UNIVERSITÀ DI SIENA
1240

Dipartimento di Economia politica e statistica

Corso di Laurea in
MSC IN FINANCE

Pricing and Hedging the Bermudan Callability in a negative rates Environment: An implementation of the Longstaff and Schwartz approach

Relatore:
Prof. Antonio Mannolini
Correlatore:
Prof. Claudio Pacati

Candidato:
Andrea Carapelli

Anno Accademico 2018-2019

Acknowledgments

Un ringraziamento speciale in primo luogo va ai miei genitori che mi hanno sempre sostenuto standomi accanto sia nei momenti buoni che in quelli meno buoni, sempre pronti a dare la loro opinione e ad accettare le mie scelte.
Grazie Elena, per le lunghe conversazioni, per quando sono giù di morale e mi ricordi cos'è quello che voglio, sei una persona più unica che rara.
Grazie al mio relatore Antonio Mannolini e al prof. Pacati per i preziosi insegnamenti e per avermi dato la possibilità di imparare da veri esperti del settore.
Infine un ringraziamento alla persona che più mi ha aiutato nella stesura della tesi, il mio amico/tutor Giuseppe Trapani, tutte le cose che mi hai insegnato in questi mesi non sono quantificabili, quello che è stato l'inizio di una crescita professionale lo devo principalmente a te, ad Antonio e a tutti gli altri membri del desk Interest Rate Derivatives, Marco, Martina, Luigi, Elisa ed Eros.

Pricing and Hedging the Bermudan Callability in a negative rates Environment: An implementation of the Longstaff and Schwartz approach

Carapelli Andrea

April 2020

Abstract

Pricing the *callability* of Bermudan options concerns the evaluation of an optimal stopping strategy, this thesis presents a step-by-step implementation of the least squares Monte Carlo algorithm for Bermudan swaptions in Python. Starting from the work of Longstaff and Schwartz [5] I expose the challenges of pricing and hedging interest rate exotic derivatives in the European market using one factor and two factor Gaussian short rate models.

Key Words: Interest rate Derivatives, Bermudan Swaptions, Gaussian short rate Models, Least squares Monte Carlo.

Contents

Introduction

Fixed income derivatives are among the most traded financial derivatives around the world, banks and financial companies typically use those instruments to neutralize interest rates risk coming from all the contracts made with corporations, institutional investors and retailers.

Asset Class	**Cash**	**Derivatives**	**OTC**
Long Term	Fixed rate bond Floating rate bond	Bond future Bond future option	Swaps, Swaptions Caps, Floors Cross Currency Swaps Exotics
Short Term	Deposit/Loan	I.R. Future	FRA FX Swap EuroDollar Futures

The breakdown caused by the last decade financial crisis entailed significant changes to all financial and economic realities. During those years several efforts has been made both by the regulator and the policy maker to overcome further system instability. Rebuilding a solid credit system to restore trust in financial and banking sector has been the first necessity in a new reality where banks were no longer considered risk free agents. To this end, many changes has been made in terms of capital requirements, collateralized contracts and monetary policy.

In the European market all of the post crisis measures nowadays turned into a *new normality* characterized by negative interest rates with tighter regulation and structural variations of previously existing market practices; banks and financial institutions are currently facing this strict regime that on one hand ensures system stability and on the other pushes all sector to a modernization with resized and diversified business units.

Moving to the impact on pricing, the events generated by the crisis triggered visible effects on the interest rate market; financial practitioners had to replace the classical single-curve approach as it was not suited anymore to capture the non-negligible basis spreads arisen between different indexes. Furthermore additional consideration has been made for pricing techniques and structural models in presence of negative interest rates, for example the hypothesis of log-normality of interest rates assumed by benchmark models (e.i. Black) did not allow the presence of negative values, as a consequence financial practitioners developed new pricing frameworks and market models in order to account for the innovated OTC markets peculiarities.

The subject of my thesis consists in the application in the European market of the least squares Monte Carlo algorithm presented by Longstaff and Schwartz [5] for the estimation of price and sensitivities of interest rate Bermudan swaption. Starting from some fundamental concepts I expose all the details needed for the implementation of the methodology.

Dealing with interest rates securities typically means dealing with periodic payments, hence the initial part of this work focuses on the foundations of cash-flow analysis, summing up the conventions needed for the discounting and the forwarding of fixed income derivatives. The remaining part of the first chapter is dedicated to some of the concepts at the foundations of interest rate market: contracts formulae in the "modern" multiple curves framework with a small study on the impact that different interpolation functions produce on the bootstrapping quality.

In the second chapter I present the short rate models theory needed to set up a future yield curve simulation, addressing some issues of the one factor and the two factor Gaussian models. Collecting some results of Andersen and Piterbarg [1], Peter Caspers [15], Brigo and Mercurio [4] I present all the formulae implemented in the Python code appendix and in the QuantLib classes. Then in the last section of the chapter I show pros and cons of working with the one factor and the two factor processes under a different probability measure discussing the property of exact Monte Carlo simulation of short rate models.

The third chapter is about the description of the least squares Monte Carlo methodology presented by Longstaff and Schwartz in their paper of 2001. This section collects some theoretical notions of Glasserman [3][7] used for the implementation of the algorithm and the analysis of the optimal exercise valuation with approximated continuation values in a dynamic programming recursion.

The last part of the thesis focuses on the implementation of the least squares Monte Carlo simulation for the estimation of the price of a Bermudan swaption on Euro market. The results are obtained combining all the theoretical notions presented in the previous chapters, starting from models calibration to swaptions volatility cube and showing the results of the simulation under the one factor short rate model and the G2++.

At the end of the document I present some considerations on the difficulty of managing first order risks of exotic contracts like Bermudan swaptions, taking into consideration model risk and coherence of volatility hedging in presence of smile.

Chapter 1

Interest Rate Derivatives

1.1 Stochastic Discount Factor

General asset pricing theory assumes that on a market composed by different possible states where investors decides periodically how much to save and how much to consume, interest rates are strictly linked to expected future marginal utility. The equilibrium between the marginal utility loss of consuming less today and the marginal utility gain coming from future payoffs gives us the basic asset pricing equation

$$P(t) = \mathbb{E}_t[m_{t+1}x_{t+1}].$$

The asset price P_t must be equal to the expected discounted value of future payoff x_{t+1}, where m_{t+1} is the stochastic discount factor depending on marginal utility. Assume to model the utility function of investors in terms of present and future wealth c:

$$U(c_t, c_{t+1}) = u(c_t) + \beta\mathbb{E}_t[u(c_{t+1})],$$

$$u(c_t) = \frac{c_t^{1-\gamma}}{1-\gamma};$$

with $u(c_t)$ being the utility function at time t and γ and β being the risk aversion and impatience coefficients.

