ECMI Modelling Week, July 17–24, 2016, Sofia, Bulgaria

Group 10

Volatility stripping in fixed income derivatives

Adriana Cola

Università degli studi di Milano, Italy
adrianacola89@gmail.com

Mirko Drazic

University of Novi Sad
mirkodrazic93@gmail.com

Slavi Georgiev

Angel Kanchev University of Ruse
georgiev.slavi.94@gmail.com

Joachim Luebbers

Technical University of Dresden
Joachimluebbers@web.de

Olavur Mortensen

Technical University of Denmark
olavurmortensen@gmail.com

Riyaz Mouhamad

Sup-Galilée - University of Paris 13
m.riyaz.22@gmail.com

Instructor: **Gerardo Oleaga**

Complutense University of Madrid
goleaga@ucm.es

Abstract. The aim of our work is to estimate the (Black) volatilities of caplets through the flat volatilities of caps quoted in the market. This process is called *caplet stripping* and involves non-linear numerical optimization techniques and/or multidimensional root finding methods. Some smoothness criteria are implemented to regularize the ill-posed optimization problem.

10.1 Preface

This report sums up the concepts learned, opinions and results of the group work 10 at the ECMI-Modelling Week 2016 in Sofia, Bulgaria. Guided by Gerardo Oleaga we had the opportunity to improve our knowledge about finance instruments in a very professional environment. Grateful about the chance to been part of the Modelling Week we want to thank everybody who made it possible and especially Gerardo Oleaga for his patience and inspiring guidance.

10.2 Introduction

This report can mainly be divided in four main parts. First there will be given a short explanation of the financial definitions and the Bootstrapping method. The second Part deals with the interpolation, filling in the data and the optimization of the algorithm. The third part provides a deeper look on the code and the language Julia itself. The discussion and outlook concludes the report.

10.3 Financial Instruments

In order to understand the pricing of a Cap/Floor basic knowledge about these finance instruments are needed.

A Cap/Floor is a Call/Put-Option where its underlying is an interest rate. This means that the buyer receives payments at the end of each period in which the interest rate exceeds/deceeds the agreed strike price. An interest rate cap/floor is designed to provide insurance against the rate of interest on the floating-rate note rising above a certain level. Figure 10.1 shows an example of a cap/floor.

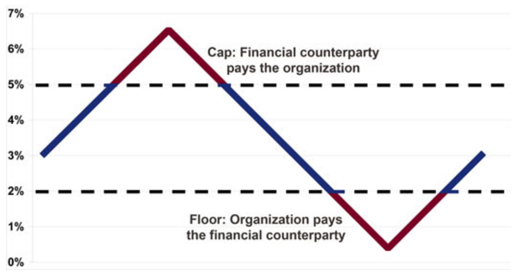

Figure 10.1: Example of a cap with the strike 5% and a floor with strike 2%

A Cap is simply the sum of its constituent caplets. For example if we have a Cap on the Libor rate with Strike 5% which has a 2 years maturity that pays at the end of each six months period. Then, the Cap is composed by four Caplets.

10.3.1 Pricing Caps and Floors

The following part lean on the paper, *Eight ways to strip your caplets: An introduction to caplet stripping.* We just explain the way of pricing a cap through its caplets. The case of a Floor and its floorlets is similar. The cap price is simply the sum of the caplet prices with a common strike K and it's fair value with covering the period T_α to T_β with nominally payments at $T_{\alpha+1}, T_{\alpha+1}, ..., T_{\beta-1}$ and T_β:

$$\begin{aligned} V_{cap}(t, K, T_\alpha, T_\beta) &= \sum_{i=\alpha}^{\beta-1} V_{caplet}(t, K, T_i) \\ &= \sum_{i=\alpha}^{\beta-1} P(t, T_{i+1}^p)\tau_i Black(F_i(t), K, T_i, \sigma(K, T_i), \chi) \end{aligned}$$

To fully understand this equation the pricing of caplets needs to be understood. Therefore, the following notation is used:

- the value $L(T_i, T_{i+1}) \equiv L_i$ with the Libor rate from the period T_i to T_{i+1} fixing at T_i^f (normaly two days before T_i)
- the forward rate $F(t, T_i, T_{i+1}) \equiv F_i(t)$ for this Libor with $t \leq T_i^f$.

