BACHELOR THESIS IN MATHEMATICS/
APPLIED MATHEMATICS

Pricing Caps in the Heath, Jarrow and Morton Framework Using Monte Carlo Simulations in a Java Applet

by

Michail Kalavrezos

Kandidatarbete i matematik/tillämpad matematik
Date: 2007-11-21

Project name: Pricing Caps in the Heath, Jarrow and Morton Framework Using Monte Carlo Simulations in a Java Applet

Author: Michail Kalavrezos

Supervisor: Dr Anatoliy Malyarenko

Examiner: Professor Dmitrii Silvestrov
ABSTRACT

In this paper the Heath, Jarrow and Morton (HJM) framework is applied in the programming language Java for the estimation of the future spot rate. The subcase of an exponential model for the diffusion coefficient (volatility) is used for the pricing of interest rate derivatives (caps).

Keywords: Heath, Jarrow and Morton framework, Java, interest rate derivatives, caps, Monte Carlo Simulations
Acknowledgement

I would like to thank my supervisor Dr. Anatoliy Malyarenko for his valuable guidance and advice that led to the completion of my thesis.
# Table of Contents

**LIST OF FIGURES** ................................................................................................................................. 6

**INTRODUCTION** ........................................................................................................................................ 7

**PART I: THEORETICAL CONSIDERATIONS** .............................................................................................. 8

1.1 HEATH JARROW MORTON FRAMEWORK ......................................................................................... 8
1.2 SUBCASE: EXPONENTIAL MODEL ....................................................................................................... 11
1.3 MONTE CARLO SIMULATIONS ........................................................................................................... 11
1.4 INTEREST RATE CAP .......................................................................................................................... 12
1.5 GRAPHICAL DESCRIPTION OF A CAP WITH N CAPLETS .............................................................. 13
1.6 CAP’S VALUE ....................................................................................................................................... 12
1.7 HOW TO PRICE A CAP ....................................................................................................................... 13

**PART II: PROGRAM REALIZATION** ....................................................................................................... 14

2.1 NOTATION ............................................................................................................................................... 14
2.2 ALGORITHM .......................................................................................................................................... 14
2.3 USER’S GUIDE ...................................................................................................................................... 17
   2.3.1 Description of the applet and instructions .................................................................................. 17
   2.3.2 Inserting wrong values for the parameters ............................................................................... 19

**PART III: COMMENTS AND CONCLUSION** ............................................................................................. 20

3.1 COMMENTS ON THE VALUES ............................................................................................................. 20
3.2 CONCLUSION ....................................................................................................................................... 21

**LIST OF REFERENCES** ............................................................................................................................ 23

**APPENDIX A** ........................................................................................................................................... 24
List of Figures

Figure 1.1 Cap with n caplets.................................................................11
Figure 2.1 Notation used in the program .............................................12
Figure 2.2 The steps of the algorithm ..................................................14
Figure 2.3 The input panel.................................................................15
Figure 2.4 Reset period chosen to be one month ..................................15
Figure 2.5 The input and output panels after the execution of the program........17
Figure 2.6 Information window for a wrong entry for the number of intervals........18
Figure 2.7 Information window for a wrong entry for the cap rate...............18
Figure 3.1 Undesired result due to the exponential volatility structure............19
Introduction

The interest rate (or term structure) models have been assisting in the more accurate valuation of the financial instruments. According to Hull [1,Chapter 28] a broad classification of these models stems from the origin of their derivation. If they are derived from basic economic theories, they are called equilibrium models. The equilibrium models begin with assumptions about economic variables and reach a process for the short rate. If the models are planned to be consistent with the current term structure they are called no-arbitrage models.

A more general approach to creating a term structure model is the Heath, Jarrow and Morton (HJM) [2] framework. The two reasons that this framework is given the characterization ‘a more general approach’ are because it allows for a choice of the volatility structure of the forward curve and has no restriction over the number of factors to be used. The implementation of the HJM framework provides a model with independent random variables in it which can be simulated by a random number generating program.

The purpose of this paper is to apply the HJM framework using Monte Carlo simulations in the programming language Java for the estimation of the future spot rate. This rate will then be used for the pricing of caps. The volatility structure is chosen to follow an exponential model.

The remaining paper consists of three parts with the following structure:

The first part contains the theoretical considerations explaining the derivation of the main idea of the HJM framework and the choice of the volatility structure. A brief explanation of the caps as well as a graphical depiction of them follow.

The second part deals with the realization of the program starting with the notation translation from mathematics to Java language. A user’s guide is also presented including the main features of the program.

The third part includes the comments on the data generated by the program and the conclusion. The appendix A contains the complete program.
Part I: Theoretical considerations

1.1 Heath Jarrow Morton framework

In their paper David Heath, Bob Jarrow and Andy Morton (HJM) explain the no-arbitrage conditions that must be met by a model of the yield curve. For the description of the model the following notation will be used:

\[ B(t,T) \] : Price at time \( t \) of a zero-coupon bond with principal $1 maturing at time \( T \)

\[ \nu(t,T) \] : Volatility of \( B(t,T) \)

\[ f(t,T) \] : Instantaneous forward rate at time \( t \) for a financial contract commencing at time \( T \)

\[ r(t) \] : Short-term risk-free interest rate at time \( t \)

\( W \) : Standard Brownian motion

The price of the bond, \( B(t,T) \), should be equal to the face value discounted to the current time. For that discounting the forward rates are used and the relation between bond price and forward rates is

\[
B(t,T) = \exp\left(-\int_{t}^{T} f(t,u)du\right) \tag{1.1}
\]

Taking the logarithm on both sides of (1.1) and differentiating we reach the following

\[
f(t,T) = -\frac{\partial}{\partial T} \log B(t,T) \tag{1.2}
\]

In the HJM framework the forward rate curve follows the stochastic process

\[
df(t,T) = \mu(t,T)dt + \sigma(t,T)dW(t) \tag{1.3}
\]

The differential is with respect to time \( t \). The drift and diffusion coefficients, \( \mu \) and \( \sigma \) respectively could be either stochastic or functions with independent variables equal to the past and current prices of the forward rates and their volatilities. In order to reach the centrepiece of the HJM framework this paper will refer to the mathematical path followed in Paul Glasserman’s book [3]. Therefore the two coefficients will be considered as deterministic functions of time \( t \), maturity \( T \) and current forward curve.
For the risk-neutral dynamics of asset prices, the price of a dividend free asset is a martingale when divided by the numeraire asset associated with the risk neutral measure

$$\beta(t) = \exp\left(\int_0^t r(u)du\right) \quad (1.4)$$

In the case of interest rates the above condition is not directly applied since the forward rates are not asset prices. The following dynamics are used in order to expect that the discounted bond prices are positive martingales

$$\frac{dB(t,T)}{B(t,T)} = r(t)dt + v(t,T)^T dW(t), \quad 0 \leq t \leq T^*. \quad (1.5)$$

The diffusion coefficient, i.e. the bond volatilities may be functions of current bond prices. The relation between bond prices and forward rates described in (1.2) allows for a unique correspondence between a bond price and a forward rate therefore the bond volatilities may be functions of current forward rates. If we apply Ito’s formula, see Kijima [4, Chapter 1], on (1.5) we get the following

$$d \log B(t,T) = \left[ r(t) - \frac{1}{2} v(t,T)^T v(t,T) \right] dt + v(t,T)^T dW(t) \quad (1.6)$$

which in turn can be differentiated with respect to $T$ and then by interchanging the order of differentiation with respect to $t$ and $T$ the result is

$$\frac{\partial}{\partial T} d \log B(t,T) = \frac{\partial}{\partial T} \left[ r(t) - \frac{1}{2} v(t,T)^T v(t,T) \right] dt + \frac{\partial}{\partial T} v(t,T)^T dW(t) \quad (1.7)$$

Through the use of (1.2) we get

$$df(t,T) = -\frac{\partial}{\partial T} d \log B(t,T) = -\frac{\partial}{\partial T} \left[ r(t) - \frac{1}{2} v(t,T)^T v(t,T) \right] dt - \frac{\partial}{\partial T} v(t,T)^T dW(t) \quad (1.8)$$

