put);
        Real underlying = 334;
        Real strike = 300;
        Spread dividendYield = 0.00;
        Rate riskFreeRate = 0.001;
        Volatility volatility = 0.20;
        Date maturity(17, Jan, 2020);
        DayCounter dayCounter = Actual365Fixed();

        //Retrieve option characteristics
        std::cout << "Option type = "  << type << std::endl;
        std::cout << "Maturity = "        << maturity << std::endl;
        std::cout << "Underlying price = "        << underlying << std::endl;
        std::cout << "Strike = "                  << strike << std::endl;
        std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
                  << std::endl;
        std::cout << "Dividend yield = " << io::rate(dividendYield)
                  << std::endl;
        std::cout << "Volatility = " << io::volatility(volatility)
                  << std::endl;
        std::cout << std::endl;
        std::string method;
        std::cout << std::endl ;

        //Set up exercise
        boost::shared_ptr<Exercise> europeanExercise(new EuropeanExercise(maturity));


        // Market Data
        Handle<Quote> underlyingH(boost::shared_ptr<Quote>(new
        SimpleQuote(underlying)));

        //Bootstrap the yield/dividend/vol curves

        Handle<YieldTermStructure>
        flatTermStructure(boost::shared_ptr<YieldTermStructure>(new
```

```
            FlatForward(settlementDate, riskFreeRate, dayCounter)));
72
73          Handle<YieldTermStructure>
            flatDividendTS(boost::shared_ptr<YieldTermStructure>(new
            FlatForward(settlementDate, dividendYield, dayCounter)));
74
75          Handle<BlackVolTermStructure>
            flatVolTS(boost::shared_ptr<BlackVolTermStructure>(new
            BlackConstantVol(settlementDate, calendar, volatility,dayCounter)));
76
77
78          // Pay off construction
79          boost::shared_ptr<StrikedTypePayoff> payoff(new PlainVanillaPayoff(type,
            strike));
80
81          boost::shared_ptr<BlackScholesMertonProcess> bsmProcess(new
            BlackScholesMertonProcess(underlyingH, flatDividendTS,flatTermStructure,
            flatVolTS));
82
83          // Option Definition
84          VanillaOption europeanOption(payoff, europeanExercise);
85
86          // Pricing Engine
87          Size timeSteps = 801;
88          europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
            BinomialVanillaEngine_2<CoxRossRubinstein_2>(bsmProcess,timeSteps)));
89          //std::cout << europeanOption.NPV() << std::endl;
90
91          // Greeks calculated using the binomial tree
92          std::cout << "Delta calculated with the actual Binomial engine:
            "<<europeanOption.delta() << std::endl;
93          std::cout << "Gamma calculated with the actual Binomial engine:
            "<<europeanOption.gamma() << std::endl;
94
95          // Greeks calculated using the analytic B&S formula
96          method = "Black-Scholes";
97          europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
98                                          new AnalyticEuropeanEngine(bsmProcess)));
99          std::cout <<  std::endl;
100         std::cout << "Black & Scholes Delta: "<<europeanOption.delta() << std::endl;
101         std::cout << "Black & Scholes Gamma: "<<europeanOption.gamma() << std::endl;
102
103         return 0;
104
105     } catch (std::exception& e) {
106         std::cerr << e.what() << std::endl;
107         return 1;
108     } catch (...) {
109         std::cerr << "unknown error" << std::endl;
110         return 1;
111     }
112 }
113
114
```