- the year fraction τ_i for the period with respect to the Day Count Convention
- χ is either $+1$ for caplets or -1 for floorlets.

The payoff of a caplet is structured like a call option on a Libor rate:

$$\text{payoff}_{caplet} = \tau_i(\chi[L_i - K])^+$$

The payment for a standard caplet starting at T_i is done at T_{i+1}.
The Libor rate fixes at T_i, so:

$$V_{caplet}(T_i, K, T_i) = P(T_i, T_{i+1}^p)\tau_i(\chi[L_i - K])^+$$

- $P(T_i, T_{i+1})$ is the discount factor from the payment time.

Now the fair value is:

$$V_{caplet}(t, K, T_i) = P(t, T_{i+1})\tau_i \mathbb{E}_t^{\mathbb{T}_{i+1}}[(\chi[L_i - K])^+]$$

- $\mathbb{E}_t^{\mathbb{T}_{i+1}}[(\chi[L_i - K])^+]$ is the expectation in the T_{i+1} forward measure

Therefore

$$F_i(t) = \mathbb{E}_t^{\mathbb{T}_{i+1}}[L_i]$$

The price of caplets is given by the Black formula through the implied volatilities $\sigma(K, T_i)$ (for a particular strike and expiry):

$$V_{caplet}(t, K, T_i) = P(t, T_{i+1})\tau_i Black(F_i(t), K, T_i - t, \sigma(K, T_i), \chi) \tag{10.1}$$

with

$$\text{Black}(F, K, T, \sigma, \chi) = \chi(F\phi(\chi_{d1}) - K\phi(\chi_{d2}))$$

and

$$d_1 = \frac{\log(\frac{F}{K}) + \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}; \quad d_2 = d_1 - \sigma\sqrt{T}$$

The value of a cap is now obtained as the sum over the price of the caplets, each one priced through its corresponding Black volatility.

Under Black's model, financial derivatives prices are determined by the volatility of the underlying variable. In the fixed income market, the volatility of the forward interest rates are basic inputs in the pricing formulas but unfortunately, this parameters are not directly observable in the market, and they must be inferred from quoted prices of complex contracts. Therefore we need a procedure to obtain this parameters from quoted market prices.

Nowadays negative forward rates/strikes are appearing for some currencies and Black's model cannot handle them because the logarithm term of the expression $(\frac{F}{K})$ is not defined for negative values. Therefore in that cases a remodeling of the pricing formula is needed.

10.3.2 Cap "flat" Volatility

For the market observed price of a cap with strike K the implied *flat* volatility is the single value given to all the constituent caplet's volatilities which recovers the cap price, that is:

$$V_{cap}(t, K, T_\alpha, T_\beta) = \sum_{i=\alpha}^{\beta-1} P(t, T_{i+1}^p)\tau_i Black(F_i(t), K, T_i, \sigma(K, T_i), \chi)$$

so the value $\sigma(K, T_\alpha, T_\beta)$ solves the equation above. The price of a one year cap will generally be different from a two year cap because of the difference in flat volatility values, even for the same strike. Hence the cap flat volatility is an alternative quoting convention to price.

10.4 Processing and cleansing data

For the purpose of our work, we will use the data for cap volatilities provided in [1]. Those are real market data, so we have to fill and further process the raw values to make fit the numerical procedure.

Firstly, we have data where the outliers have been removed - simply because they violate the data consistency. Therefore, holes appear in the flat volatility matrix provided. In order to fix it, me make several "Laplacian" iterations to fill in the holes. The main idea is to obtain a data set where the holes are filled with a weighted mean of the surrounding data. In the first iteration, holes are filled with zero values. After five or six turns, the matrix is stabilized (data outside the holes are not modified). We may adjust the procedure giving more weight in the direction of strikes than cap maturities or vice versa. We needed to be careful when processing values on indices that are either borderline or when their neighbours are not equally distanced from the current value. Any small deviation could ruin the root finding or optimisation algorithm.

For the sake of completeness, for our data we needed about five iterations to get satisfying results. This value is derived empirically derived, checking that further changes of the data after the fifth or sixth iteration are negligible small, so the result fits our needs.

10.5 Bootstrapping

So, as we mentioned before, the market quotes a set of cap prices, ordered by strike and maturity, through their flat volatilities. We aim to find the implied volatilities for each caplet in a cap.

