Computational Finance and its Object Oriented Implementation. **Exercise Handout 10**
Lecture: Prof. Dr. Christian Fries, Exercises: Dr. Andrea Mazzon, Tutorium: Lorenzo Berti
Wintersemester 2022/23

Exercise 1

The class

`LIBORMarketModelConstructionWithDynamicsAndMeasureSpecificationViaDirectVolatilityScaling`

in the package `com.andreamazzon.handout9` gives a solution to Exercise 2 of Handout 9 directly scaling the volatility defined in the method `createVolatilityStructure`.

In order to do that, the method `createLIBORMarketModel` has to pass two more arguments to `createVolatilityStructure`, i.e., `Dynamics dynamicsType` and `Measure measureType`. So there are now eight arguments of `createLIBORMarketModel` that are then passed to `createVolatilityStructure`. This is maybe not so nice.

Your goal for this exercise is therefore to restructure the class, making the two methods above non static anymore, defining some fields of the class (for example, the things that are used by both methods) and writing a suitable constructor.

Exercise 2

In the exercise repository you can find the class

`com.andreamazzon.handout10.CalibrationWithSwaptions`

which performs a calibration of the covariance structure of a LIBOR Market Model, based on the price of swaptions for different strikes produced by an object of type `LIBORMonteCarloSimulationFromLIBORModel`, which stands for the *true* LIBOR Market Model.

In particular, in this class we do the following:

- In the constructor, we give an object `trueLIBORMarketModel` of type `LIBORMonteCarloSimulationFromLIBORModel` which provides the tenure structure of the swaptions based on which we calibrate, the LIBOR Market Model simulation that we use to compute the target prices (see next point) and the simulation time discretization, the tenure structure, the forward curve and the discount curve of the LIBOR Market Model whose covariance we have to calibrate: in the end, we want to see how much the calibrated model differs from `trueLIBORMarketModel`.
- We construct an array of objects of type

  `net.finmath.montecarlo.interestrate.CalibrationProduct`,

  specifying that we want to calibrate based on Swaptions with tenure structure given by `trueLIBORMarketModel`, exercise date of our choice and strike which varies in the for loop identifying the entries of the array. In particular, we compute the initial value S_0 of the par swap rate and let the strike run from $\frac{1}{2}S_0$ to $2S_0$: the swaption of the i-th element of our array of `CalibrationProducts` has strike K_i. The target values of our calibration products (i.e., the prices we are pretending to see in the market) are the Monte-Carlo prices of such swaptions when the LIBOR Market Model simulation is represented by `trueLIBORMarketModel`, plus some random noise.
- We then calibrate based on the array we construct at the point above: in order to do that, we construct an object of type `LIBORMarketModelFromCovarianceModel`, whose `CalibrationProduct[]` field is given by such an array and whose simulation time discretization, tenure structure, forward curve and discount curve are taken from `trueLIBORMarketModel`.

In the `main` method of the test class `CalibrationTest` (you also find it in the repository, in the `test` part of our code) we then construct a LIBOR Market Model simulation `originalLiborMarketModelSimulation` with a given covariance structure: this is the object we give to the class constructed above, in order to let it play the role of `trueLIBORMarketModel`. We then perform the calibration calling the appropriate methods, and we compare the prices we get for the swaptions (for the same strikes you used to calibrate or others) with `originalLiborMarketModelSimulation` and with the calibrated model.

However, these prices differ a lot from each other: why? Try to change anything you want of one of the two classes, or both, to get smaller errors.