Since both (1.3) and (1.8) express the evolution of the forward rate through a stochastic differential equation, their drift and diffusion coefficients must be equal. We start with the diffusion coefficient
\[ \sigma(t,T) = -\frac{\partial}{\partial T} v(t,T) \]  

(1.9)

By differentiating (1.9) we get

\[ v(t,T) = -\int_t^T \sigma(t,u) du + h \]  

(1.10)

where \( h \) is a constant. The condition for the volatility at maturity is that it is equal to zero

\[ v(T,T) = 0 \]

This is easily accepted since the bond’s price at maturity is (theoretically) equal to the principal ($1). Consequently \( h \) becomes equal to zero and (1.10) becomes

\[ v(t,T) = -\int_t^T \sigma(t,u) du \]  

(1.11)

As for the drift coefficients

\[ \mu(t,T) = -\frac{\partial}{\partial T} \left[ r(t) - \frac{1}{2} v(t,T)^T v(t,T) \right] = \left( \frac{\partial}{\partial T} v(t,T) \right)^T v(t,T) \]  

(1.12)

Through the relation of (1.11) we can rewrite (1.12) as

\[ \mu(t,T) = \sigma(t,T)^T \int_t^T \sigma(t,u) du \]  

(1.13)

With the result of (1.13) substituted in (1.3) the stochastic differential equation of the evolution of the forward rate becomes

\[ df(t,T) = \left( \sigma(t,T)^T \int_t^T \sigma(t,u) du \right) dt + \sigma(t,T)^T dW(t) \]  

(1.14)

which is the main finding of the HJM framework and it describes the no-arbitrage condition dynamics of the forward curve under the risk neutral measure. Since we have \( d \) number of factors in the \( \sigma \) vector we use the subscript \( i \) for the vector components and (1.14) becomes
Each factor $\sigma_i$ accounts for a contribution to the drift coefficient and to the diffusion coefficient.

### 1.2 Subcase: Exponential model

As it was mentioned in the introduction, the HJM framework provides the evolution of the forward rate curve without a strictly defined volatility structure. In this paper the volatility structure is chosen to be

$$\sigma(t,T) = \sigma \exp(-\alpha(T - t)) \quad (1.16)$$

with both $\sigma$ and $\alpha$ having positive values. This particular structure causes smaller movements for the forward rates with long maturities that the ones with short maturities. If the diffusion coefficient in (1.13) is substituted by (1.16) we get the drift coefficient to be

$$\mu(t,T) = \sigma^2 e^{-\alpha(T-t)} \int_t^T e^{-\alpha(T-t)} dt = \frac{\sigma^2}{\alpha} (e^{-2\alpha(T-t)} - e^{-\alpha(T-t)})$$

Now the drift coefficient and the diffusion coefficient (1.16) and (1.17) can be used for the SDE of the forward curve evolution.

### 1.3 Monte Carlo Simulations

The main idea in the Monte Carlo simulations according to Kijima [4, Chapter 10] is that by making use of the Strong Law of Large Numbers and the Central Limit Theorem and in order to evaluate the expected value of a random variable we need to generate sufficiently many independent, identically distributed random numbers.

The number of generated numbers $n$ that guarantees the desired confidence interval $\varepsilon$ is

$$n = \left( \frac{\sigma \chi_{\alpha}}{\varepsilon} \right)^2$$

with $\chi_{\alpha}$ denoting the 100$(1-\alpha)$ percentile of the standard normal distribution.
1.4 Interest Rate Cap

An interest rate cap is a derivative (financial contract) with an interest rate as the underlying. A cap may be extending over one or more time periods. Every single time period (called reset period), whose length is agreed by the two parts of the contract, is covered by a caplet. For each caplet there is a prespecified interest rate (cap rate). This rate is set in relation to a major interest floating rate index (e.g., Libor, Stibor etc.). In case that this rate index exceeds the cap rate claims can be made by the buyer of the cap. The caplet therefore is providing protection against movements of the rate index over a prespecified maximum level within a certain period.

Apart from buying a cap for protection (hedging) against interest rate risk (for example someone who has issued debt with floating rate) a cap can be bought for speculative reasons from someone who expects a raise in the interest rate.

The price paid by the buyer of the cap is called the premium. Since the caps are over-the-counter derivatives the length of the reset period coincides with the index reset period and may be extending from one day to one year e.g., in case of Libor[5]: overnight/sunday next, 1 week, 2 weeks, 1 month, 2 months,…,12 months.

1.6 Cap’s value

The cap value consists of two parts:

- **Intrinsic Value**

  If the cap rate is smaller than the implied forward rate, the cap has Intrinsic value. We can get the implied forward rate directly from a forward rate agreement (FRA) or by estimating the forward rate curve from the zero-coupon bond prices in the market. If the cap rate is smaller than the implied forward (i.e., has positive Intrinsic value), the cap is said to be ‘in the money’. Accordingly if the cap rate is greater than or equal to the implied forward the cap is said to be ‘out of the money’ and ‘at the money’ respectively.
• **Time Value**

The cap provides a guarantee (if it is bought for hedging) that the future rate the debt-issuer has to pay will not exceed a certain level. The level of the interest rate changes according to the policy followed by the central governments and their understanding of the needs of the economy. Changes in the condition of the markets lead to changes in the interest rate. There are periods (years) with many changes and other relative flatter periods. The volatility of the interest rate is responsible for the time value of the cap.

The impact of the volatility on the cap price becomes smaller as time approaches the maturity of the cap. Therefore, in a market with no changes, the passing of time will simply reduce the cap’s value.

### 1.5 Graphical description of a cap with n caplets

To ease the explanation of a cap the following figure is displayed.

![Figure 1.1 Cap with n caplets](image)

**Figure 1.1 Cap with n caplets**

### 1.7 How to price a cap

The price of a cap is the sum of the prices of its caplets. The price of a caplet can be estimated by finding the present value of the expected gain from the caplet. To find the expected gain we need to find primarily the expected value of the interest rate index at the maturity of the caplet. Then the difference (if positive) between this interest rate index and the cap rate should be multiplied with the principal amount and the reset period. This product should be discounted to find the the present value of the caplet.
In this paper the expected interest rate index is estimated with the use of (1.16).

**Part II: Program realization**

### 2.1 Notation

<table>
<thead>
<tr>
<th>Mathematical</th>
<th>Explanation</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1, \sigma_2 )</td>
<td>parameterisation coefficients</td>
<td>sigmaone, sigmatwo</td>
</tr>
<tr>
<td>( \alpha_1, \alpha_2 )</td>
<td>parameterisation coefficients</td>
<td>alpaone, alphatwo</td>
</tr>
<tr>
<td>( u_1, u_2, ..., u_M )</td>
<td>known time moments</td>
<td>knownTimePoints[]</td>
</tr>
<tr>
<td>( f(0, u_1), ..., f(0, u_M) )</td>
<td>known values of implied forward rate</td>
<td>forwardRates[]</td>
</tr>
<tr>
<td>( \Delta t )</td>
<td>simulation interval (measured in days)</td>
<td>deltati</td>
</tr>
<tr>
<td>( p )</td>
<td>periods covered (no of caplets)</td>
<td>p</td>
</tr>
<tr>
<td>( r )</td>
<td>reset period(length of a caplet in days)</td>
<td>r</td>
</tr>
<tr>
<td>( s )</td>
<td>number of simulation intervals within a caplet</td>
<td>s</td>
</tr>
<tr>
<td>( f(0, t_l) ) for ( l=0,1,...,N,N+1 )</td>
<td>forward rate after the interpolation</td>
<td>matchedforward[]</td>
</tr>
<tr>
<td>( \hat{f}(0, t_l) ) for ( l=0,1,...N )</td>
<td>discrete forward rate</td>
<td>discreteforwardrate[]</td>
</tr>
<tr>
<td>( \hat{\sigma}_1, \hat{\sigma}_2 )</td>
<td>the diffusion coefficients</td>
<td>sigmaonehat , sigmatwohat</td>
</tr>
<tr>
<td>( \sum_{l=1}^{s} \hat{\sigma}<em>1(t</em>{i-1}, t_i) )</td>
<td>the sum of the diffusion coefficient ( \hat{\sigma}_1 )</td>
<td>summaone</td>
</tr>
<tr>
<td>( \sum_{l=1}^{s} \hat{\sigma}<em>2(t</em>{i-1}, t_i) )</td>
<td>the sum of the diffusion coefficient ( \hat{\sigma}_2 )</td>
<td>summatwo</td>
</tr>
<tr>
<td>( \hat{\mu}_1, \hat{\mu}_2 )</td>
<td>the drift</td>
<td>miouonehat , mioutwohat</td>
</tr>
<tr>
<td>( \hat{f}(t_l, t_i) ) for ( l=0,1,...N )</td>
<td>simulated spot rate</td>
<td>fHatSpot[]</td>
</tr>
<tr>
<td>( c )</td>
<td>cap rate(strike rate)</td>
<td>c</td>
</tr>
<tr>
<td>( n )</td>
<td>number of Monte Carlo simulations</td>
<td>n</td>
</tr>
<tr>
<td>( P_i )</td>
<td>payoff of the caplet ( i ) at its maturity ( t_M )</td>
<td>P[i]</td>
</tr>
<tr>
<td>( D_M )</td>
<td>discount factor for the caplet ( i ) at its maturity ( t_M )</td>
<td>D[i]</td>
</tr>
<tr>
<td>( PV_i )</td>
<td>Present value of the caplet</td>
<td>presentValue[]</td>
</tr>
<tr>
<td>( P_{cap} )</td>
<td>price of the cap</td>
<td>PV</td>
</tr>
</tbody>
</table>