We implemented firstly the Bootstrapping method, because it is very fast and simple. The method allows to find a rapid solution, provided that it exists. Moreover, cap prices are recovered exactly, so there is no arbitrage. The procedure is as follows: First we order the caps by the size of maturity, starting from the shortest one. Then we find the price differences for the caps – these are the prices of some sub-caps, formed by two or more caplets. The first partition is the same as the shortest cap. The partitions obtained consist of caplets, so they can be priced as caps, using the aforementioned formulae.

We have to match the price of the cap partitions with caplet volatilities – one side gives us the market price and the other gives us the price in terms of volatilities (through Black formula). We can then apply a 1-D root finding method for the price difference in order to obtain the *implied* volatility of the caplets.

Our "implied" volatility (which is assigned to the whole partition) appear to be a flat volatility of these set of caplets.

The procedure can be improved making a linear interpolation of the volatility in order to get different values for each caplet, linked by a straight line. We can apply again a root finding algorithm for the slope coefficient and derive values, which are consistent (ie. continuous) with the corresponding volatility of the previous caplet.

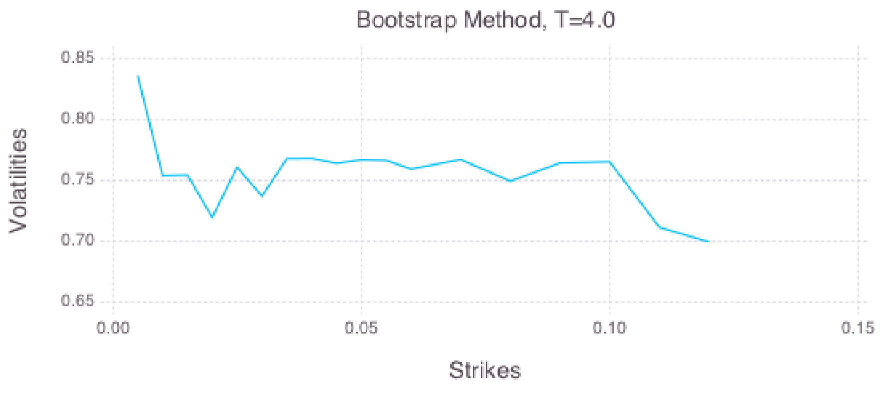

Figure 10.2: Volatility smile

Now we will look at the behaviour of volatilities depending on strikes and maturities. On Figure 10.2 we have the interpolated volatility curve for different strikes, observing the so called «volatility smile». We have fixed the maturity to 4.0 years, which is an intermediate value. As we will see later, for very small expiries we could get a "sad" smile.

On Figure 10.3 we observe the behavior of the volatility for a fixed strike and subsequent values of maturities, the term structure. We see how it looks like for different strikes from the volatility surface, discussed in Section 10.6.

Before continuing, let us mention one significant difference between the two graphs. While the term structure is somewhat smooth, the volatility smile looks rather implausible. The explanation is simple – there is no coupling across strike values to impose any smoothness in that direction.

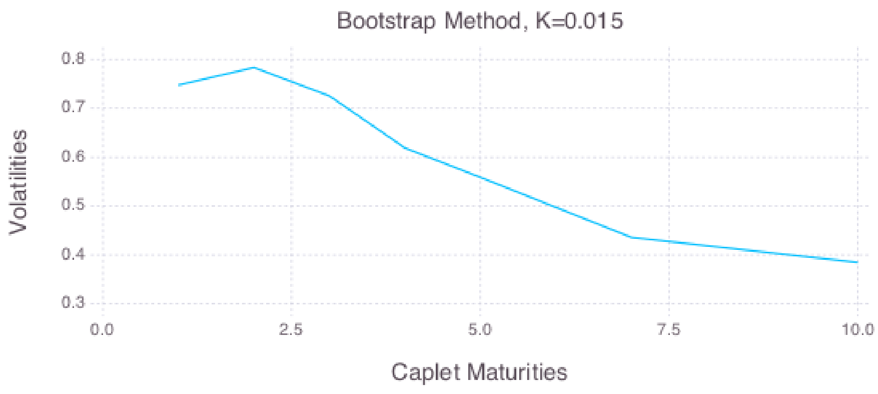

Figure 10.3: Term structure

10.6 Interpolation

On Figure 10.4 we can see how volatility depends on independent values of strikes and maturities. Our goal is almost achieved, but there is a problem – the surface is too rough and peaky.

