LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

quantLab

Computational Finance and its Object Oriented Implementation

(with Application to Interest-Rates and Hybrid Models)

Project:

Monte-Carlo Simulation and Calibration of a
Discrete Term Structure Model

and

Valuation of Interest Rate Derivatives

Christian P. Fries
email@christian-fries.de

Andrea Mazzon
mazzon@math.lmu.de

Lorenzo Berti
lorenzo.berti@campus.lmu.de

January 2nd, 2023

Version 0.7 (9.0.7)

Abstract

The following are the exercises of the project home-work of the lecture *Computational Finance and its Object Oriented Implementation (Winter 2022/2023)*, which is part of the final exam.

Support is provided by

Christian Fries, email@christian-fries.de

Andrea Mazzon, mazzon@math.lmu.de

Lorenzo Berti lorenzo.berti@campus.lmu.de

Contents

1 Introduction

The project consists of the improvements of a Monte-Carlo simulation of an Euler-scheme discretization of a Discrete Term Structure Model (LIBOR Market Model), the calibration of the model and the investigation of properties of interest rate products using this model.

The project consists of several (largely independent) parts. The group may choose to work on a selection (see below):

- improve the model allowing to use more general state-space transform - specifically improve the discretization of the displaced lognormal model,
- improve the model allowing to simulate under different measures - specifically under the T_k-forward measure,
- improve the model allowing a better interpolation of the forward rates (by simulating the short period bond),
- calibrate the displaced lognormal model to swaptions,
- investigate interest rate derivatives on the backward rate.

There are five tasks. A group of two students should solve at least one task. A group of three of four students should consider working on two tasks. The improvements of the model could be solved jointly with a single implementation.

These tasks do have a methodological aspect and a software design aspect. You could try to create a short and elegant solution.

1.1 Aim of the Project

The aim of the project is to perform a *collaborative development*, including testing and documentation.

Your project solution should consider:

- writing clean, concise, elegant and easy to understand code, avoid code duplication (DRY);
- write unit test (e.g. using JUnit);
- document the code (e.g. using JavaDoc).

It is not the primary aim to get an accurate solution.

Designing the code you may consider:

- your solution should try to follow the *open-closed-principle*: open for extentions, closed for modification. Example: consider extending the model from one asset to many assets.

- The type hierarchy of your models, products and test should follow the *Liskov substitution principle*. Example: products can be valued with different models; tests can be run with different models and products.

1.2 Evaluation of the Project Results

To successfully pass the review of the project we may ask for a short presentation of a part of the solution and ask you for answering a set of project related questions.

Note: The implementation part of the project can be solved very elegantly using object oriented implementation techniques requiring only few lines of new code. We encourage you to discuss your ideas with us in order to improve your solution.

1.3 Submission of the Solution

Each working group will receive its own password protected Git repository.

We will ask each group to present some parts of their solution and perform a short evaluation. The date of the presentation/evaluation will be announced shortly.

The solution has to be submitted to the repository before the presentation date.

1.4 Working with the Git Repository

For an introduction to Git we recommend `https://try.github.io/`

Remark

DON'T PANIC. WE WILL TRY TO ASSIST YOU!

THIS IS A FIRST DRAFT. CHECK FOR UPDATES!

2 Software Tools and Libraries

2.1 finmath lib

- You should implement your solution using the interfaces provided in the finmath lib. This is just to make your solutions interoperable. Please checkout the version at https://github.com/finmath/finmath-lib
- You may use anything else from finmath lib and if your solution is very short due to heavy re-use of existing code, then this is good.
- You may also re-write classes, if you see a good reason.

2.2 Apache commons-math

- The Apache commons-math library provides additional mathematical and numerical tools. Since commons-math is a dependency in fimath-lib, including finmath-lib will make commons-math available to you.

2.3 Visualization of Results

- We provide a small project which allows easily create plot in Java at https://github.com/finmath/finmath-lib-plot-extensions. The project provides wrapper code for popular plotting libraries (JFreeChart, JFreeSVG, JZY3D and iText).

3 Model Implementation

3.1 Task: Implement a more accurate Displaced Diffusion Model

Exercise 1 (Displaced Diffusion Model):

The displaced diffusion model specifies a log-normal dynamic for $L_i + d_i$.