Figure 2.1 Notation used in the program

### 2.2 Algorithm

The beginning of the process consists of defining the array of the time points for the simulation intervals throughout the cap. These points are

\[
t_l = (l \times \Delta t)/360 \quad \text{where} \quad l=0,1,...,(s*p+1) \quad \text{and} \quad s*p=N (=\text{maturity of nth caplet})
\]

we divide by 360 since this is the time convention we apply. Now that the time points have been defined we use the already given implied forward rate value at specific time points to
create the discrete forward rate curve. This forward rate curve is created with the use of cubic spline interpolation.

The diffusion coefficients \( \hat{\sigma}_1, \hat{\sigma}_2 \) are then calculated as

\[
\hat{\sigma}_1(t_{i-1}, t_i) = \sigma_i \exp(-\alpha_i(l - i + 1)\Delta t),
\]

\[
\hat{\sigma}_2(t_{i-1}, t_i) = \sigma_2 \exp(-\alpha_2(l - i + 1)\Delta t)
\]

for \( 1 \leq i \leq N \) and \( i \leq l \leq N \).

The next step is to calculate the drift coefficients

\[
\hat{\mu}_1(t_{i-1}, t_j) = \frac{\Delta t}{2} \left[ \left( \sum_{l=i}^{j} \hat{\sigma}_1(t_{l-1}, t_l) \right)^2 - \frac{1}{2} \left( \sum_{l=i}^{j-1} \hat{\sigma}_1(t_{l-1}, t_l) \right)^2 \right]
\]

\[
\hat{\mu}_2(t_{i-1}, t_j) = \frac{\Delta t}{2} \left[ \left( \sum_{l=i}^{j} \hat{\sigma}_2(t_{l-1}, t_l) \right)^2 - \frac{1}{2} \left( \sum_{l=i}^{j-1} \hat{\sigma}_2(t_{l-1}, t_l) \right)^2 \right]
\]

Now we have the necessary formulas so that we can simulate the discrete forward rate according to the following

\[
\hat{f}(t_i, t_j) = \hat{f}(t_{i-1}, t_j) + [\hat{\mu}_1(t_{i-1}, t_j) + \hat{\mu}_2(t_{i-1}, t_j)]\Delta t + \sqrt{\Delta t} [\hat{\sigma}_1(t_{i-1}, t_j) Z_{1i} + \hat{\sigma}_2(t_{i-1}, t_j) Z_{2j}]
\]

Where \( i \leq j \leq N \) and \( Z_{1i}, Z_{2j} \) are independent standard normal random variables.

The Monte Carlo simulations are repeated \( n \) times and each step in the procedure is providing one more element of the array of the simulated spot rate. The procedure is depicted in the following figure.
| \( \hat{f} (0, t_1) \) | \( \hat{f}(0,0) \) | \( \hat{f}(0,1) \) | \( \hat{f}(0,2) \) | \( \ldots \) | \( \ldots \) | \( \hat{f}(0,N) \) |
| \( \hat{\sigma}_1, \hat{\sigma}_2 \) | \( \hat{\sigma}_{1,2} (t_0, t_1) \) | \( \hat{\sigma}_{1,2} (t_1, t_2) \) | \( \ldots \) | \( \ldots \) | \( \hat{\sigma}_{1,2} (t_{N-1}, t_N) \) |
| \( \sum_{i=1}^{N} \hat{\sigma}_{i,2} (t_{i-1}, t_i) \) | \( \sum_{i=1}^{N} \hat{\sigma}_{i,2} (t_0, t_i) \) | \( \sum_{i=1}^{N} \hat{\sigma}_{i,2} (t_0, t_i) \) | \( \ldots \) | \( \ldots \) | \( \sum_{i=1}^{N} \hat{\sigma}_{i,2} (t_0, t_i) \) |
| \( \mu_1, \mu_2 \) | formulas 1, 2 | formulas 1, 2 | \( \ldots \) | \( \ldots \) | formulas 1, 2 |

**Point 1**

After the simulation

| \( \hat{f}(1,1) \) | \( \hat{f}(1,2) \) | \( \ldots \) | \( \ldots \) | \( \hat{f}(1,N) \) |
| \( \hat{\sigma}_1, \hat{\sigma}_2 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( \sum_{i=1}^{N} \hat{\sigma}_{i,2} (t_{i-1}, t_i) \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( \mu_1, \mu_2 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |

**Point 2**

| \( \hat{f}(2,2) \) | \( \ldots \) | \( \ldots \) | \( \hat{f}(2,N) \) |
| \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |

**Final array**

| \( \hat{f} (t_i, t_j) \) | \( \downarrow \) | \( \downarrow \) | \( \downarrow \) | \( \hat{f}(N,N) \) |

Figure 2.2 The steps of the algorithm

Explanation about Point 1, Point 2, etc:

When the first row (Point 1) is created, the simulations make use of the array \( \hat{f} (0, t_i) \) (discreteforwardrate[] in Java) and a new array \( \text{fhatsim} \) in Java is created and its values are assigned to another array \( \text{taxi[]} \) in Java which in turn is used for the construction of the second raw (Point 2) and so on.

Once the array of the simulated spot rate \( \text{fHatspot} \) in Java is created we can calculate the payoff of each caplet at the time of payment (one reset period after the caplet’s maturity).

This payoff discounted at the maturity of the caplet is

\[
P_i = \frac{\left( \hat{f}(t_{M,i}, t_M) - c \right) \left( r/360 \right)}{1 + \left( \hat{f}(t_{M,i}, t_M) \left( r/360 \right) \right)}
\]

with \( \hat{f} (t_{M,i}, t_M) \) equal to the simulated spot rate at the maturity of the caplet \( i \).

To find the present value of the caplet’s payoff, \( Pvi \), we have to multiply \( P_i \) with the discount factor \( D_{iM} \)
\[ D_{IM} = \exp \left[ \sum_{i=0}^{IM} \hat{f}(t_i, t_{i+1}) \Delta t \right] \quad \text{and} \quad PV_i = P_i \cdot D_{IM} \]

Finally for the price of the cap, \( P_{\text{cap}} \), all the caplets’ present values must be added

\[ P_{\text{cap}} = \sum_{i=1}^{P} PV_i \]

### 2.3 User’s guide

#### 2.3.1 Description of the applet and instructions

The applet consists of two panels appearing in vertical order: the input panel and the output panel. The input panel is shown in figure 2.3.

![Figure 2.3 The input panel](image)

The input panel requires first a choice to be made upon the length of the reset period. This choice is realized by clicking on one of the buttons at the top row of the input panel.