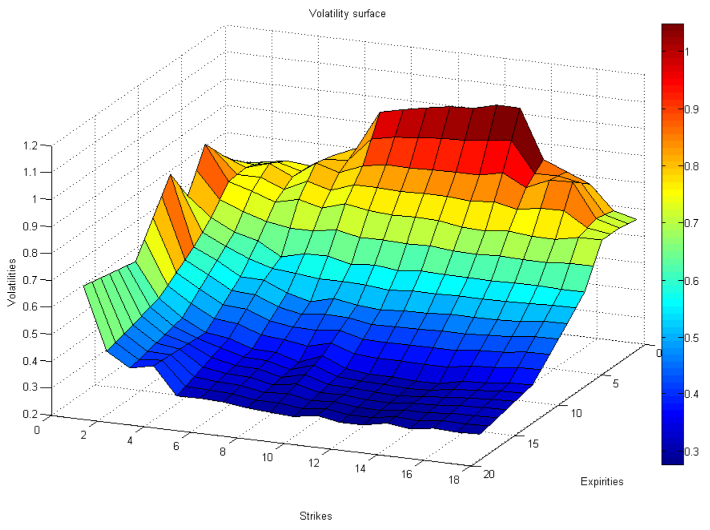

Figure 10.4: Volatility surface

In order to make it smoother, we first tried to do a polynomial interpolation between market data. If we apply a linear interpolation, the result will be the same as the previous one, so this is a matter of a little interest. Quadratic or cubic interpolations would suit better, but the polynomial interpolations are not appropriate in our case, because higher order polynomials produce big oscillations. The best results were obtained through cubic spline interpolation, as shown in the figure.

The smoothed plot (10.5) uses more knots than the market values, which are

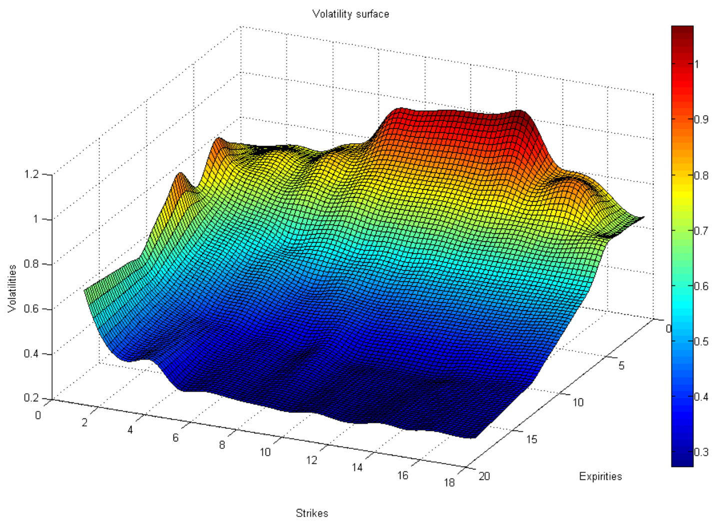

Figure 10.5: Volatility surface – smoothed

equally distributed. The latter plot is much smoother and both surfaces have a good property – they fit exactly the market data. They do not induce arbitrage but we loose control of the data parameters. In the following, we will apply methods that could adjust accuracy and smoothness in a more suitable way.

10.7 Optimization - Penalty approach

First of all, we consider a function $F : R^m \mapsto R^n$ which takes a vector $\vec{\sigma}$ of m implied volatilities and returns a vector of n cap values. Hence, $\vec{\sigma} \mapsto F(\sigma) = \vec{p}$. In our case the values are $m = 19$ and $n = 7$, so the problem is ill-posed and does not have a unique solution. To deal with this, we are imposing an additional penalty term G, which represents a scalar measure of smoothness.