As mentioned in the lecture, it is possibly an improvement to apply the Euler scheme discretization to $Y_i := \log(L_i + d_i)$.

For our generalized Euler scheme, this suggests the state space transformation

$$\begin{aligned} f_i(x) &:= \exp(x) - d_i \\ f_i^{-1}(x) &:= \log(L_i + d_i). \end{aligned}$$

The implementation `LIBORMarketModelFromCovarianceModel` is a bit limited as it only allows $f(x) = x$ and $f(x) = \exp(x)$.

1. Provide an improved implementation that allows to use the above *state space transformation* and perform an Euler-Scheme simulation.
2. Check your implementation by comparing it with the implementation using a plain Euler scheme (that is $f_i(x) = x$, state space `NORMAL`) and then using covariance model

   `LIBORMarketModelFromCovarianceModel`

   that uses the Blended Volatility Model ($\sigma(t, L) = \alpha L + (1 - \alpha)$).
3. Analyse the numerical error and (for example) the caplet implied (Black or Bachelier) volatility for different values of d_i (say 0, 0.03, etc.).

Hints and Remarks:

- You could just duplicate the class `LIBORMarketModelFromCovarianceModel` and add another `enum` and a vector of displacements d_i. That would do the first part. But you may try to do it better and create an interface providing the transformation. Note however that it also needs to provide the corresponding drift adjustment.
- If you like to solve this exercise by cleaning up the design and using an interface, you have to be a bit careful: the current implementation interprets the covariance model as the model of the transformed variable (Y), the drift calculation in the script is given for the back-transformed variable (L), but the Euler scheme expects the drift and the factor loadings to be for the variable Y (if you get stuck here, feel free to ask us).

- The Euler scheme provides the vector L at the current time step when calling `getDrift` and `getFactorLoadings` as the argument `realizationAtTimeIndex`.
- The exercise may be also viewed as an exercise on implementation design. It is possible to create a much cleaner design than the one that currently exists.

3.2 Task: Implement another Measure

Exercise 2 (Add the T_k Forward Measure):

The implementation `LIBORMarketModelFromCovarianceModel` allows to specify two different probability measures: SPOT and TERMINAL.

1. Improve the implementation (by duplicating the class and changing the code) and allow to create a simulation under the T_k-forward measure.

Hints and Remarks:

- The original version of `LIBORMarketModelFromCovarianceModel` contains some more advanced code on interpolating the numeraire that depends on the measure. You may ignore (and delete) this part of the code, just assuming the interpolation method LINEAR.
- Test the accuracy of the `Forward` and the `Caplet` under different measures.
- If you solve the next exercise, than this might provide you an easy way to solve this exercise.

Exercise 3 (Improve the Object Oriented Design to allow for Different Measures):

The implementation `LIBORMarketModelFromCovarianceModel` allows to specify two different probability measures: SPOT and TERMINAL.

The object oriented design of the class can be improved by providing the measure through objects implementing a suitable interface.

1. Design an interface that allows to provide the quantities that are relevant for implementing different measures (which quantities are relevant here?)
2. Create a new implementation of `LIBORMarketModel`, e.g. by duplicating `LIBORMarketModelFromCovarianceModel`, that uses your new interface to provide the measure.
3. Implement your interface for the SPOT and TERMINAL measure.
4. Implement your interface for the T_k-forward measure.
5. Test the accuracy of the `Forward` and the `Caplet` under different measures.

Hints and Remarks:

- The original version of `LIBORMarketModelFromCovarianceModel` contains some more advanced code on interpolating the numeraire that depends on the measure. You may ignore (and delete) this part of the code, just assuming the interpolation method LINEAR.
- The exercise may be also viewed as an exercise on implementation design. It is possible to create a much cleaner design than the one that currently exists.

3.3 Task: Implement a Better Interpolation

Exercise 4 (Simulate the Short Period Bond $P(T_{m(t)+1}; t)$ / Factional Forward Rate $L(T, T_{i+1}; T)$):

The specification of the short period bond $P(T_{m(t)+1}; t)$ is essential for the factional forward rate $L(T, T_{i+1}; T)$. It determines the interpolation of the forward rates.

The implementation `LIBORMarketModelFromCovarianceModel` uses a simple deterministic interpolation, where $L(T, T_{i+1}; T)$ is $\mathcal{F}_{T_i}$-measurable.