The first order condition for an optimal portfolio allocation computed for the above equation gives us the general pricing formula

$$P(t) = \mathbb{E}_t \left[\beta \frac{u'(c_{t+1})}{u'(c_t)} x_{t+1} \right];$$

with the discount factor expressed in terms of marginal utility

$$m_{t+1} = \beta \frac{u'(c_{t+1})}{u'(c_t)}.$$

The above equation gives us the price of the asset, prior the assumption that we know the future payoff x_{t+1} and that we are able to estimate the marginal utility of investors. If there is no uncertainty about the marginal rate of substitution for present and future wealth/consumption we can rewrite the pricing formula defining a risk-free rate

$$P(t) = \frac{1}{R^f} \mathbb{E}[x_{t+1}];$$

where $\frac{1}{R^f}$ represent the risk-free discount factor.

As a direct implication of this generalization we are allowed to incorporate all risk correction into a single discount factor. This implies some important results: each different asset is correlated in a different way with respect to the common discount factor, correlation tells us that m must be considered a random variable but also that we have the possibility to make asset-specific risk correction and still be able to respect the no arbitrage theorem.

1.1.1 The Money Market Account

Knowing investors' utility function is the strong assumption of general asset pricing theory that is needed to represent discount factors in terms of marginal utility; in real world we cannot measure properly individuals' utility functions but we need to account for financial variables as they are the closest estimators of expected marginal utility growth and discount factors.

When it comes to the evaluation of assets like interest rate derivatives the risk-free rate is expressed through the money market account, which represents an investment that is accruing interest continuously at the risk-free rate prevailing in the market at every instant.

$$dB(t) = r_t B(t) dt,$$

with $B(t)$ being the value of the money market account at time t and r_t instantaneous rate (short rate) at which the bank account grows.

$$B(t) = B(0)\exp\left(\int_0^t r_s ds\right),$$
$$B(0) = 1.$$

The money market account represents a *locally* risk-free investment, the absence of volatility implies that we have complete knowledge of the return by observing the prevailing short rate r at time t.

The definition of money market account is fundamental for relating amounts of currencies available at different times, using B as numeraire allow us to express payoffs in terms of quantity of bank account and to define a discount factor (possibly stochastic) D between t and T as

$$D(t,T) = \frac{B(t)}{B(T)} = \exp\left(-\int_t^T r_s ds\right),$$

where r_s can be a deterministic function of time, in that case $D(t,T)$ will be the price of the bond, or modeled as a stochastic process in order to embody interest rates' volatility.

1.2 Term Structure of Interest Rates

Interest rate modeling for pricing of derivatives and for risk management requires a prior correct estimation of the yield curve dynamics. Starting from the few liquid instruments that are directly quoted in the market, financial and economic practitioners have developed different methodologies to extract a smooth and continuous yield curve from the implied rates.

The market practice in this field has changed after the crisis of 2007; before 2007 banks were considered risk-free institutions, this means that neither liquidity nor credit issues were included in pricing theory. Financial modeling relied on this old view of banking institutions as it was expressed through the single curve framework, with inter-bank overnight rates (LIBOR/EURIBOR) considered proxies for risk free investments.

The classic single curve approach is not suited to take into account the basis spread arisen between inter-bank rates after 2007, it does not consider collateralization of OTC derivatives or any liquidity risk implied in inter-bank credit transfers. The single curve was used both for discouting and forwarding cash flows and was built according to the principle of ”maximum liquidity”, it was common practice to use simple compounding deposits for maturities spanning from one day (overnight) to three months, FRA rates from 3 months to 2 years and swap rates for longer maturities. However this approach is no longer valid nowadays, building a yield curve using instruments indexed to different underlying rate tenors would lead to inconsistencies and non-homogeneous results.

The modern approach, or multiple curve approach, has evolved to take into account the basis spread of inter bank rates and the funding requirements of different OTC derivatives. The new practice relies on building a unique discount curve and multiple forwarding curves created out of homogeneous instruments having the same underlying rate tenor.

In order to build a correct yield curve it is important to define some basic concepts first, starting from the compounding at which our investment accrues and the different types of interest rate.

- **Simple compounding:**

$$L(t,T) = \frac{1 - P(t,T)}{\tau(t,T)P(t,T)}$$

- **Periodic compounding**

$$Y^k(t,T) = \frac{k}{P(t,T)^{\frac{1}{k\tau(t,T)}}} - k$$

with k = 1 we obtain the annually compounded spot interest rate:

$$Y(t,T) = \frac{1}{P(t,T)^{\frac{1}{\tau(t,T)}}} - 1$$

then we can obtain the continuously compounded rate as the limit for k going to ∞

$$\lim_{n\to\infty} \frac{k}{P(t,T)^{\frac{1}{k\tau(t,T)}}} - k = -\frac{ln(P(t,T))}{\tau(t,T)}$$

- **Continuous compounding**

$$R(t,T) = -\frac{lnP(t,T)}{\tau(t,T)}$$

from which we can derive the bond-price

$$P(t,T) = e^{-R(t,T)\tau(t,T)}$$

Then starting from the definition of forward rate agreement (FRA) we can define the forward rate $F(t,T,S)$ with $t \leq T \leq S$ as the fixed rate that makes the contract fair at inception.

$$\mathbf{FRA}(t,T,S) = \mathbb{E}^{\mathbb{Q}}\big[D(t,S)(L(T,S)-K)(S-T)\big]$$

By absence of arbitrage we obtain the following formula of the value of the FRA at time t

$$\mathbf{FRA}(t,T,S,\tau,N,K) = N\big[P(t,S)\tau(T,S)K - P(t,T) + P(t,S)\big]$$

- **Simply compounded forward interest rate**
The value of the fixed rate K which makes the value of the contract equal to zero at inception, on a unit notional, is the simply compounded forward rate

$$F(t,T,S) = \frac{1}{\tau(T,S)}\left[\frac{P(t,T)}{P(t,S)-1}\right]$$

- **Instantaneous forward interest rate**
The instantaneous forward interest rate $f(t,T)$ is a forward rate with maturity S very close to the fixing date T

$$f(t,T) = f(t,T,T+\Delta T) - \frac{\partial lnP(t,T)}{\partial T}$$

with ΔT small, and we can express the zero coupon bond price function as

$$P(t,T) = e^{-\int_t^T f(t,u)du}$$

1.2.1 Yield Curve Construction

In the following sections I will reproduce the techniques used by Ametrano and Bianchetti[11][13] to bootstrap a smooth and continuous yield curve. The bootstrapping procedure is performed according to *exact-fit* principle: the yield curve fitted on a given set of market quotes must reprice correctly such liquid instruments. The results are obtained applying a natural cubic spline interpolation function to a logarithmic scale of interest rates with underlying EONIA and EURIBOR 6M.