Therefore, our problem consists of minimising

$$||F(\vec{\sigma}) - \vec{p}||^2 + \lambda G(\vec{\sigma})$$

for some given λ to be fixed. Term $G(\vec{\sigma})$ is of the form $\vec{\sigma}^T P \vec{\sigma}$ where the P is some penalty matrix such that the whole term $\vec{\sigma}^T P \vec{\sigma}$ is close to zero only if $\vec{\sigma}$ is perfectly smooth, and increases the less smooth $\vec{\sigma}$ is. The second derivative encodes how the slope of the function changes, so actually it provides a measure of the curvature of the graph. The curvature will be bigger for bigger second derivatives. This is why we select the penalty matrix as $P = D^T D$ where D is the 2^{nd} order discrete differentiation operator such that $D\vec{\sigma}$ is the vector of central difference estimates

$$\vec{\sigma}(t-1) - 2\vec{\sigma}(t) + \vec{\sigma}(t+1)$$

of $\frac{d^2\vec{\sigma}}{d^2t}$ in 17 middle dots. Hence, D is a tridiagonal matrix

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

So with the smoothness criterion, the penalty term becomes:

$$\lambda \cdot G(\vec{\sigma}) = \lambda \vec{\sigma}^T P \vec{\sigma} = \lambda \vec{\sigma}^T D^T D \vec{\sigma} = \lambda \cdot \|D\vec{\sigma}\|^2$$

The weight of the smoothness of the volatility surface in the optimization procedure will depend on the value of λ. For bigger λ, the penalty term becomes more significant than the first term, so we will obtain smoother surfaces, but then the first term will possibly be far from zero, so we will have some errors in cap prices. This implies arbitrage opportunities. As λ decreases towards 0, we will obtain accurate cap prices, but we should expect our surface to be less smooth.

In figure 10.6, we observe two pictures obtained by using the penalty method. The upper one is called "volatility smile", and it represents the volatility values depending on strikes, while the lower one, called "term structure volatility", represents the volatility depending on caplet maturities. Also we need to point out that the upper plot is taken from the surface for caplet expiry $T = 4$ and that the lower plot corresponds to the strike value $K = 1.5\%$

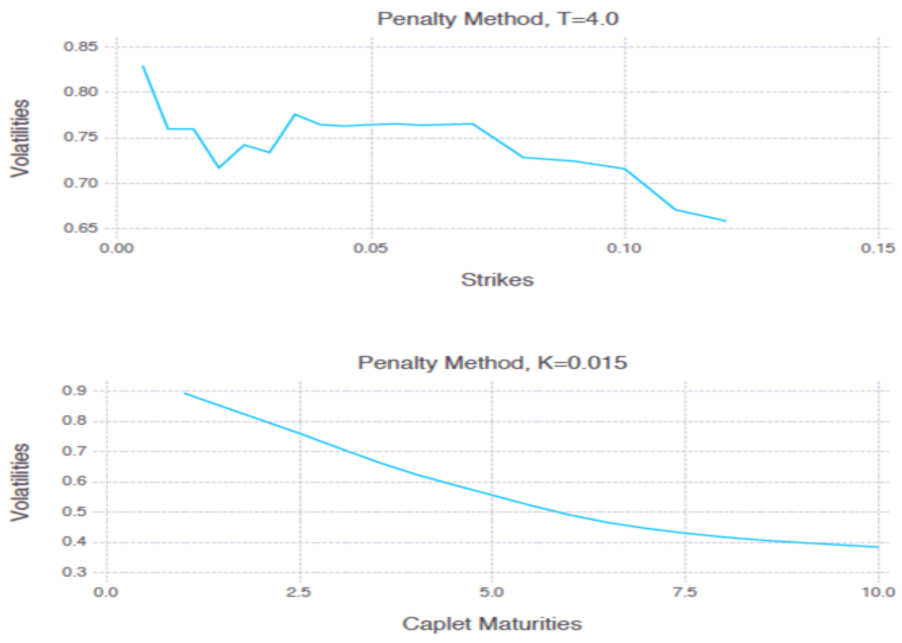

Figure 10.6: Volatility smile and Term structure volatility

In figure 10.7, we show two surfaces, one obtained from the bootstrapping method and one from the penalty method. It is obvious that the results from the penalty method are smoother, especially if we pay attention to the term structure, for the smallest strike value 0.5%.