1. Derive how to *model* the short period bond or the fractional forward rate to be consistent with a given discrete forward rate model.
2. Implement an improvement of `LIBORMarketModelFromCovarianceModel` that allows to get the interpolated forward rate $L(T, T_{i+1}; T)$ and the interpolated numéraire.
3. Test your implementation by considering appropirate `Forward Rate Agreeements` and `Caplets`.

Hints and Remarks:

- This exercise is maybe more demanding in both aspects: deriving the mathematical model and creating a clean implementation.
- Creating a clean implementation based on the current implementation may be difficult because you might need access to the `BrownianMotion`. This is possible since the `process` is provided as an argument. In that case, the simulation time discretization needs to be finer for all forward rates. Alternatively, you may use a slightly more dirty implementation and use a new `BrownianMotion` just for "bridging" the interpolation.
- (Optional) There two models in the library that allow to simulate fractional/arbitrary forward rates (that is, there is more flexibility with respect to the tenor time discretization, but there is still the limitation to the simulation time discretization): the `HullWhiteModel` and the `LIBORMarketModelWithTenorRefinement`. You may consider using them as a benchmark.

4 Model Parameter Calibration: Properties of Model Parameters of a Displaced Lognormal Model

4.1 Task: Analyse the Properties of Different Model Parameters

Consider the following model parameters of a simple displaced lognormal discrete term structure model.

Tenor Time Discretization:

- Semi-Annual ($\Delta T = 0.5$) periods up to 40 years.

Initial Value:

- A flat forward rate curve fixed at $L_i(t_0) = 1\%$.

Covariance Structure:

- A two parameter, time-homogenous exponential decay in the volatility function: So the volatility of the i-th forward rate is $\sigma_i = \sigma \exp(-c(T_i))$
- A one parameter displacement (the same displacement for all forward rates).
- A one parameter exponential decay in the correlation $\rho_{i,j} = \exp(-\alpha|T_i - T_j|)$.
- (Optional - not so relevant): A factor reduction of the correlation to 2 factors.

Simulation Time Discretization:

- Semi-Annual ($\Delta t = 0.5$) periods up to 40 years.

Monte-Carlo Parameters:

- Different number of paths (5000, 10000, 100000 or more).

Exercise 5 (Investigate the "value" of different financial products as a function of different model parameters.):

Investigate the "value" of different financial products as a function of different model parameters.

- As "value" consider the value (price) or an implied volatility.
- Consider floaters, caplets or swaptions on the forward rate or caplets on a backward rate.
- Suggestion: How does a swaption depend on α and c?
- How do Swaptions with different strikes depend on the parameter d (this is a surface: the value is seen as a function of strike and displayment)?.
- Suggestion: How do a caplet on the forward rate and a caplet on the backward rate depend on the parameter c?
- Interpret the results.

Notes:

- Consider to convert the value in terms of an implied volatility.
- Example:
  - Plot the value (price) of a caplet or swaption as a function of the displacement parameter.
  - Plot the corresponding implied Bachelier or Black volatility (calculated from the price) as a function of the displacement parameter.

4.2 Task: Calibrate the Displaced Lognormal Model

Consider the model and model parametrization from the previous exercise.

Exercise 6 (Calibrate a Displaced Lognormal Discrete Term Structure Model to Swaptions):

Calibrate the models covariance structure (the volatility, the two exponential decay parameter, the displacement) to a given set of Swaption values.

You may like to investigate the different calibration strategies or using different numerical optimizers.