![Figure 2.4 Reset period chosen to be one month](image)

Secondly the user need to enter the values of the following parameters:

1. **Principal**: the amount upon which the cap will be written
Acceptable value, $v$: $1 < v < 2000000000$

2. **Cap rate**: the maximum interest rate that can be reached by the interest rate index before claims can be realized

Acceptable value, $v$: $0 < v$

3. **Periods covered (caplets)**: the integer number of the reset periods covered by the cap

Acceptable value, $v$: $0 < v$ and $v$ is an integer

4. **Number of iterations =I**: the maximum number of the Monte Carlo simulations to be executed

Acceptable value, $v$: $1 < v < 2000000000$ and $v$ is an integer

5. **Sigma One**: a positive number used in the volatility structure

Acceptable value, $v$: $0 < v$

6. **Alpha One**: a positive number used in the volatility structure

Acceptable value, $v$: $0 < v$

7. **Sigma Two**: a positive number used in the volatility structure

Acceptable value, $v$: $0 < v$

8. **Alpha Two**: a positive number used in the volatility structure

Acceptable value, $v$: $0 < v$

The last input to be inserted is the obtained forward rate at the corresponding time points. Although there are default values for certain time points these points as well as the forward rate values can be changed by clicking on the value and inserting the new one. There is also the possibility to insert forward rates at intermediate time points by clicking on the row preceding the one to be added and then on the button with the label ‘Add’. The new row will appear with a time difference of 0.25 years (approximately three months). The forward rate values can also be removed by clicking on the row and then on the button with the label ‘Delete’.

In case that no choice is made about the reset period or no values are inserted in the text fields the applet will execute with the default values.

When the values have been inserted the execution of the program starts by clicking on the button with the label ‘Calculate’. To set all the fields to the default values the user has to click on the button with the label ‘Reset’.

Once the ‘Calculate’ button has been pressed the program will be executed and the results will appear in the output panel.
Figure 2.5 The input and output panels after the execution of the program.

The output panel displays the price of the cap using three different numbers of simulations and the evolution of the forward rate is depicted in the diagram.

### 2.3.2 Inserting wrong values for the parameters

The case of inserting not acceptable values should not prevent the user from continuing the calculations. Therefore windows with information about the wrong entry are popping up and the text field with the wrong entry is set to the last acceptable value inserted after the user has clicked ‘OK’ or exit (‘x’). In figure (2.5) the number of simulation intervals has been wrongly set to a non-integer number (2.2). The information window has popped up and the program execution will not continue before the user has closed this window (as explained previously) making sure that the user receives this piece of information.
In figure (2.6) the value of the cap rate has been wrongly set to a negative number (-4.25). The information window has popped up and points the place and type of mistake. As previously the program expects for this window to close before the execution can continue.

Figure 2.6 Information window for a wrong entry for the number of intervals

Figure 2.7 Information window for a wrong entry for the cap rate

**Part III: Comments and conclusion**

**3.1 Comments on the values**

Since the volatility structure has been chosen to be exponential the parameters for the volatility should be chosen with caution. High values of the \( \text{sigmas} \) and \( \text{alphas} \) make the volatility
soaring and the result is extremely high (unreasonable) expected spot rates. In figure (3.1) the high values of \(\sigma_1\) and \(\sigma_2\) result in an expected interest rate twice as high as the current within five months.

![Applet Input and Output](image)

Figure 3.1 Undesired result due to the exponential volatility structure

### 3.2 Conclusion

The purpose of this paper was to apply the HJM framework using Monte Carlo simulations in the programming language Java for the estimation of the future spot rate. This rate was then used for the pricing of caps. The volatility structure was chosen to follow an exponential model.
The application provides the price of a cap under the no-arbitrage conditions. The choice of the volatility structure requires the (volatility) parameters to be adjusted when the maturity of the cap changes.
List of references


/**
 * @(#)  HJMmodel.java  1.0 24/08/2007
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 * Description: HJM framework for pricing caps
 * @version 1.0 Aug 07
 * @author Michail Kalavrezos
 * Mail: michail_kalavrezos@yahoo.se
 */

import java.util.Random;
import java.awt.*;
import java.text.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.border.*;
import org.jfree.chart.*;
import org.jfree.chart.axis.*;
import org.jfree.chart.plot.*;
import org.jfree.data.xy.*;
import org.jfree.data.statistics.*;

public class HJMmodel extends JApplet
    implements FocusListener,
                ActionListener,
                TableModelListener
{
    // class variables

    // panels
    private JPanel mainPanel = null;
    private JPanel inputPanel = null;
    private JPanel outputPanel = null;
    private JPanel output1Panel = null;
    private JPanel buttonPanel = null;
    private JPanel button1Panel = null;
    private JPanel dataPanel = null;
    private JPanel data1Panel = null;
    private JPanel button2Panel = null;

private JPanel button3Panel = null;

private ChartPanel graphPanel = null;

// button groups
private ButtonGroup fileGroup = null;
private ButtonGroup methodGroup = null;
private ButtonGroup graphGroup = null;
private ButtonGroup periodGroup = null;

// radio buttons
private JRadioButton onedayButton = null;
private JRadioButton oneweekButton = null;
private JRadioButton onemonthButton = null;
private JRadioButton threemonthsButton = null;
private JRadioButton sixmonthsButton = null;

// buttons
private JButton calculateButton = null;
private JButton resetButton = null;
private JButton addButton = null;
private JButton deleteButton = null;

// Scroll panes
private JScrollPane forwardPane = null;

// table models
private ForwardTableModel forwardModel = null;

// tables
private JTable forwardTable = null;

// text fields
// String constants
private final String ONEDAY = "One day";
private final String ONEWEEK = "One week";
private final String ONEMONTH = "One month";
private final String THREEMONTHS = "Three months";
private final String SIXMONTHS = "Six months";

private final String CALCULATE = "Calculate";
private final String RESET = "Reset";
private final String INPUT = "Input";
private final String OUTPUT = "Output";

private final String SELECT = "Select a row";
private final String ERROR = "Error";

// table columns names
private final String TIME = "Time in years";
private final String FORWARD = "Forward rate";
// button names
private final String ADD = "Add";
private final String DELETE = "Delete";
private final String NUMBEROFITERATIONS_LABEL = "Number of Iterations= I";
private JTextField numberOfiterationsField = null;

private final String PRINCIPAL_LABEL = " Principal";
private JTextField principalField = null;

private final String CAPRATE_LABEL = " Cap rate";
private JTextField caprateField = null;

private final String SIGMAONE_LABEL = " Sigma One";
private JTextField sigmaoneField = null;

private final String SIGMATWO_LABEL = " Sigma Two";
private JTextField sigmatwoField = null;

private final String ALPHAOONE_LABEL = " Alpha One";
private JTextField alphaooneField = null;

private final String ALPHATWO_LABEL = " Alpha Two";
private JTextField alphatwoField = null;

private final String CAPPRICE_LABEL = "Cap Price (I)"
private JTextField cappriceField = null;

private final String CAPPRICE2_LABEL = "Cap Price (I/10)"
private JTextField capprice2Field = null;

private final String CAPPRICE4_LABEL = "Cap Price (I/100)"
private JTextField capprice4Field = null;

private final String PERIODSCOVERED_LABEL = "Periods covered (caplets)"
private JTextField periodscoveredField = null;

private final String SIMULATIONINTERVALS_LABEL = "Simulation intervals"
private JTextField simulationintervalsField = null;

private final String RESETPERIOD_LABEL = "RESET PERIOD"

// Tooltips
private final String NUMBEROFITERATIONS_TOOLTIP = "The Recommended value is >10000";
private final String PRINCIPAL_TOOLTIP = "The initial amount borrowed";
private final String CAPRATE_TOOLTIP = "Interest rate expressed in percentage points";
private final String SIGMAONE_TOOLTIP = "Estimated or observed value of sigma one";
private final String SIGMATWO_TOOLTIP = "Estimated or observed value of sigma two";
private final String ALPHAOONE_TOOLTIP = "Estimated or observed value of alpha one";
private final String ALPHATWO_TOOLTIP = "Estimated or observed value of alpha two";
private final String CAPPRICE_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
private final String CAPPRICE2_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";
private final String CAPPRICE4_TOOLTIP = "DO NOT INSERT ANY VALUE HERE";

private final String SIMULATIONINTERVALS_TOOLTIP = "Integer number of simulation intervals within one reset period";
private final String PERIODSCOVERED_TOOLTIP = "Insert the integer number of the reset periods covered by the cap";