EONIA curve is built starting from few Overnight Indexed Swaps (OIS) market quotes, OIS interest rates are the rate at which the collateral/margin is remunerated, their floating leg is tied to overnight rates and thus they are considered proxies of risk free.

$$\mathbf{OIS}(t,T,S,R_{on},K) = N[R_{on}(t,T,S) - K]A(t,S)$$
$$A(t,S) = \sum_{j=1}^{m} P(t,S_j)\tau(S_{j-1}.S_j)$$

with $R_{on}^{OIS}(t,T,S)$ being the equilibrium rate

$$R_{on}^{OIS}(t,T,S) = \frac{\sum_{i=1}^{n} P(t,T_i)R_{on}\tau(T_{i-1},T_i)}{A(t,S)} = \frac{P(t,T_0) - P(t,T_n)}{A(t,S)}$$

This condition holds under the assumption of perfect collateralization, in this case the floating leg value simplifies to the single curve expression. [1]

The first two instruments considered are the over-night and the spot-next deposits, then I used OIS quotes from maturity one week to 30 years. Another approach could be to use OIS quotes with maturities fixed on ECB dates in order to include and capture market expectations of ECB forward guidance. All market quotes, observed on 6 February 2020, are reported in the appendix in table C.1

[1]see Ametrano and Bianchetti [11][13]

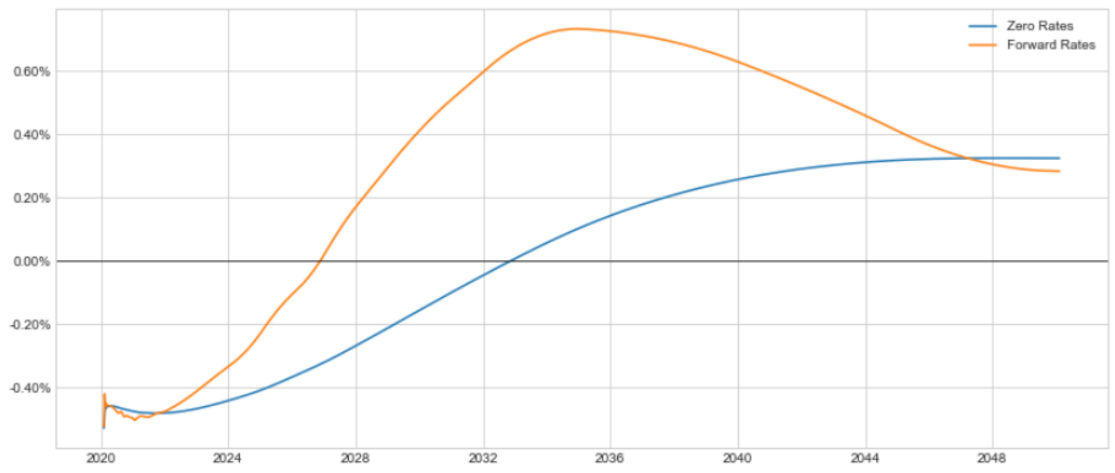


Figure 1.1: EONIA Yield Curve

The EURIBOR 6M forwarding curve is bootstrapped starting from a 6-months deposit that remunerates the fixing rate for EURIBOR 6M index, it is a good practice to extrapolate synthetic deposits quotes to be used for tenors less than 6 months spanning the initial part of the curve. Successively there are 12 FRA quotes with tenor 6 months, and then SWAP rates from 2Y to 50Y maturity

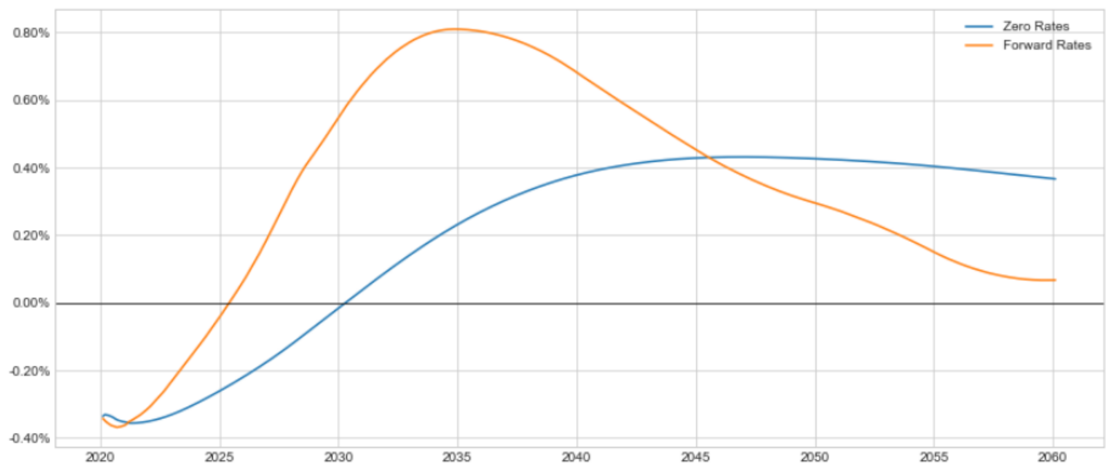


Figure 1.2: EUR:6M Yield Curve

As we could expect EURIBOR 6M curve presents higher yields with respect to OIS, if we compare forward rates with tenor 6M computed on the two curves the risk premium implied in the EURIBOR curve just gets evident

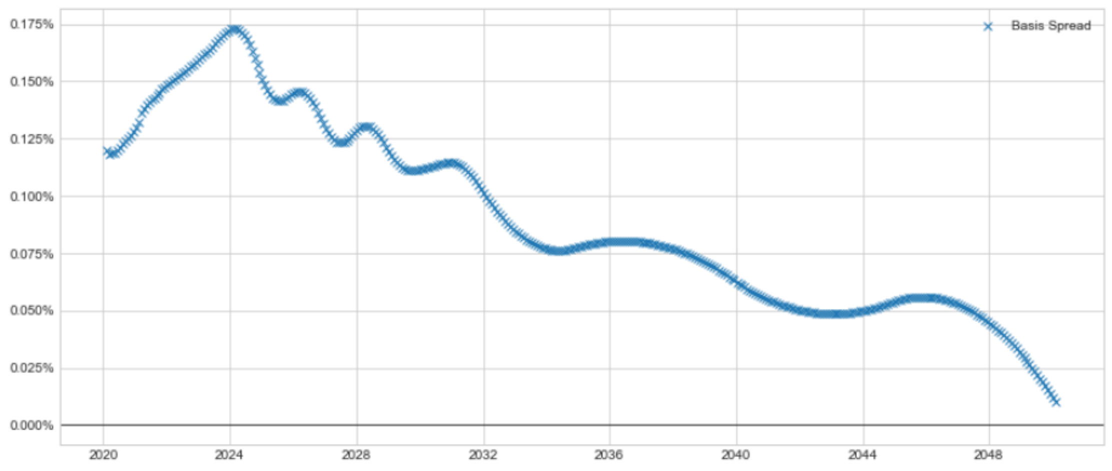
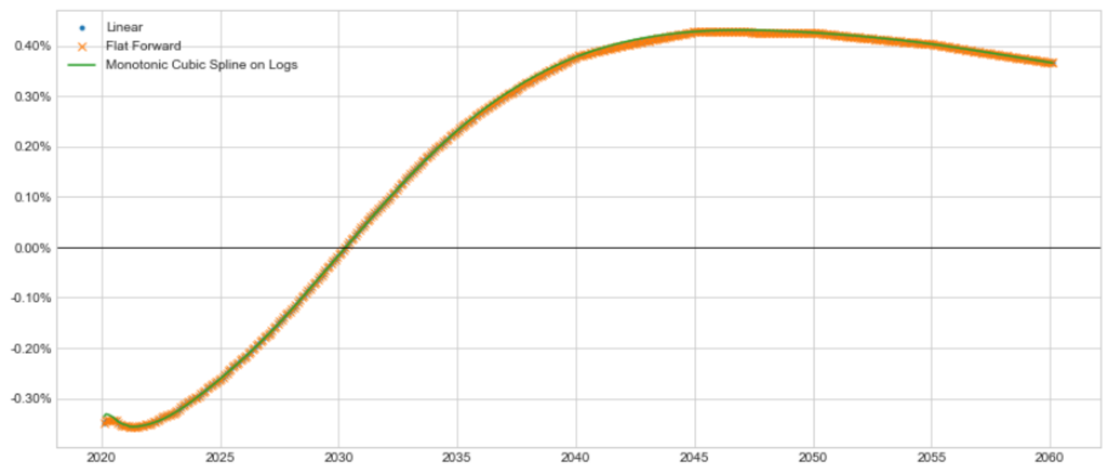