Use the following data for the initial value of the forward curve and the swaption product to which the model should be calibrated:

```
Forward rates:
   L(0.0,0.5;0.0)   = 0.0061    L(0.5,1.0;0.0)   = 0.0061
   L(1.0,1.5;0.0)   = 0.0067    L(1.5,2.0;0.0)   = 0.0073
   L(2.0,2.5;0.0)   = 0.0080    L(2.5,3.0;0.0)   = 0.0092
   L(3.0,3.5;0.0)   = 0.0111    L(3.5,4.0;0.0)   = 0.0136
   L(4.0,4.5;0.0)   = 0.0160    L(4.5,5.0;0.0)   = 0.0182
   L(5.0,5.5;0.0)   = 0.0202    L(5.5,6.0;0.0)   = 0.0217
   L(6.0,6.5;0.0)   = 0.0227    L(6.5,7.0;0.0)   = 0.0236
   L(7.0,7.5;0.0)   = 0.0246    L(7.5,8.0;0.0)   = 0.0252
   L(8.0,8.5;0.0)   = 0.0254    L(8.5,9.0;0.0)   = 0.0257
   L(9.0,9.5;0.0)   = 0.0268    L(9.5,10.0;0.0)   = 0.0282
   L(10.0,10.5;0.0)   = 0.0292    L(10.5,11.0;0.0)   = 0.0298
   L(11.0,11.5;0.0)   = 0.0300    L(11.5,12.0;0.0)   = 0.0299
   L(12.0,12.5;0.0)   = 0.0295    L(12.5,13.0;0.0)   = 0.0289
   L(13.0,13.5;0.0)   = 0.0282    L(13.5,14.0;0.0)   = 0.0274
   L(14.0,14.5;0.0)   = 0.0266    L(14.5,15.0;0.0)   = 0.0259
   L(15.0,15.5;0.0)   = 0.0252    L(15.5,16.0;0.0)   = 0.0247
   L(16.0,16.5;0.0)   = 0.0242    L(16.5,17.0;0.0)   = 0.0238
   L(17.0,17.5;0.0)   = 0.0235    L(17.5,18.0;0.0)   = 0.0233
   L(18.0,18.5;0.0)   = 0.0231    L(18.5,19.0;0.0)   = 0.0230
   L(19.0,19.5;0.0)   = 0.0229    L(19.5,20.0;0.0)   = 0.0228
   L(20.0,20.5;0.0)   = 0.0227    L(20.5,21.0;0.0)   = 0.0227
   L(21.0,21.5;0.0)   = 0.0226    L(21.5,22.0;0.0)   = 0.0226
   L(22.0,22.5;0.0)   = 0.0226    L(22.5,23.0;0.0)   = 0.0226
   L(23.0,23.5;0.0)   = 0.0226    L(23.5,24.0;0.0)   = 0.0226
   L(24.0,24.5;0.0)   = 0.0227    L(24.5,25.0;0.0)   = 0.0228
   L(25.0,25.5;0.0)   = 0.0228    L(25.5,26.0;0.0)   = 0.0230
   L(26.0,26.5;0.0)   = 0.0231    L(26.5,27.0;0.0)   = 0.0232
   L(27.0,27.5;0.0)   = 0.0234    L(27.5,28.0;0.0)   = 0.0235
   L(28.0,28.5;0.0)   = 0.0237    L(28.5,29.0;0.0)   = 0.0239
   L(29.0,29.5;0.0)   = 0.0242    L(29.5,30.0;0.0)   = 0.0244
   L(30.0,30.5;0.0)   = 0.0247    L(30.5,31.0;0.0)   = 0.0250
   L(31.0,31.5;0.0)   = 0.0252    L(31.5,32.0;0.0)   = 0.0256
   L(32.0,32.5;0.0)   = 0.0259    L(32.5,33.0;0.0)   = 0.0262
   L(33.0,33.5;0.0)   = 0.0265    L(33.5,34.0;0.0)   = 0.0268
   L(34.0,34.5;0.0)   = 0.0272    L(34.5,35.0;0.0)   = 0.0275
   L(35.0,35.5;0.0)   = 0.0278    L(35.5,36.0;0.0)   = 0.0281
   L(36.0,36.5;0.0)   = 0.0283    L(36.5,37.0;0.0)   = 0.0286
   L(37.0,37.5;0.0)   = 0.0288    L(37.5,38.0;0.0)   = 0.0291
   L(38.0,38.5;0.0)   = 0.0293    L(38.5,39.0;0.0)   = 0.0294
   L(39.0,39.5;0.0)   = 0.0296    L(39.5,40.0;0.0)   = 0.0297
   L(40.0,40.5;0.0)   = 0.0297    L(40.5,41.0;0.0)   = 0.0297
   L(41.0,41.5;0.0)   = 0.0297    L(41.5,42.0;0.0)   = 0.0297
   L(42.0,42.5;0.0)   = 0.0296    L(42.5,43.0;0.0)   = 0.0295
   L(43.0,43.5;0.0)   = 0.0294    L(43.5,44.0;0.0)   = 0.0293
   L(44.0,44.5;0.0)   = 0.0291    L(44.5,45.0;0.0)   = 0.0289
   L(45.0,45.5;0.0)   = 0.0287    L(45.5,46.0;0.0)   = 0.0285
   L(46.0,46.5;0.0)   = 0.0283    L(46.5,47.0;0.0)   = 0.0280
   L(47.0,47.5;0.0)   = 0.0278    L(47.5,48.0;0.0)   = 0.0275
   L(48.0,48.5;0.0)   = 0.0272    L(48.5,49.0;0.0)   = 0.0269
   L(49.0,49.5;0.0)   = 0.0267    L(49.5,50.0;0.0)   = 0.0264
   L(50.0,50.5;0.0)   = 0.0264

Swaption calibration products:
   {swapPeriodLength=0.5, swaprate=0.006314080206540701, swapStart=15.0, marketImpliedBlackVolatility=0.559, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.016314080206540703, swapStart=15.0, marketImpliedBlackVolatility=0.377, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.0213140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.335, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.023814080206540703, swapStart=15.0, marketImpliedBlackVolatility=0.32, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.0263140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.308, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.0288140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.298, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.0313140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.29, type=swaption}
   {swapPeriodLength=0.5, swaprate=0.0363140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.28, type=swaption}
```

```
{swapPeriodLength=0.5, swaprate=0.0463140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.27, type=swaption}
{swapPeriodLength=0.5, swaprate=0.021526003733826127, swapStart=12.0, marketImpliedBlackVolatility=0.385, type=swaption}
{swapPeriodLength=0.5, swaprate=0.023647577063380477, swapStart=13.0, marketImpliedBlackVolatility=0.351, type=swaption}
{swapPeriodLength=0.5, swaprate=0.025292554182271033, swapStart=14.0, marketImpliedBlackVolatility=0.325, type=swaption}
{swapPeriodLength=0.5, swaprate=0.0263140802065407, swapStart=15.0, marketImpliedBlackVolatility=0.308, type=swaption}
{swapPeriodLength=0.5, swaprate=0.026986537703910136, swapStart=17.0, marketImpliedBlackVolatility=0.288, type=swaption}
{swapPeriodLength=0.5, swaprate=0.02628914143433379, swapStart=20.0, marketImpliedBlackVolatility=0.279, type=swaption}
{swapPeriodLength=0.5, swaprate=0.023191375958219743, swapStart=25.0, marketImpliedBlackVolatility=0.29, type=swaption}
{swapPeriodLength=0.5, swaprate=0.023052085660527022, swapStart=30.0, marketImpliedBlackVolatility=0.272, type=swaption}
{swapPeriodLength=0.5, swaprate=0.024687354453749092, swapStart=35.0, marketImpliedBlackVolatility=0.235, type=swaption}
{swapPeriodLength=0.5, swaprate=0.02733884379259926, swapStart=40.0, marketImpliedBlackVolatility=0.192, type=swaption}
```