//Error messages
private final String NOT_A_NUMBER = " Enter a number";
private final String NOT_INTEGER = " Enter an integer number";
private final String NOT_DOUBLE = " Enter a double number";
private final String NOT_POSITIVE = "Enter a positive number";

// numerical constants
private final int NUMBEROFITERATIONS = 100000;
private final int PRINCIPAL = 1000000;
private static double CAPRATE = 4.2;
private static double SIGMAONE = 0.00002;
private static double SIGMATWO = 0.00003;
private static double ALPHAONE = 0.022;
private static double ALPHATWO = 0.017;
private static double CAPPRICE = 0.0;
private static double CAPPRICE2 = 0.0;
private static double CAPPRICE4 = 0.0;

private int PERIODSCOVERED = 5;
private int SIMULATIONINTERVALS = 3;

private final double[] KNOWN_TIME_POINTS={0.00,1.00,2.00,3.00,6.00,10.00};
private final double[] FORWARD1_RATES = {4.00,4.51,4.82,5.14,5.54,5.85};

// numerical variables
static int r;
private int numberOfiterations = NUMBEROFITERATIONS;
static int n; //numberOfiterations;
private int principal = PRINCIPAL;
//static int prin;//principal;
private double caprate = CAPRATE;
//static int c;//caprate;
private static double sigmaone = SIGMAONE;
private static double sigmatwo = SIGMATWO;
private static double alphaone = ALPHAONE;
private static double alphatwo = ALPHATWO;
private double capprice = CAPPRICE;
private double capprice2 = CAPPRICE2;
private double capprice4 = CAPPRICE4;

private int periodscovered = PERIODSCOVERED;
static int p; //periodscovered;
private int simulationintervals = SIMULATIONINTERVALS ;
static int s; //simulationintervals;
//boolean variable

// number formatters
private DecimalFormat numberFormatter = null;
private DecimalFormat numberFormatter1 = null;

// class methods
// initialising
public void init () {

    // Initialise formatter
    DecimalFormatSymbols symbols = new DecimalFormatSymbols();
    symbols.setDecimalSeparator('.');
    numberFormatter = new DecimalFormat("#.#####", symbols);
    numberFormatter1 = new DecimalFormat("#,###.##", symbols);

    // get content pane
    Container contentPane = getContentPane();

    // create main panel
    mainPanel = new JPanel(new BorderLayout());
    mainPanel.setLayout(new GridLayout(0,1));
    // add main panel to content pane
    contentPane.add(mainPanel);

    // create input panel
    inputPanel = new JPanel(new BorderLayout());
    inputPanel.setPreferredSize(new Dimension(500,200));
    inputPanel.setBorder(new TitledBorder(INPUT));
    // add it to the main panel
    mainPanel.add(inputPanel);

    button2Panel = new JPanel();
    button2Panel.setLayout(new BoxLayout(button2Panel,BoxLayout.X_AXIS));
    button2Panel.setPreferredSize(new Dimension(200,20));
    // add it to the input panel
    inputPanel.add(button2Panel,BorderLayout.NORTH);

    JLabel label= new JLabel(RESETPERIOD_LABEL);
    button2Panel.add(label);

    // create button group
    periodGroup = new ButtonGroup();

    // create oneday button
    onedayButton = new JRadioButton(ONEDAY);
    periodGroup.add(onedayButton);
    // add action listener
    onedayButton.addActionListener(this);
    // add it to button panel
    button2Panel.add(onedayButton);

    // create oneweek button
    oneweekButton = new JRadioButton(ONEWEEK);
    periodGroup.add(oneweekButton);
    // add action listener
    oneweekButton.addActionListener(this);
    // add it to button panel
button2Panel.add(oneweekButton);

    // create onemonth button
    onemonthButton = new JRadioButton(ONEMONTH);
    periodGroup.add(onemonthButton);
    onemonthButton.setSelected(true);

    // add action listener
    onemonthButton.addActionListener(this);
    // add it to button panel
    button2Panel.add(onemonthButton);

threemonthsButton = new JRadioButton(THREEMONTHS);
    periodGroup.add(threemonthsButton);
    // add action listener
    threemonthsButton.addActionListener(this);
    // add it to button panel
    button2Panel.add(threemonthsButton);

    sixmonthsButton = new JRadioButton(SIXMONTHS);
    periodGroup.add(sixmonthsButton);
    // add action listener
    sixmonthsButton.addActionListener(this);
    // add it to button panel
    button2Panel.add(sixmonthsButton);

    // create button 1 panel
    button1Panel = new JPanel();
    // add it to the input panel
    inputPanel.add(button1Panel,BorderLayout.SOUTH);

    // create calculate button
    calculateButton = new JButton(CALCULATE);
    // add action listener
    calculateButton.addActionListener(this);
    // add it to button panel
    button1Panel.add(calculateButton);

    //create reset button
    resetButton = new JButton(RESET);
    //add action listener
    resetButton.addActionListener(this);
    // add it to button panel
    button1Panel.add(resetButton);

    // create data panel for the inputs
    dataPanel = new JPanel(new GridLayout(0,2));
    dataPanel.setPreferredSize(new Dimension(300,100));
    // add it to input panel
    inputPanel.add(dataPanel,BorderLayout.WEST);

    //create labels, create text field, add focus listener and then add the
    //labels and text field to data panel
    label = new JLabel(PRINCIPAL_LABEL);
    principalField = new JTextField();
    dataPanel.add(label);
    principalField.addFocusListener(this);
    dataPanel.add(principalField);
label = new JLabel(CAPRATE_LABEL);
caprateField = new JTextField();
dataPanel.add(label);
caprateField.addFocusListener(this);
dataPanel.add(caprateField);

label = new JLabel(PERIODSCOVERED_LABEL);
periodscoveredField = new JTextField();
dataPanel.add(label);
periodscoveredField.addFocusListener(this);
dataPanel.add(periodscoveredField);

label = new JLabel(SIMULATIONINTERVALS_LABEL);
simulationintervalsField = new JTextField();
dataPanel.add(label);
simulationintervalsField.addFocusListener(this);
dataPanel.add(simulationintervalsField);

label = new JLabel(NUMBEROFITERATIONS_LABEL);
numberofiterationsField = new JTextField();
dataPanel.add(label);
numberofiterationsField.addFocusListener(this);
dataPanel.add(numberofiterationsField);

label = new JLabel(SIGMAONE_LABEL);
sigmaoneField = new JTextField();
dataPanel.add(label);
sigmaoneField.addFocusListener(this);
dataPanel.add(sigmaoneField);

label = new JLabel(ALPHAONE_LABEL);
alphaoneField = new JTextField();
dataPanel.add(label);
alphaoneField.addFocusListener(this);
dataPanel.add(alphaoneField);

label = new JLabel(SIGMATWO_LABEL);
sigmatwoField = new JTextField();
dataPanel.add(label);
sigmatwoField.addFocusListener(this);
dataPanel.add(sigmatwoField);

label = new JLabel(ALPHATWO_LABEL);
alphatwoField = new JTextField();
dataPanel.add(label);
alphatwoField.addFocusListener(this);
dataPanel.add(alphatwoField);

//create data panel
data1Panel = new JPanel(new BorderLayout());
data1Panel.setPreferredSize(new Dimension(180,100));
// add it to input panel
inputPanel.add(data1Panel,BorderLayout.EAST);

// create forward model
forwardModel = new ForwardTableModel();
// add columns
forwardModel.addColumn(TIME);
forwardModel.addColumn(FORWARD);
// add rows
for (int i=0; i<KNOWN_TIME_POINTS.length; i++) {
Object[] row = {
(new Double(KNOWN_TIME_POINTS[i]),
new Double(FORWARD1_RATES[i])));
//new
String(numberFormatter.format(FORWARD1_RATES[i]))};
forwardModel.addRow(row);