Figure 1.3: Basis Spread EUR:6M/EONIA

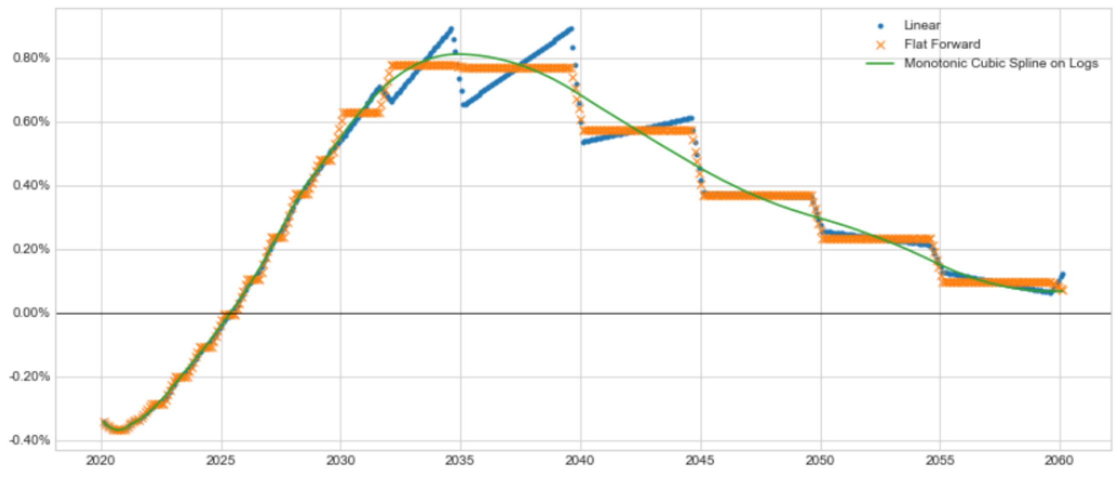
1.2.2 Interpolation Functions

The choice of the interpolation function has an high importance in determining the smoothness of our term structure. The study of bootstrapping algorithms and interpolation functions belongs to a whole specialized field in quantitative finance and economics, my purpose here is to reproduce some popular and widely used interpolation schemes taking as a reference the study of Ametrano and Bianchetti[11].

The first figure in (1.4) is an example of how different interpolation functions do not affect the quality of the zero rates curve, however, we can see that applying a linear or a flat interpolation function leads to an inconsistent forward rates term structure.



(a) EUR:6M zero rates



(b) consequences on forward rates

Figure 1.4

The construction of a smooth term structure for forward rates seems to be the sound principle in financial markets, starting from a careful selection of the initial liquid instruments, most of the yield curve construction algorithms are designed to capture different special situations like periods of higher liquidity demand (turn of the year effect) or market expectation for Central Banks decisions.

1.3 Pricing Under Different Measures

By definition of absence of arbitrage opportunities, the basic fundamental theorem of asset pricing, it is impossible to invest zero today and receive a positive amount tomorrow, this directly implies the law of one price: two portfolios having the same payoffs in the same states of the world must have the same price; all financial variables are modeled on the basis of those two theorems[2]. Consider a financial market on a time horizon $[0, T]$ with a finite number n of traded assets, assuming that the market is complete and arbitrage-free under the risk neutral measure $\mathbb{Q}$ the price of each traded assets $X_i(t)$ can be expressed through the stochastic process

$$dX_i(t) = \mu_i(t, X_i(t))dt + \sigma_i(t, X_i(t))dW_i^{\mathbb{Q}}(t)$$

with μ_i and σ_i being respectively the drift and diffusion coefficient of asset i and $dW_i^{\mathbb{Q}}$ representing the risk neutral dynamics of a Brownian motion

For financial modeling considerations it is always possible to express prices in terms of the risk-neutral numeraire, that is the bank account described earlier. However, in some cases it can be helpful to normalize prices in terms of other numeraires in order to ease computations. A numeraire denotes the unit of account in which other assets are denominated, it must be strictly positive and cannot pay dividends. For a more formal and wider explanation about probability measures see Appendix B.

Once defined the numeraire as any positive non-dividend paying asset, the selection of an equivalent martingale measure is basically a matter of choosing a precise numeraire to normalize the price of all traded securities.

Define N as the numeraire and $\mathbb{Q}^N$ as the probability measure associated with N, by no arbitrage assumption the price of a given security X deflated by N is a

[2]See Appendix B

martingale under the measure associated with that numeraire $\mathbb{Q}^N$.

$$X(t) = \mathbb{E}^{\mathbb{Q}^N}\left[\frac{N(t)}{N(T)}X(T)\right]$$
$$X(t) = N(t)\mathbb{E}^{\mathbb{Q}^N}\left[\frac{X(T)}{N(T)}\right]$$

Now let M be another numeraire associated with the probability measure $\mathbb{Q}^M$, this new probability measure is an equivalent martingale measure with respect to $\mathbb{Q}^N$

$$V(t) = M(t)\mathbb{E}^{\mathbb{Q}^M}\left[\frac{X(T)}{M(T)}\right] = N(t)\mathbb{E}^{\mathbb{Q}^N}\left[\frac{X(T)}{N(T)}\right]$$
$$V(t) = \mathbb{E}^{\mathbb{Q}^M}\left[\frac{X(T)}{M(T)}\right] = \mathbb{E}^{\mathbb{Q}^N}\left[\frac{X(T)}{M(T)}\frac{M(T)/M(t)}{N(T)/N(t)}\right]$$

Under the risk-neutral measure $\mathbb{Q}$ the numeraire is represented by the money market account $B(t)$ defined, the process $X(t)$ deflated by B is a martingale

$$X(t) = \mathbb{E}^{\mathbb{Q}}\left[\frac{B(t)}{B(T)}X(T)\right]$$
$$X(t) = \mathbb{E}_t^{\mathbb{Q}}\left[e^{-\int_t^T r(u)du}X(T)\right]$$