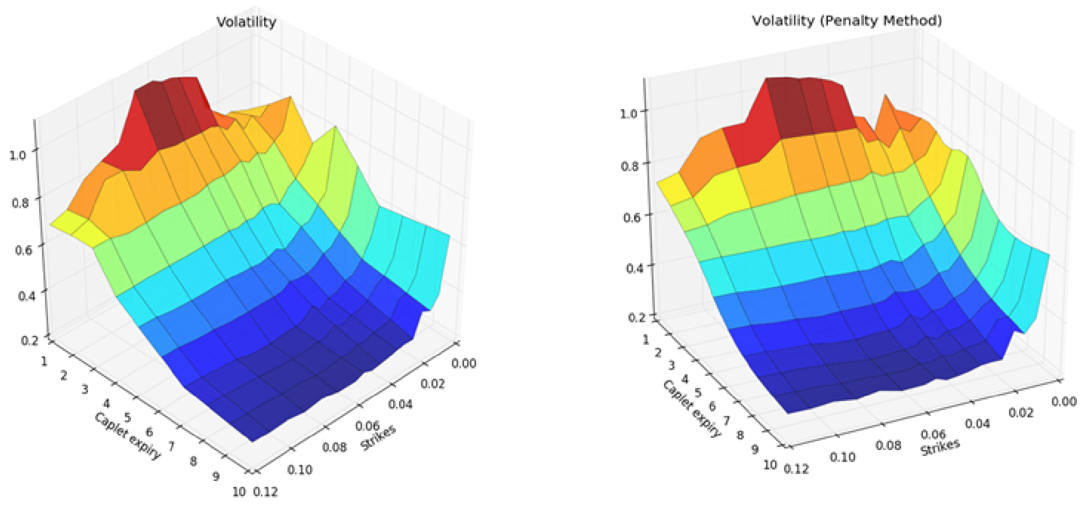

Figure 10.7: Surfaces from bootstrapping method and penalty method

Finally, in figure 10.8 we represent three graphs which summarize our work.

The top graph in figure 10.8 corresponds to the bootstrapping method. The results are not smooth, but we have obtained completely exact prices. The middle graph in figure 10.8 is the penalty method, corresponding to the value of lambda that is too high. In this case the smoothness is achieved, but we have got too high price error. The bottom graph in figure 10.8 is the optimal, we obtained a smooth curve, combined with a low pricing error.

10.8 Implementation

This section describes a few details of the implementation of the models.

10.8.1 Julia

Julia is a fairly new programming language that has adopted features from many of the most popular programming languages today. Some of these features are: dynamic typing and high level of abstraction like Python, performance (almost) like C, multiple dispatch methods, metaprogramming like Lisp, and more.

The primary downside with Julia at the moment is the lack of well tested specialized libraries. In this project, we only needed a plotting library, an optimization library and a simple statistics package, all of which Julia does provide (PyPlot, Optim and Distributions, respectively); therefore, Julia fits well to

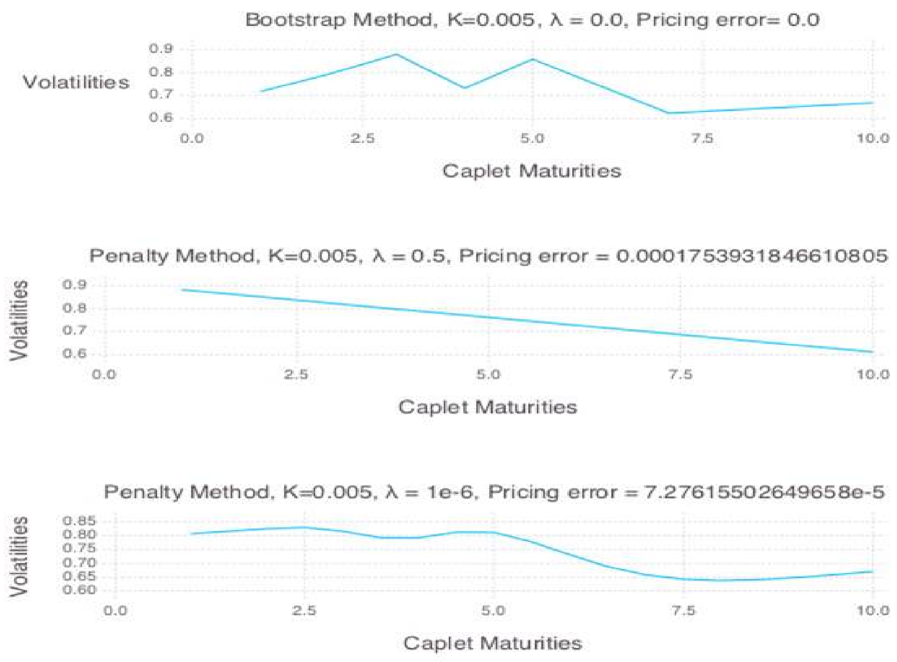

Figure 10.8: bootstrapping method, penalty method for $\lambda = 0.5$, penalty method for $\lambda = 10^{-6}$

this project. Furthermore, Julia allows calling C functions directly, and calling Python functions requires little work.