You do not need to copy that data by hand.

You find this data in the class `LectureProjectData` at

https://github.com/qntlb/computational-finance-lecture/blob/master/src/main/java/info/quantlab/computationfinance/lecture/montecarlo/interestrates/project/LectureProjectData.java

Notes:

- For the optimization problem you may use the Levenberg-Marquard optimizer. Given a function f, it tries to find the optimal parameter x^* such that $||f(x^*)|| = \min_x ||f(x)||$.

4.3 Task: Derive and Implement Analytic Valuation formula for a Swaption under Discrete Forward Rate Model

We consider a displaced LIBOR Market Model, i.e.,

$$\mathrm{d}L_i(t) \;=\; \mu_i(t)\,\mathrm{d}t \;+\; \sigma_i(t)\,(L_i(t)+d_i)\;\mathrm{d}W_i^{\mathbb{Q}^N}(t), \tag{1}$$

with initial conditions

$$L_i(0) \;=\; L_{i,0}.$$

Exercise 7 (Derive an Analytic Valuation formula for a Swaption under a Multi-Curve Displaced LIBOR Market Model):

1. Derive an analytic valuation formula for a Swaption under a Multi-Curve Displaced LIBOR Market Model by generalizing the derivation of the swaption approximation in a single-curve log-normal model.