The T-Forward Measure

Under the T-Forward measure the numeraire asset is the T-maturity zero-coupon bond, such that

$$\frac{V(t)}{P(t,T)} = \mathbb{E}_t^{\mathbb{T}}\left[\frac{V(T)}{P(T,T)}\right]$$

as $P(T,T) = 1$ the equation simplifies

$$V(t) = P(t,T)\mathbb{E}_t^{\mathbb{T}}[V(T)]$$

The Swap Measure

The swap measure $Q^{\alpha,\beta}$ is induced by the annuity numeraire $A_{\alpha,\beta}$, considered as a linear combination of zero-coupon bonds.

$$A_{\alpha,\beta}(t) = \sum_{i=\alpha+1}^{\beta}\tau_i P(t,T_i)$$

Under the measure $\mathbb{Q}^{\alpha,\beta}$ induced by the annuity numeraire, the forward swap rate $S_{\alpha,\beta}$ is a martingale

$$S_{\alpha,\beta}(t) = \frac{P(t,T_\alpha) - P(t,T_{\alpha+\beta})}{A_{\alpha,\beta}(t)}$$

1.3.1 Interest Rate Swaps and Swaptions

An interest rate swap is a linear derivative that allows two counterparties to exchange cash flows, the most generic vanilla swap denotes an exchange of fixed for floating payments or vice-versa.

Using the notation of Mercurio[16] let's consider a floating leg linked to the LIBOR rate fixed in advance at T_{k-1} with $k = a, \dots, b$. and payed in arrears at each time $T_a, \dots, T_b$. At time t the value of the floating leg is equal to the discounted expectation of future LIBOR fixing rates. For a single payment:

$$\mathbf{FL}(t, T_{k-1}, T_k) = \tau_k P(t, T_k)\mathbb{E}^{T_k}[L(T_{k-1}, T_k)]$$

Under the forward measure $\mathbb{Q}^{T_k}$ the expected value of future LIBOR rate is the forward rate observed in t, fixed in T_{k-1}, to be exchanged in T_k

$$\mathbf{FL}(t, T_{k-1}, T_k) = \tau_k P(t, T_k)F(t, T_{k-1}, T_k)$$

The floating leg net present value is obtained by summing the value of all flows

$$\mathbf{FL}(t, T_a, \dots, T_b) = \sum_{k=a+1}^{b} \tau_k P(t, T_k)F(t, T_{k-1}, T_k)$$

Now consider the swap's fixed leg with K being the fixed rate paid on each payment date $T_c. \dots, T_d.$, the net present value is equal to the discounted future cash flows

$$\mathbf{FIX}(t, T_c, \dots, T_d) = K \sum_{j=c+1}^{d} \tau_j P(t, T_j)$$

Now we can exploit the price of a payer swap as the difference between the floating leg and the fixed leg

$$\mathbf{IRS}(t, T_a, \dots, T_b, T_c, \dots, T_d) = \sum_{k=a+1}^{b} \tau_k P(t, T_k)F(t, T_{k-1}, T_k) - K \sum_{j=c+1}^{d} \tau_j P(t, T_j)$$

with K being the fair rate of the swap that makes the value of the swap equal to zero at inception

$$S_{a,b,c,d}(t) = \frac{\sum_{k=a+1}^{b} \tau_k P(t,T_k) F(t,T_{k-1},T_k)}{\sum_{j=c+1}^{d} \tau_j P(t,T_j)} \tag{1.1}$$

European Swaptions

A European Swaption gives the holder the right, but not the obligation, to enter in a swap with a given tenor at a certain maturity. Consider an option with maturity T_a to enter in a payer swap with payment times given by $T_{a+1},\dots,T_b$ for the floating leg and $T_{c+1},\dots,T_d$ for the fixed leg, with $T_a = T_c$ and $T_b = T_d$, the payoff of the swaption at T_a is

$$[S_{a,b,c,d}(T_a) - K]^+ \sum_{j=c+1}^{d} \tau_j P(T_c,T_j)$$

Where $S_{a,b,c,d}(T_a) - K$ is the difference between the swap rate defined above and the fixed rate considered as the strike of the swaption. The second term represent the annuity factor, defined as a linear combination of discount factors

$$A^{c,d}(t) = \sum_{j=c+1}^{d} \tau_j P(T_c,T_j)$$

It is convenient to take the expected value of the payoff under the measure $\mathbb{Q}^{c,d}$ induced by the annuity numeraire $A^{c,d}(t)$

$$\begin{aligned}
&\mathbf{PS}(t,T_{a+1},\dots,T_b,T_{c+1},\dots,T_d) \\
&= \sum_{j=c+1}^{d} \tau_j P(t,T_j) \mathbb{E}^{\mathbb{Q}^{c,d}} \left\{ \frac{[S_{a,b,c,d}(T_a) - K]^+ \sum_{j=c+1}^{d} \tau_j P(T_c,T_j)}{A^{c,d}(T_c)} \right\} \\
&= \sum_{j=c+1}^{d} \tau_j P(t,T_j) \mathbb{E}^{\mathbb{Q}^{c,d}} \left\{ [S_{a,b,c,d}(T_a) - K]^+ \right\}
\end{aligned}$$

since the forward swap rate $S_{a,b,c,d}$ is a martingale under $\mathbb{Q}^{c,d}$. A payer European swaption is a call option on the forward swap rate and a receiver European swaption is a put option on the same underlying, hence it's a market benchmark to price European swaptions using the Black-76 formula

$$\mathbf{PS}(t, T_{a+1}, \ldots, T_b, T_{c+1}, \ldots, T_d) = \sum_{j=c+1}^{d} \tau_j P(T_c, T_j)\mathbf{Black}(K, S_{a,b,c,d}(t), \sigma_{a,b,c,d}\sqrt{T_a - t})$$

where $\sigma_{a,b,c,d}$ is the Black volatility implied in the market.

Bermudan Swaptions

A Bermudan swaption is characterized by three dates $T_k < T_h < T_n$ and gives the holder the right to enter into a swap on a given set of exercise dates between T_k and T_h. Once the option is exercised at time T_{ex} the holder enters into a swap with first fixing date T_{ex} and maturity T_n, hence the tenor of the swap depends on the time at which the option is exercised.

In order to make a correct estimation of this exotic derivative we must take into account its non-linear nature, since we are dealing with multiple possible payoff it's not sufficient to account only for future discounted cash flows, at each different Bermudan date there is a different probability to exercise the option and a different value for the underlying swap.

The problem of choosing the optimal exercise date is one of the most challenging numerical problems in finance, at each exercise date the option-holder decides whether exercising rather than holding the option until next date, comparing the value of the swap at T_{ex} with the intrinsic value of the option

$$\mathbf{BS}(T_{ex}) = max\big[\mathbf{IRS}(T_{ex}, T_n)\ \ ,\ \ \mathbf{I}(T_{ex}, T_h)\big]$$

with T_{ex} exercise date, T_h maturity of the option and T_n maturity of the swap.