10.8.2 User-defined types

Julia allows user-defined types, and we took advantage of that in our project. To make caps and caplets easier to deal with, we define some types. The caplet is defined as

```
type Caplet
  strike :: Float64
  S :: Float64
  T :: Float64
end
```

and the cap is defined as

```
type Cap
  strike :: Float64
  dates :: Vector{Float64} # size n+1
end
```

As we shall see later on, this allows us to work with caps and caplets in a much more intuitive manner, for example by passing them directly to functions that return the price by dispatching in terms of the corresponding type.

10.8.3 Multiple dispatch

Julia identifies a function not only by its name, but by the input types too. This way, we can define a single function with several methods, where each method accepts different arguments. Each method can handle the arguments differently, but can also behave similarly.

We defined one pricing function for the caplets,

```
price(t::Float64, disc::Float64, c::Caplet,
      forward::Float64, vol::Float64)
```

and another pricing function for the caps,

```
price(t::Float64, disc::Vector{Float64}, cap::Cap,
      forwards::Vector{Float64}, vol::Vector{Float64})
```

The difference between these functions is that the former accepts a caplet, and the latter accepts a cap, and the forward rate and flat volatilities are now vectors of 64 bit floats. Julia will note the inputs passed and call the correct method automatically.

Using multiple dispatch, we do not need to think about which pricing function to call, only what the correct inputs are. Furthermore, we avoid exceedingly long function names such as `pricing_caplets`, `pricing_caps`, `pricing_caplets2`, etc.

10.8.4 The Optim package

The Optim[1] package contains some optimization procedures that we used to find a smoother volatility surface. We did not implemented the gradient for the function we wanted to minimize, but luckily Optim is able to automatically estimate the derivatives using finite differences. Alternatively, one can use the keyword "autodiff" to enable automatic differentiation which yields exact gradients, although we did not take advantage of this functionality.

10.8.5 Methods that modify the input values

When filling in missing data, as explained in a previous section, we took advantage another functionality of Julia that allows us to directly modify the inputs of a function. We defined the function

```
filldata!(x::Vector{Float64},y::Vector{Float64},
          data::Array{Float64},iter::Int)
```

The exclamation mark (!) is a convention used in Julia stating that the input variables may be modified by the procedure. This is usually called "pass-by-reference", and some languages allow only this kind of behavior, while others

[1]`http://www.juliaopt.org/Optim.jl/latest/`

allow only "pass-by-value". Julia allows our software to be organised in both ways, although strictly speaking it is "pass-by-reference" in both cases.

If the "data" variable becomes very large, this will be very useful, as we would not need to allocate another copy of this array.

10.9 Conclusion

We have implemented several methods to obtain the volatilities of individual caplets (caplet stripping). In addition, we have implemented two methods for smoothing the volatility surface: the interpolation method and the penalty method.

The main advantage of the penalty method is the possibility to obtain a well controled smooth surface. Unfortunately, as we give a bigger value to λ we see the increasing price error. The moral is: "If you want more smoothness you have to pay with higher errors in recovering market prices."

To sum up, we have learned the meaning of some fixed income market derivatives (Caps and Caplets), the way they are priced and quoted through Black volatilities, and hard technical problems arising from simple questions. We reviewed some elementary (1-D root finding) methods, and other more advanced (non linear optimization with penalty terms). Further, we implemented all the methods from scratch, using an interesting and rather new computing language, which demanded an extra effort, but it was worth learning.

Bibliography

[1] R. White, Y. Iwashita, Eight ways to strip your caplets: An introduction to caplet stripping, OpenGamma Quantitative Research nr. 24, 2014.

[2] Julia language documentation: `http://docs.julialang.org/en/release-0.5/`