Introduction to Quantlib

Projects

1 Use constant parameters in Monte Carlo engines

In Monte Carlo engines, repeated calls to the process methods may cause a performance hit; especially when the process is an instance of the `GeneralizedBlackScholesProcess` class, whose methods in turn make expensive method calls to the contained term structures.

The performance of the engine can be increased at the expense of some accuracy. Create a new class that models a Black-Scholes process with constant parameters (underlying value, risk-free rate, dividend yield, and volatility); then modify the `MCEuropeanEngine` class so that it still takes a generic Black-Scholes process and an additional boolean parameter. If the boolean is `false`, the engine runs as usual; if it is `true`, the engine extracts the constant parameters from the original process (based on the exercise date of the option; for instance, the constant risk-free rate should be the zero-rate of the full risk-free curve at the exercise date) and runs the Monte Carlo simulation with an instance of the constant process.

Compare the results (value, accuracy, elapsed time) obtained with and without constant parameters and discuss.

Hint: in the `MCEuropeanEngine` class, you will have to override the `pathGenerator` method inherited from the parent class `MCVanillaEngine`.

If you want to go the extra mile: apply the same modification to an engine with a path-dependent payoff, such as the `MCDiscreteArithmeticAPEngine` class (which is used to price an Asian option). How do the results change in this case?

2 Enable time-dependent steps for binomial tree engines

Currently, the `BinomialTree` class builds a tree based on constant parameters (risk-free rate, volatility etc.) extracted from the passed stochastic process. In the library, there is also an experimental `ExtendedBinomialTree` class which uses time-dependent parameters at each step of the tree; however, this currently causes a performance hit because the parameters are recalculated several times.

Analyze the `ExtendedBinomialTree` class and determine how to add data members so that common results are cached, thus increasing its performance. Measure the time to price a European option with different numbers of time steps before and after your modifications.

Also, use a Black-Scholes process with a non-constant risk-free rate and volatility and compare the results from the constant and time-dependent trees for a European option. Do you expect the result to change? What if the option is American instead?

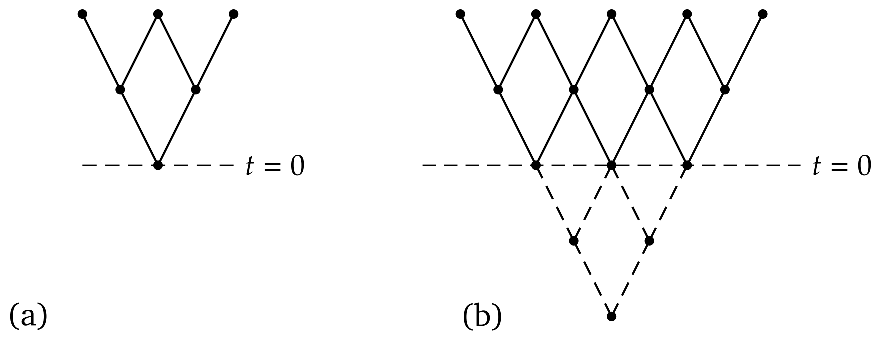

Figure 1: trees with one (a) and three (b) points at $t = 0$.

3 Add Greeks to binomial tree engines

The Delta and Gamma of an option can be calculated numerically as

$$\Delta(u) = \frac{P(u + \delta u) - P(u)}{\delta u}, \; \Gamma(u) = \frac{\Delta(u) - \Delta(u - \delta u)}{\delta u}$$

where $P(u)$ is the price of the option as a function of the underlying value u and δu is a small perturbation of such value; the current Delta and Gamma are $\Delta(u_0)$ and $\Gamma(u_0)$ where u_0 is the current value (that is, at $t = 0$) of the underlying.

On a binomial tree as currently implemented, there's a single point at $t = 0$ as shown in figure 1(a). This makes it impossible to calculate quickly and accurately the Delta and Gamma of an option; we can either re-run the calculation with a perturbed value of the underlying asset (which is slow, since we need three full calculations for Delta and Gamma) or we can use points at the first time step after $t = 0$ to obtain option values for a perturbed underlying (which is not accurate, since the option value at those points would be taken at $t \neq 0$; this is what the `BinomialVanillaEngine` class currently does).

The problem can be solved by having three points for $t = 0$ with the center point at u_0, as shown in figure 1(b), so that we can obtain the option values for $u + \delta u$ and $u - \delta u$ to use in the formulas for Delta and Gamma. Modify the `BinomialTree` class accordingly, and modify the `BinomialVanillaEngine` class so that it calculates Delta and Gamma correctly. Do you expect the extra points at each step to have a noticeable effect of the performance of the engine? Why? Verify your intuition by timing the code before and after your changes.

Note: the tree is built based on $\log u$ instead of u, so you'll have two different δu for the points at the left and right side. Take this into account during the calculations.

Hint: you don't need the part of the tree before $t = 0$ in figure 1(b).

time steps	value	time steps	value
360	12.7624691555	365	12.756572671
361	12.7559902568	366	12.7623429479
362	12.7624336546	367	12.7568998734
363	12.7562325066	368	12.7622865313
364	12.7623909426	369	12.7572141581

Table 1: Value of an American put option on a binomial Cox-Ross-Rubinstein tree for different numbers of time steps. $u = 100$, $k = 110$, $T = 1$, $r = 3\%$, $q = 0$, $\sigma = 20\%$.

4 Fix oscillations in binomial tree engines

As can be seen in table 1, options prices calculates on trees with different time steps oscillate between two values depending on whether the number of time steps is even or odd. Among others, Chung and Shackleton[1] argue that this can be helped by settings the option values at the penultimate nodes (i.e., those at the last time before maturity) to the analytic Black-Scholes values for a European option. This is acceptable because on the tree there's no exercise between the penultimate nodes and the last nodes (i.e., those at maturity).

Modify the `BinomialVanillaEngine` class so that it implements this pricing scheme (if you want, you can trigger this behavior by means of an additional boolean parameter passed to the engine constructor).

Hint: since you're going to overwrite the option values at the penultimate nodes, you don't need the nodes at maturity at all. This might simplify setting up the tree.

[1] *The Binomial Black-Scholes Model and the Greeks,* The Journal of Futures Markets, vol. 22 no. 2 (2002)