The non-linearity of Bermudan-swaptions implies that there is no possibility to estimate their price in closed-formula. Despite their complexity, Bermudan swaptions are the most liquid instruments among exotic fixed income securities, as they represent the correct solution for hedging callable instruments. The choice of a market model able to calibrate on a rich set of quoted instruments is the first challenge of exotic Libor derivatives. Now assume to that we have a model calibrated on a complex volatility structure that efficiently explain most of the market prices grid, then the subsequent step is to use a pricing technique that is efficient both for price valuation and time execution.

The forward looking nature of Monte Carlo simulation is ideal when it comes to estimate path dependent securities, however the method is not suited when it comes to address an optimal stopping problem. What drives a stopping decision is basically given by the comparison of the exercise value with the remaining value of the option, then a successful pricing of Bermudan swaptions in pure Monte Carlo simulations requires the estimation of the most valuable swap by "brute force". Applications of "brute force" nested Monte Carlo represents a quite slow solution, the least squares Monte Carlo algorithm presented by Longstaff and Schwartz[5] and implemented for interest rate Bermudan swaptions in the next chapters combine Monte Carlo simulation to a dynamic programming formulation of the problem with good results in terms of speed and accuracy.

Chapter 2

Gaussian Short Rate Models

2.1 One Factor Short Rate

This chapter collects some results of Andersen and Piterbarg [1], Brigo and Mercurio [4], including the explicit formulae implemented in the QuantLib gaussian1d class (Peter Caspers [15]) and the formulae implemented in my code for the simulation of the Gaussian short rate model (GSR) and the G2++.

The general one factor Gaussian short rate dynamics are given by

$$dr(t) = \kappa(\vartheta(t) - r(t))dt + \sigma_r(t)dW(t);$$

with κ and σ_r piecewise constant under the risk neutral measure and ϑ that matches the initial yield curve

$$\vartheta(t) = \frac{1}{\kappa(t)}\frac{\partial f(0,t)}{\partial t} + f(0,t) + \frac{1}{\kappa(t)}\int_0^t e^{-2\int_u^t \kappa(s)ds}\sigma_r(u)^2 du.$$

It can be useful to switch variable and work with $x(t) = r(t) - f(0,t)$ with $f(0,t)$ being the instantaneous forward rate observed at t, then we can rewrite the above equation as

$$dx(t) = (y(t) - \kappa(t)x(t))dt + \sigma_r(t)dW(t);$$

with y being a deterministic function of time

$$y(t) = \int_0^t e^{-2\int_u^t \kappa(s)ds}\sigma_r(u)^2 du.$$

Under the T-forward measure the dynamics becomes

$$dx(t) = (y(t) - \sigma_r(t)^2 G(t,T) - \kappa(t)x(t))dt + \sigma_r(t)dW^T(t)$$

with

$$G(t,T) = \int_t^T e^{-\int_t^u \kappa(s)ds} du$$

Now define a, b, c as

$$\begin{aligned} a(t) &= -\kappa(t), \\ b(t) &= y(t) - \sigma_r(t)^2 G(t,T), \\ c(t) &= \sigma_r(t); \end{aligned}$$

and then rewrite $dx(t)$ as

$$dx(t) = (a(t)x(t) + b(t))dt + c(t)dW(t).$$

Integrating we get to the solution of the stochastic differential equation

$$x(t) = e^{\int_o^t a(u)du}\left(x(0) + \int_0^t e^{-\int_0^s a(u)du} b(s)ds + \int_0^t e^{-\int_0^s a(u)du} c(s)dW(s)\right).$$

For each $t < T$ we can compute conditional expectation and conditional variance, drawing values from the Normal distribution of the short rate.

$$\begin{aligned} E(x(T)|x(t)) &= A(t,T)x(t) + \int_t^T A(s,T)b(s)ds, \\ Var(x(T)|x(t)) &= \int_t^T A(s,t)^2 c(s)^2 ds, \\ A(s,T) &= e^{\int_s^T a(u)du}; \end{aligned}$$

with zero coupon bond price

$$P(t,T) = \frac{P(0,T)}{P(0,t)} e^{-x(t)G(t,T) - \frac{1}{2}y(t)G(t,T)^2}.$$

2.1.1 Hull and White

Recalling the definition of money market account where $r(t)$ is the short rate representing the continuously compounded rate at which the risk free investment increases

$$dB(t) = r_t B(t)dt,$$

with $r(t)$ that evolves under the risk neutral measure according to

$$dr(t) = [\vartheta(t) - ar(t)]dt + \sigma dW(t),$$

now consider a and σ to be positive constants, with ϑ chosen to fit the currently observed yield curve in the market

$$\vartheta(t) = \frac{\partial f^M(0,t)}{\partial T} + af^M(0,t) + \frac{\sigma^2}{2a}(1 - e^{-2at}),$$

where $\frac{\partial f^M(0,t)}{\partial T}$ denotes partial derivative of f^M with respect to its second argument and $f^M(0,T)$ being the market instantaneous forward rate at time 0 for maturity T

$$f^M(0,T) = -\frac{\partial lnP^M(0,T)}{\partial T},$$

with $P^M(0,T)$ the market discount factor for maturity T. The equation can be integrated to obtain

$$r(t) = r(s)e^{-1(t-s)} + \alpha(t) - \alpha(s)e^{-a(t-s)} + \sigma\int_s^t e^{-a(t-u)}dW(u),$$

where

$$\alpha(t) = f^M(0,t) + \frac{\sigma^2}{2a^2}(1 - e^{-at})^2,$$

$r(t)$ conditional on $r(s)$ with $t > s$ is Normally distributed with mean and variance given respectively by

$$\mathbb{E}[r(t)|r(s)] = r(s)e^{-a(t-s)} + \alpha(s)e^{-a(t-s)},$$
$$Var[r(t)|r(s)] = \frac{\sigma^2}{2a}[1 - e^{-2a(t-s)}].$$

It's possible to obtain the price for zero-coupon bonds, allowing us to compute a new term structure as a function of a given short rate. It's shown that

$$P(t,T) = A(t,T)e^{-B(t,T)r(t)}, \tag{2.1}$$

where

$$B(t,T) = \frac{1 - e^{-a(T-t)}}{a},$$
$$A(t,T) = \frac{P(0,T)}{P(0,t)}exp\Big(-B(t,T)\frac{\partial lnP(0,T)}{\partial t} - \frac{1}{4a^3}\sigma^2(e^{-aT} - e^{-at})^2(e^{2at} - 1)\Big).$$

2.2 Two Factor G2++

One factor models assumes that the dynamics of the short rate $r(t)$ are described by a single stochastic factor. This is like assuming the dynamics of all term structure to be function of a single short rate. It's not easy to find a realistic market situation that can be correctly represented by a one factor model, in such models it's impossible to find any feasible opportunities to capture interest rates correlation. Following some considerations of Rebonato [19], it's shown that a model with two or more factors capable to include forward rates correlation produces better results and is ideal to explain an acceptable portion of interest rates variability.