   Remark: The derivation is very similar to that for a Swaption under a log-normal model. An additional step is the determination of a swap rate displacement from the forward rate displacement. Assuming that each forward rate has its own displacement parameter d_i, this can be done in the following way:

   Assuming a displaced local volatility function $\sigma_i(t)\,(L_i(t)+d_i)$ you consider a displaced log-normal swap rate

   $$\mathrm{d}S_{a,b}(t) \;=\; \mu_{a,b}(t)\,\mathrm{d}t \;+\; \sigma_{a,b}(t)\,(S_{a,b}(t)+d_{a,b})\;\mathrm{d}W_i^{\mathbb{Q}^N}(t). \tag{2}$$

   (a) Derive $d_{a,b}$ from d_i by considering the limit case $L_{i,0} = -d_i$ (plug this into the formula that give the swap rate as a function of the forward rates).

   (b) Derive $\sigma_{a,b}$ analogously to the log-normal model.

   Note: the model considered in the previous exercise assumed a single parameter d, this case is simpler - if you like, you may also just consider that case.

2. Optional: Assume that the forward rate and bonds are derived from different curves, i.e.,

   $$\begin{aligned} L_i &= \frac{P(T_i)-P(T_{i+1})}{(T_{i+1}-T_i)P(T_{i+1})} \\ S_{a,b} &= \frac{\sum_{j=1}^{m} L_i(T_{i+1}-T_i)P^\circ(T_{i+1})}{\sum_{j=1}^{m}(T_{i_{j+1}}-T_{i_j})P^\circ(T_{i_{j+1}})} \end{aligned}$$

   with

   $$P \;\neq\; P^\circ.$$

Notes:

- In the more general case with $P \neq P^\circ$, the definition of the swap rate is generalized:. It does not hold that the float leg can be written as $P(T_a) - P(T_b)$. We assume that there are two different zero bond curves. One is used to calculate the forward rates, the other is used for discounting (or to define the numeraire).
- The Monte-Carlo simulation of the discrete forward rate model discussed in the lecture supports the use of a separate forward curve and discount curve.
- The new local volatility function will lead to different covariances and weights. The move to a separate forward and discount curve will impact the weights.

Exercise 8 (Implement an Analytic Valuation formula for a Swaption under a Multi-Curve Displaced LIBOR Market Model):

Implement an analytic valuation formula for a Swaption under a Displaced LIBOR Market Model. That is: implement the formula from the previous exercise.

Notes:

- Consider re-using the existing code, e.g. the calculation of the integrated covariance matrix.
- The valuation is only admissible if the LMM carries a local volatility function of a specific type. You may check for the covariance model using `instanceof`.

Exercise 9 (Calibrate a Displaced Lognormal Discrete Term Structure Model to Swaptions):

Repeat the calibration exercise above using your analytic formula.

5 Product Valuation

5.1 Task: Exploring Products on the Backward Rate

Exercise 10 (Exploring Products on the Backward Rate):

Implement some interest rate products that use the interface

`TermStructureMonteCarloSimulationModel`

as a model and investigate the properties of the products using different versions (e.g. time-discretizations) of the model, for example

- A backward rate, given a certain time-discretization for the rolling bond (accrual account).
- A swap on the backward rate.
- A caplet on the backward rate.

Check that the swap on the backward rate give (apart from numerical errors) the same value as the swap on the corresponding forward rate, but the caplet on the backward rate is different from the caplet on the forward rate. Why is that? Explore the difference.

Remarks:

- When you value a product on the backward rate, the model should have a fine time discretization that comprises the time steps in the accrual account of the backward rate. You may choose a backward rate with a weekly accrual, to limit the memory requirements of your computer.
- Be careful if you use too few paths. The Monte-Carlo error may distract you.
- You might consider designing your products such that you may reuse some code.
- When you analyse the product, you may need a model with a much finer time discretization (daily, weekly). This may be issue as it increases the memory requirement. A trick may be: run experiments with a small number of paths (1000) using may different seed values for the random number generator (10 or 20), then calculate the average of all the values.

References

[1] FINMATH.NET: finmath lib - source code repository.
http://finmath.net/finmath-lib/.

[2] FINMATH.NET: finmath lib - api documentation.
http://finmath.net/finmath-lib/apidocs.

[3] FINMATH.NET: finmath lib plot extensions.
https://github.com/finmath/finmath-lib-plot-extensions.

20 pages. 0 figures. 0 tables.