// add table model listener
forwardModel.addTableModelListener(this);

// create forward table
forwardTable = new JTable(forwardModel);
// put it into the scroll pane
forwardPane = new JScrollPane(forwardTable);
forwardTable.setPreferredScrollableViewportSize(
    new Dimension(100,100));
// install the custom editors on the columns
TableColumn col = forwardTable.getColumnModel().getColumn(0);
col.setCellEditor(new FirstColumnCellEditor());
col = forwardTable.getColumnModel().getColumn(1);
col.setCellEditor(new SecondColumnCellEditor());
// add it to forward panel
data1Panel.add(forwardPane, BorderLayout.CENTER);

button3Panel = new JPanel();
data1Panel.add(button3Panel, BorderLayout.SOUTH);

// create add button
addButton = new JButton(ADD);
// add action listener
addButton.addActionListener(this);
// add it to control panel
button3Panel.add(addButton);

// create delete button
deleteButton = new JButton(DELETE);
// add action listener
deleteButton.addActionListener(this);
// add it to control panel
button3Panel.add(deleteButton);

// create output panel
outputPanel = new JPanel();
outputPanel.setBorder(new TitledBorder(OUTPUT));
// add it to the main panel
mainPanel.add(outputPanel);

// create output1 panel
output1Panel = new JPanel();
output1Panel.setLayout(new BoxLayout(output1Panel,BoxLayout.X_AXIS));
output1Panel.setPreferredSize(new Dimension(480,20));
// add it to the output panel
outputPanel.add(output1Panel,BorderLayout.NORTH);

// add label and field to output1 panel
label = new JLabel(CAPPRICE_LABEL);
cappriceField = new JTextField();
output1Panel.add(label);
output1Panel.add(cappriceField);

// add label and field to outputONE panel
label = new JLabel(CAPPRICE2_LABEL);
capprice2Field = new JTextField();
output1Panel.add(label);
output1Panel.add(capprice2Field);

// add label and field to outputONE panel
label = new JLabel(CAPPRICE4_LABEL);
capprice4Field = new JTextField();
output1Panel.add(label);
output1Panel.add(capprice4Field);

// create output panel
graphPanel = new ChartPanel(null);
graphPanel.setPreferredSize(new Dimension(550, 250));
// graphPanel.setBorder(new TitledBorder(GRAPH));
// add it to the main panel
outputPanel.add(graphPanel, BorderLayout.SOUTH);

// add tooltip
numberofiterationsField.setToolTipText(NUMBEROFITERATIONS_TOOLTIP);
principalField.setToolTipText(PRINCIPAL_TOOLTIP);
caprateField.setToolTipText(CAPRATE_TOOLTIP);
// forwardratefileField.setToolTipText(FORWARDRATEFILE_TOOLTIP);
sigmaoneField.setToolTipText(SIGMAONE_TOOLTIP);
sigmatwoField.setToolTipText(SIGMATWO_TOOLTIP);
alphaconefField.setToolTipText(ALPHACONE_TOOLTIP);
alphatwoField.setToolTipText(ALPHATWO_TOOLTIP);
cappriceField.setToolTipText(CAPPRICE_TOOLTIP);
capprice2Field.setToolTipText(CAPPRICE2_TOOLTIP);
capprice4Field.setToolTipText(CAPPRICE4_TOOLTIP);
periodscoveredField.setToolTipText(PERIODSCOVERED_TOOLTIP);
simulationintervalsField.setToolTipText(SIMULATIONINTERVALS_TOOLTIP);

// set value
numberofiterationsField.setText(numberFormatter.format(NUMBEROFITERATIONS));
principalField.setText(numberFormatter.format(PRINCIPAL));
caprateField.setText(numberFormatter.format(CAPRATE));
sigmaoneField.setText(numberFormatter.format(SIGMAONE));
sigmatwoField.setText(numberFormatter.format(SIGMATWO));
alphaconefField.setText(numberFormatter.format(ALPHACONE));
alphatwoField.setText(numberFormatter.format(ALPHATWO));
cappriceField.setText(numberFormatter.format(CAPPRICE));
periodscoveredField.setText(numberFormatter.format(PERIODSCOVERED));
simulationintervalsField.setText(numberFormatter.format(SIMULATIONINTERVALS));

// method of Action listener
public void actionPerformed(ActionEvent e){
    // determine, who called action listener
    Object source = e.getSource();

    if (source == resetButton){
        // reset all TextField and variables to the initial values
        numberofiterations = NUMBEROFITERATIONS;
        numberofiterations=
        Field.setText(numberFormatter.format(NUMBEROFITERATIONS));
    }
principal = PRINCIPAL;
principalField.setText(numberFormatter.format(PRINCIPAL));

caprate = CAPRATE;
caprateField.setText(numberFormatter.format(CAPRATE));

sigmaone = SIGMAONE;
sigmaoneField.setText(numberFormatter.format(SIGMAONE));

alphaone = ALPHAONE;
alphaoneField.setText(numberFormatter.format(ALPHAONE));

sigmatwo = SIGMATWO;
sigmatwoField.setText(numberFormatter.format(SIGMATWO));

alphatwo = ALPHATWO;
alphatwoField.setText(numberFormatter.format(ALPHATWO));

periodscovered = PERIODSCOVERED;
periodscoveredField.setText(numberFormatter.format(PERIODSCOVERED));

simulationintervals = SIMULATIONINTERVALS;
simulationintervalsField.setText(numberFormatter.format(SIMULATIONINTERVALS));

capprice = CAPPRICE;
cappriceField.setText(numberFormatter.format(CAPPRICE));

capprice2 = CAPPRICE2;
capprice2Field.setText(numberFormatter.format(CAPPRICE2));

capprice4 = CAPPRICE4;
capprice4Field.setText(numberFormatter.format(CAPPRICE4));

if (source == calculateButton) {

    // read table into memory
    int size = forwardModel.getRowCount();
    double[] knownTimePoints = new double[size];
    double[] forwardRates = new double[size];
    for (int i=0; i<size; i++) {
        Object result = forwardModel.getValueAt(i,0);
        knownTimePoints[i] = ((Double)result).doubleValue();
        result = forwardModel.getValueAt(i,1);
        forwardRates[i] = ((Double)result).doubleValue();
    }

    // Here we call the methods that calculate the cap's price and display the price
    // First we set the reset period value according to the button pressed
    int r=0;
    if (onedayButton.isSelected()) {
        r=1;
    } else if (oneweekButton.isSelected()) {
        // Additional code...
    }

    }
r=7;
}
if (onemonthButton.isSelected()){
  r=30;
}
if (threemonthsButton.isSelected()){
  r=90;
}
if (sixmonthsButton.isSelected()){
  r=180;
}
int size1=(periodscovered*simulationintervals)+1;

//here we call the method that creates the relevant time points
double[] knownTimePoints1=(new Prohiro().getTimePoints(periodscovered,simulationintervals,r));

//here we call the method that creates the relevant simulated spot rates points
  double[] forwardRates1=(new Prohiro().getfHatSpot(periodscovered,simulationintervals,r,
  sigmagon, sigmatwo, alphanon, alphanon2, numberofiterations,knownTimePoints,forwardRates));

double[] forwardRates2=(new Prohiro().getfHatSpot(periodscovered,simulationintervals,r,
  sigmagon, sigmatwo, alphanon, alphanon2, (numberofiterations/10),knownTimePoints,forwardRates));
  double[] forwardRates3=(new Prohiro().getfHatSpot(periodscovered,simulationintervals,r,
  sigmagon, sigmatwo, alphanon, alphanon2, (numberofiterations/100),knownTimePoints,forwardRates));

//here we call the method that gives the present value of the cap
cappriceField.setText(numberFormatter1.format((new Prohiro().getPrice(forwardRates1,periodscovered,r,simulationintervals,caprate)*principal)));

capprice2Field.setText(numberFormatter1.format((new Prohiro().getPrice(forwardRates2,periodscovered,r,simulationintervals,caprate)*principal)));
  capprice4Field.setText(numberFormatter1.format((new Prohiro().getPrice(forwardRates3,periodscovered,r,simulationintervals,caprate)*principal)));