The Gaussian two factor model defines the short rate as the sum of two correlated Gaussian random variables, under the risk neutral measure $\mathbb{Q}$ the interest rate dynamics are

$$r(t) = x(t) + y(t) + \varphi(t), \quad r(0) = r_0.$$

With $\varphi(t)$ being a deterministic function of time, chosen to exactly fit the initial term structure of interest rates, $\varphi(0) = r_0$

$$\begin{aligned}\varphi(t) =& f(0,t) + \frac{\sigma^2}{2a^2}(1-e^{-at})^2\\ &+ \frac{\eta^2}{2b^2}(1-e^{-bt2}) + \rho\frac{\sigma\eta}{ab}(1-e^{-at})(1-e^{-bt}),\end{aligned}$$

and the two processes $x(t)$ and $y(t)$ defined as

$$\begin{aligned}dx(t) &= -ax(t)dt + \sigma dW_1(t), \quad x(0) = 0;\\ dy(t) &= -by(t)dt + \eta dW_2(t), \quad y(0) = 0;\end{aligned}$$

where W_1 and W_2 are two different Brownian motion with correlation ρ

$$dW_1(t)dW_2(t) = \rho dt.$$

If we integrate we obtain the two stochastic differential equations solutions

$$\begin{aligned}x(t) =& x(s)e^{-a(t-s)} + \sigma\int_s^t e^{-a(t-u)}dW_1(u),\\ y(t) =& y(s)e^{-b(t-s)} + \eta\int_s^t e^{-b(t-u)}dW_2(u);\end{aligned}$$

then the short rate evaluated at t_0 for a given maturity T is expressed as

$$r_0(T) = \sigma \int_0^T e^{-a(T-u)} dW_1(u) + \eta \int_0^T e^{-b(T-u)} dW_2(u) + \varphi(T);$$

letting $T \to \infty$ the two factors reverts to their initial value $x(\infty), y(\infty) \to 0$ and the $r(T)$ process mean reverts to $\varphi(t)$.

By the properties of stochastic integral of deterministic functions we obtain mean, variance and covariance from the bivariate Normal distribution of $r(t)$ conditional on $r(s)$ with $s < t$ as

$$\begin{gathered}
\mathbb{E}^{\mathbb{Q}}[x(t)|x(s)] = x(s)e^{-a(t-s)}, \quad \mathbb{E}^{\mathbb{Q}}[y(t)|y(s)] = y(s)e^{-b(t-s)}; \\
Var_s^{\mathbb{Q}}[x(t)] = \frac{\sigma^2}{2a}[1 - e^{-2a(t-s)}], \quad Var_s^{\mathbb{Q}}[y(t)] = \frac{\eta^2}{2b}[1 - e^{-2b(t-s)}]; \\
cov_s[x(t), y(t)] = \rho \frac{\sigma\eta}{a+b}[1 - e^{-(a+b)(t-s)}].
\end{gathered}$$

The two processes can also be rewritten in terms of two independent Brownian motions $\tilde{W}_1$ and $\tilde{W}_2$ applying a Cholesky decomposition

$$\begin{aligned}
&dW_1(t) = d\tilde{W}_1, \\
&dW_2(t) = \rho d\tilde{W}_1 + \sqrt{1-\rho^2} d\tilde{W}_2.
\end{aligned}$$

Substituting in the two equation for $x(t)$ and $y(t)$ by integration we get

$$\begin{aligned}
x(t) &= x(s)e^{-a(t-s)} + \sigma \int_0^t e^{-a(t-u)} d\tilde{W}_1(u), \\
y(t) &= y(s)e^{-b(t-s)} + \eta\rho \int_0^t e^{-b(t-u)} d\tilde{W}_1(u) + \eta\sqrt{1-\rho^2} \int_0^t e^{-b(t-u)} d\tilde{W}_2(u).
\end{aligned}$$

From the short rate we obtain the discount bond term structure

$$P(t,T) = \mathbb{E}^{\mathbb{Q}}\left[e^{-\int_t^T r_s ds}\right];$$

where the integral in the exponential $I(t,T) = \int_t^T r_s ds$ is shown[1] to be Normally

[1]Brigo Mercurio [4] p. 145

distributed with mean and variance given respectively by

$$\begin{aligned}
\mathbb{E}_t^{\mathbb{Q}}[I(t,T)] =& \frac{1-e^{-a(T-t)}}{a}x(t) + \frac{1-e^{-b(T-t)}}{b}y(t), \\
var_t[I(t,T)] =& \frac{\sigma^2}{a^2}\left[T-t+\frac{2}{a}e^{-a(T-t)} - \frac{1}{2a}e^{-2a(T-t)} - \frac{3}{2a}\right] \\
& \frac{\eta^2}{b^2}\left[T-t+\frac{2}{b}e^{-b(T-t)} - \frac{1}{2b}e^{-2b(T-t)} - \frac{3}{2b}\right] \\
& 2\rho\frac{\sigma\eta}{ab}\left[T-t+\frac{e^{-a(T-t)}-1}{a} + \frac{e^{-b(T-t)}-1}{b} - \frac{e^{-(a+b)(T-t)}-1}{a+b}\right].
\end{aligned}$$

With the zero coupon bond price at time t being defined as

$$\begin{aligned}
P(t,T) &= \frac{P(0,T)}{P(0,t)}e^{A(t,T)}, \\
A(t,T) &= \frac{1}{2}[var(t,T) - var(0,T) + var(0,t)] - \mathbb{E}_t^{\mathbb{Q}}[I(t,T)].
\end{aligned}$$

2.3 Monte Carlo Simulation

2.3.1 GSR Simulation in QuantLib

The Gaussian one factor model simulated for a given time grid of N intervals $t_0 < t_1 < \cdots < T_N$ is distributed as a Normal random variable with mean and variance given by [2]

$$\begin{aligned}
E(x(T)|x(t)) &= e^{\int_s^T a(u)du}x(t) + \int_t^T e^{\int_s^T a(u)du}b(s)ds, \\
Var(x(T)|x(t)) &= \int_t^T \left(e^{\int_s^T a(u)du}c(s)\right)^2 ds.
\end{aligned}$$

The simulation of the random variable $x(t)$ requires a prior definition of the discretization rule, typically depending on the schedule of the security we are willing to price; consider a security with N time intervals $t_0 < t_1 < \cdots < T_N$

$$x(t_{i+1}) = E(x_{i+1}|x(t_i)) + \sqrt{Var(x_{i+1}|x(t_i))}Z_i, \quad i = 0, \ldots, N-1;$$

[2]QuantLib.Gsr class, formulas for process.expectation and process.stdDeviation

with $Z_0, \dots Z_{N-1}$ being independent draws from a Gaussian distribution. The distribution of the values $x(t_1), \dots, x(t_n)$ are exactly that of the Gaussian process for the same value of $x(0)$. [3]