// Here we show the results graphically
// create dataset
XYSeriesCollection dataset = new XYSeriesCollection();

// create series
XYSeries forwardSeries = new XYSeries("Forward rate");
XYSeries caprateSeries = new XYSeries("Cap rate");
XYSeries simulationSeries = new XYSeries("Iterations=1");
XYSeries simulation2Series = new XYSeries("Iterations=I/2");
XYSeries simulation3Series = new XYSeries("Iterations=I/4");
double minResult = Double.POSITIVE_INFINITY;
double maxResult = Double.parseDouble(caprateField.getText());

// fill series
for (int i=0; i<size1; i++) {
    //forwardSeries.add(knownTimePoints[i], forwardRates[i]);
    caprateSeries.add(knownTimePoints1[i], Double.parseDouble(caprateField.getText()));
    simulationSeries.add(knownTimePoints1[i], forwardRates1[i]);
    simulation2Series.add(knownTimePoints1[i], forwardRates2[i]);
    simulation3Series.add(knownTimePoints1[i], forwardRates3[i]);

    if (forwardRates1[i]<minResult) minResult = forwardRates1[i];
    if (forwardRates1[i]>maxResult) maxResult = forwardRates1[i];
}

// add series to data set
dataset.addSeries(simulationSeries);
dataset.addSeries(simulation2Series);
dataset.addSeries(simulation3Series);
dataset.addSeries(caprateSeries);

JFreeChart chart = ChartFactory.createXYLineChart(
    //"Simulated Spot Rates/Cap (strike) rate"
    "ESTIMATED SPOT RATES",  // chart title
    "Time (years)",  // x axis label
    "Interest rate (%)",  // y axis label
    dataset, // data
    PlotOrientation.VERTICAL,
    true, // include legend
    true, // tooltips
    false // urls
);

// change y axis
XYPlot plot = (XYPlot) chart.getPlot();
ValueAxis rangeAxis = plot.getRangeAxis();
rangeAxis.setRange(minResult-0.2, maxResult+0.2);

// show graph
graphPanel.setChart(chart);
graphPanel.setVisible(true);

} // if add button
if (source == addButton) {
    // add line
    int rowNumber = forwardTable.getSelectedRow();
    if (rowNumber == -1) {
        JOptionPane.showMessageDialog(null,
            SELECT,
            ERROR,
            JOptionPane.ERROR_MESSAGE);
        return;
    } else {
        double date = ((Double)forwardModel.getValueAt(rowNumber,0)).doubleValue();
        double rate = ((Double)forwardModel.getValueAt(rowNumber,1)).doubleValue();
        Object[] row = {new Double(date+0.01),
            new Double(rate)};
        forwardModel.insertRow(++rowNumber,row);
if (source == deleteButton) {
    // delete line
    int rowNumber = forwardTable.getSelectedRow();
    if (rowNumber == -1) {
        JOptionPane.showMessageDialog(null,
                                      SELECT,
                                      ERROR,
                                      JOptionPane.ERROR_MESSAGE);
        return;
    } else {
        forwardModel.removeRow(rowNumber);
        return;
    }
}

// if focus is gained, do nothing
public void focusGained(FocusEvent e){
}

// if focus is lost, do something
public void focusLost(FocusEvent e){
    // find the source which called focus lost
    Object source = e.getSource();

    // if the source is numberOfiterations
    if (source == numberOfiterationsField){
        numberOfiterations=readInt(numberOfiterationsField,
                                  numberOfiterations,
                                  "Number of Iterations");
        return;
    }

    // if the source is the simulation intervals
    if (source == simulationIntervalsField){
        simulationIntervals=readInt(simulationIntervalsField,
                                 simulationIntervals,
                                 "Simulation Intervals Field");
        return;
    }

    // if the source is the periods covered
    if (source == periodsCoveredField){
        periodsCovered=readInt(periodsCoveredField,
                               periodsCovered,
                               "Periods covered field");
        return;
    }

    // if the source is the principal
    if (source == principalField){
        principal=readInt(principalField,
                     principal,
                     "Principal");
        return;
    }
//if the source is the sigmaone
if (source == sigmaoneField){
sigmaone=readPositive(sigmaoneField,
    sigmaone,
    "Sigma One Field");
    return;
}

//if the source is sigmatwo
if (source == sigmatwoField){
sigmatwo=readPositive(sigmatwoField,
    sigmatwo,
    "Sigma Two Field");
    return;
}

//if the source is alpahone
if (source == alphaoneField){
alpahone=readPositive(alphaoneField,
    alpahone,
    "Alpha One Field");
    return;
}

//if the source is alphatwo
if (source == alphatwoField){
alphatwo=readPositive(alphatwoField,
    alphatwo,
    "Alpha Two Field");
    return;
}

//if the source is cap rate
if (source == caprateField){
caprate=readPositive(caprateField,
    caprate,
    "Cap rate field");
    return;
}

//read positive double numbers
private double readPositive(JTextField field,
    double oldValue,
    String title)
{
    boolean isOK = true;
    double newValue = 1;
    try { //test input
        newValue = Double.parseDouble(field.getText());
    }
    catch (NumberFormatException e){//Error message
        JOptionPane.showMessageDialog(null,
            NOT_A_NUMBER,
            title,
            JOptionPane.ERROR_MESSAGE);
        isOK = false;
    }
    if (newValue <=0)//ERROR message
        JOptionPane.showMessageDialog(null,
            NOT_POSITIVE,
            title,
JOptionPane.ERROR_MESSAGE);  
isOk = false;
}  
if (isOk) {
    return newValue;
}  
else {
    field.setText(numberFormatter.format(oldValue));
    return oldValue;
}
}

//read double numbers  
private double readDouble(JTextField field,  
    double oldValue,  
    String title){
    boolean isOK = true;
    double newValue = 1;
    try {  // test input
        newValue = Double.parseDouble(field.getText());
    } catch (NumberFormatException e){  // ERROR message
        JOptionPane.showMessageDialog(null,  
            NOT_A_NUMBER,  
            title,  
            JOptionPane.ERROR_MESSAGE);
        isOK = false;
    }  
    if (isOk) {
        return newValue;
    }  
    else {
        field.setText(numberFormatter.format(oldValue));
        return oldValue;
    }
}

//Read integer numbers  
private int readInt(JTextField field,  
    int oldValue,  
    String title) {
    boolean isOK = true;
    int newValue = 1;
    try {  // test input
        newValue = Integer.parseInt(field.getText());
    } catch (NumberFormatException e){  // ERROR message
        JOptionPane.showMessageDialog(null,  
            NOT_INTEGER,  
            title,  
            JOptionPane.ERROR_MESSAGE);
        isOK = false;
    }  
    if (newValue <=0) {  //ERROR message
        JOptionPane.showMessageDialog(null,  
            NOT_POSITIVE,  
            title,  
            JOptionPane.ERROR_MESSAGE);
        isOK = false;
    }  
    if (isOk) {
        return newValue;
    }  
    else {  
        fieldValue.setText(numberFormatter.format(oldValue));
        return oldValue;
    }
}
```java
import java.util.Random;

public class Prohiro {

    // The class has two static variables
    public double[] knowntime;
    public double[] knownrate;

    // the class has one constructor
    public Prohiro() {
        int i;
        int p=0;
        int s=0;
        int r=0;
        int n=0;
        // number of days, we use the convention 1 year=360 days
        double deltat;
        double[] discreteforwardrate;
        double[] fhat;
    }
}
```

---

```java
// public static void main(String[] args){
//}
public void tableChanged(TableModelEvent e) {
    int row = e.getFirstRow();
    int column = e.getColumn();
}
}

/**
 * @(#) Prohiro.java  1.0 24/08/2007
 *
 * Copyright (c) 2007 Mälardalen University
 * Högskoleplan Box 883, 721 23 Västerås, Sweden.
 * All Rights Reserved.
 *
 * The copyright to the computer program(s) herein
 * is the property of Mälardalen University.
 * The program(s) may be used and/or copied only with
 * the written permission of Mälardalen University
 * or in accordance with the terms and conditions
 * stipulated in the agreement/contract under which
 * the program(s) have been supplied.
 *
 * Description: Class essential for the realization of Java Applet HJM
 * @version 1.0 Aug 07
 * @author Michail Kalavrezos
 * Mail: michail_kalavrezos@yahoo.se
 */
```
double[] taxi;
double[] fHatSpot;

// The class has three methods
// the following method creates an array of double numbers representing
// the simulated spot rates for the whole time period of the cap
public double[] getTimePoints(int periodscovered, int simulationintervals, int resetperiod)
    {
        this.p=periodscovered;
        this.s=simulationintervals;
        this.r=resetperiod;
        double[] timePoints=new double[p*s+2];
        // the simulation interval's length, we assume that we have at
        // least one caplet
        double deltati=r/s;
        int l;
        for (l=0;l<p*s+2;l++)
            // this is the array of time points expressed in years that
            // we plot
            timePoints[l]=(l*deltati)/360;
        }

    return (timePoints);
}

// this method returns the simulated spot rates for the whole time
// period of the cap
public double[] getfHatSpot(int periodscovered, int simulationintervals, int resetperiod,
    double sigmaone, double sigmatwo, double alphasone,
    double alphatwo, int numberofiterations,
    double[] knownTimePoints, double[] forwardRates )
    {
        this.n=numberofiterations;
        this.p=periodscovered;
        this.s=simulationintervals;
        this.r=resetperiod;
        int l=0;
        this.deltati=r/s;
        double[] matchedforward=new double[(s*p)+2];
        double[] moments=new double[s*p+2];
        fHatSpot=new double[(s*p)+1];

        discreteforwardrate=new double[s*p+1];

        // Spline interpolation
        // We have points knownTimePoints[i] on x axis
        // and points forwardRates[i] on y axis
        // first, we estimate the derivative at the point standardTimes[0]
        double ypFirst = (forwardRates[1]-forwardRates[0])/(knownTimePoints[1]-knownTimePoints[0]);
        // create spline object
        Spline1 spline = new Spline1(knownTimePoints, forwardRates, ypFirst, 0.0);
        // fill arrays
        discreteforwardrate[0] = forwardRates[0];
        for (int i=1; i<s*p+1; i++) {
moments[i] = i*deltati/360;

discreteforwardrate[i] = spline.splint(moments[i]);
}

//create an array of the discrete forward rate from the matched forward rate
//at the simulation points
fHatSpot[0]=discreteforwardrate[0];

//create the arrays where the sigmas are going to be stored(or maybe locally!!!!)
double[][] sigmaonehat=new double[s*p+1][s*p+1];
sigmaonehat[0][0]=0.00;
double[][] sigmatwohat=new double[s*p+1][s*p+1];
sigmatwohat [0][0]=0.00;
double summaone=0.00;
double summatwo=0.00;
double[][] miouonehat=new double[s*p+1][s*p+1];
double[][] mioutwohat=new double[s*p+1][s*p+1];
taxi=new double[s*p+1];

//calculate the sigmas
for (i=1;i<s*p+1;i++){

    Random generator=new Random();
    double gen=generator.nextGaussian();
    int k=0;
    for ( k=i;k<s*p+1;k++){
        sigmaonehat[i-1][k]=sigmaone*(Math.exp(-alphaone*(k-i+1)*deltati/360));
        summaone=summaone+sigmaonehat[i-1][k];

        miouonehat[i-1][k]=((deltati/360)/2)*((Math.pow(summaone,2))-
        (1/2)*(Math.pow((summaone-sigmaonehat[i-1][k]),2)));

        sigmatwohat[i-1][k]=sigmatwo*(Math.exp(-alphatwo*(k-i+1)*deltati/360));
        summatwo=summatwo+sigmatwohat[i-1][k];

        mioutwohat[i-1][k]=((deltati/360)/2)*((Math.pow(summatwo,2))-(1/2)*
        (Math.pow((summatwo-sigmatwohat[i-1][k]),2)));
    }

    Random generator=new Random();
    double gen=generator.nextGaussian();
    int k=0;
    for ( k=i;k<s*p+1;k++){
        sigmaonehat[i-1][k]=sigmaone*(Math.exp(-alphaone*(k-i+1)*deltati/360));
        summaone=summaone+sigmaonehat[i-1][k];

        miouonehat[i-1][k]=((deltati/360)/2)*((Math.pow(summaone,2))-
        (1/2)*(Math.pow((summaone-sigmaonehat[i-1][k]),2)));

        sigmatwohat[i-1][k]=sigmatwo*(Math.exp(-alphatwo*(k-i+1)*deltati/360));
        summatwo=summatwo+sigmatwohat[i-1][k];

        mioutwohat[i-1][k]=((deltati/360)/2)*((Math.pow(summatwo,2))-(1/2)*
        (Math.pow((summatwo-sigmatwohat[i-1][k]),2)));
    }

    fhatsim= new double[M-i];
    fhatsim[0]=discreteforwardrate[k]+#
        (deltati/360)*miouonehat[i-1][k]+mioutwohat[i-1][k]...
double kr=0.00;
for (int d=1;d<n+1;d++){
    fhatsim[k-i]=discreteforwardrate[k]+(deltati/360)*(miouonehat[i-1][k]+mioutwohat[i-1][k])+(Math.sqrt(deltati/360)="/Migmaonehat[i-1][k]*Math.random()+sigmatwohat[i-1][k]*gen));
    kr+=fhatsim[k-i];
}
//taxi[] is the array whose values are recalculated
//within each simulation loop

taxi[k]=((double)kr)/n;
}

else if(k==i&&k>1){
double tr=0.00;
for (int d=1;d<n+1;d++){
    fhatsim[0]=taxi[k]+(deltati/360)*(miouonehat[i-1][k]+mioutwohat[i-1][k])+(Math.sqrt(deltati/360)="/Migmaonehat[i-1][k]*Math.random()+sigmatwohat[i-1][k]*gen));
    //tr=tr+fhatsim[0];
    //fhatsim[0]+=fhatsim[0];
    tr+=fhatsim[0];
}
fHatSpot[i]=((double)tr)/n;
}

else if(k>i&&i>1) {
double kr=0.00;
for (int d=1;d<n+1;d++){
    fhatsim[k-i]=taxi[k]+(deltati/360)*(miouonehat[i-1][k]+mioutwohat[i-1][k])+(Math.sqrt(deltati/360)="/Migmaonehat[i-1][k]*Math.random()+sigmatwohat[i-1][k]*gen));
    kr+=fhatsim[k-i];
}
taxi[k]=((double)kr)/n;
}
else{
return (fHatSpot);
}

//the following method returns the price of the cap with principal=1
public double getPrice(double[] fHatSpot, int periodscovered, int resetperiod, int simulationintervals, double caprate) {

double c=caprate;
int r=resetperiod;
this. p=periodscovered;
this. s=simulationintervals;
this. deltati=r/s;
double PV=0.00;
double[] presentValue=new double[p+1];
presentValue[0]=0.00;
double[] D=new double[p+1];
D[0]=0.00;
double[] Fhat=new double[p+1];
Fhat[0]=fHatSpot[0]/100;

int i=0;
for (i=1;i<p+1;i++) {
    //the following statement assigns the simulated spot rate on the reset day to the array Fhat[i]
    Fhat[i]=fHatSpot[i*s]/100;

    //we need to create a double variable to express the sum of the simulated spot rates between two reset days
    double sumOfSpot=0.00;
    double[] P=new double[p+1];

    //A loop to add the spots, for each caplet the number of spots is equal to (no of caplet*simulation intervals)
    //in other words this is the discount factor for each (discounted at the maturity) caplet payment
    for (int k=0;k<(i*s);k++) {
        sumOfSpot=sumOfSpot+fHatSpot[k];
    }
    P[i]=1*((double) r)/360*(Fhat[i]-c*0.01)/(1+((double) Fhat[i]*r)/360);
    D[i]=Math.exp(-(sumOfSpot/100)*deltati/360);
    //this is the present value at current time zero
    presentValue[i]=Math.max((P[i]*D[i]),0);
    PV=PV+presentValue[i];
}

return (PV);
}