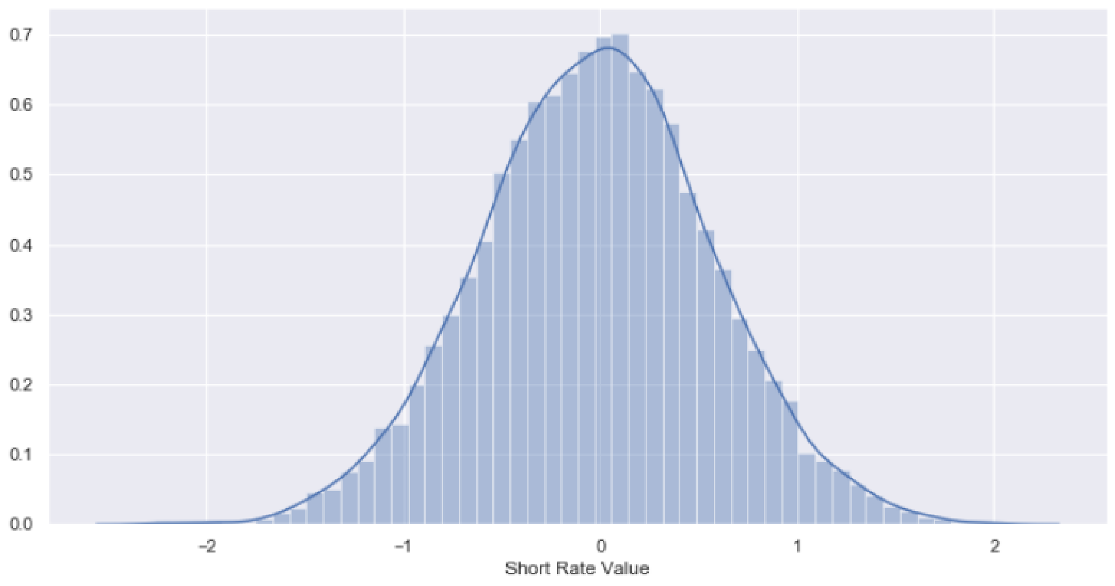


Figure 2.1: Gaussian short rate process ($\kappa = 0.05, \sigma_{1,\dots,n} = 0.0045$)
Density function for $t \to \infty$

With the required formulas reported in section(2.1) it's possible to obtain the entire discount curve as a function of the short rate for every simulated path $x(t_0), \dots, x(t_N)$, this feature of short rate models will be useful later to estimate different discount curves at future dates.

$$P(t,T) = \frac{P(0,T)}{P(0,t)} e^{-x(t)G(t,T) - \frac{1}{2}y(t)G(t,T)^2};$$

for $t = 0$ the model correctly reprice the present discount curve, the argument of the exponential simplifies and what remains is $P(0,T)$, the zero coupon bond at t_0.

[3]See Glasserman[3] on exact simulation of Gaussian short rate models

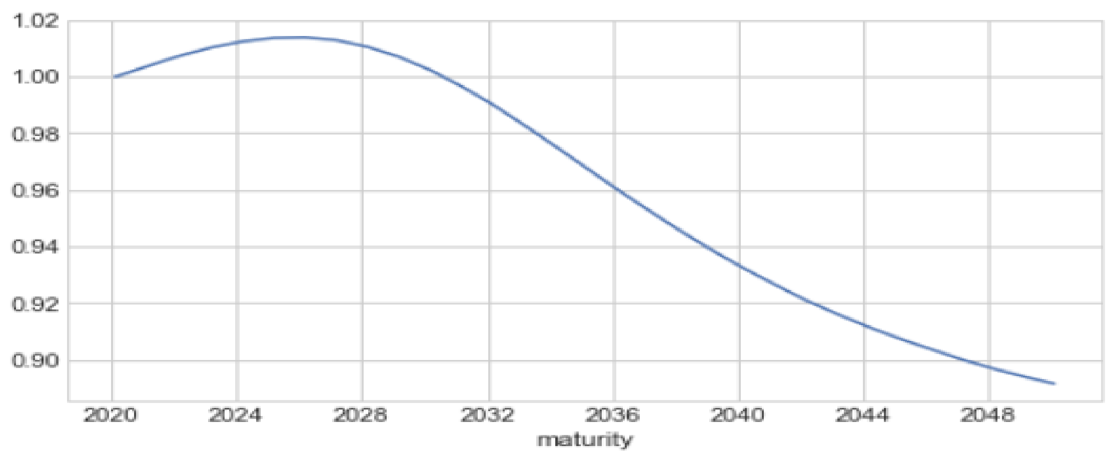


Figure 2.2: Term structure of T-bond prices with the Quantlib GSR model on EURIBOR 6M index

2.3.2 G2++ Exact Simulation

Consider a security with path dependent payoff which pays $V(T)$ at maturity, the price under the risk neutral measure $\mathbb{Q}$ is given by the discounted payoff

$$V(0) = \mathbb{E}^{\mathbb{Q}}\left[V(T)e^{-\int_0^T x(u)du}\right] :$$

To initialize the Monte Carlo simulation we discretize time into N intervals with $t_N = T$, then we can advance without bias the underlying variable $x(t)$ using equation (2.3.1). The series of simulated values of $x(t)$ will be used to evaluate future payoffs. What remains is to calculate the exact value of the discount factor $e^{-\int_0^T x(u)du}$, computing this value using integration schemes (e.i. trapezoidal rule) would lead to non-negligible inconsistencies.

To obtain unbiased results and avoid discretization errors we should simulate $e^{-\int_0^T x(u)du}$ along all the time span $(0, T)$

$$I(T) = -\int_0^T x(u)du;$$

by properties of stochastic integrals we know that $I(t,T)$ is normal, with mean and variance defined in the previous section. Another possibility to avoid discretization errors induced by integration or approximation schemes can be to simulate the process $x(t)$ under the measure induced by the numeraire $P(T)$.

Pricing under the T-measure $\mathbb{Q}^T$ allow us to rewrite our price formula as

$$V(0) = P(0,T)\mathbb{E}^{\mathbb{Q}^T}[V(T)];$$

under the $\mathbb{Q}^T$ dynamics there is no need to simulate the numeraire and thus we only need $x(t)$ for path dependency valuation.

For the exact simulation of the G2++ model consider the two risk adjusted processes $x(t)$ and $y(t)$ with dynamics under the T-forward measure $\mathbb{Q}^T$ given by

$$\begin{aligned} dx(t) &= \Big[-ax(t) - \frac{\sigma^2}{a}(1-e^{-a(T-t)}) - \rho\frac{\sigma\eta}{b}(1-e^{-b(T-t)})\Big]dt + \sigma dW_1^T(t), \\ dy(t) &= \Big[-by(t) - \frac{\eta^2}{b}(1-e^{-b(T-t)}) - \rho\frac{\sigma\eta}{a}(1-e^{-a(T-t)})\Big]dt + \eta dW_2^T(t), \\ &\qquad dW_1^T(t)dW_2^T(t) = \rho dt. \end{aligned}$$

Define the risk adjustments necessary for change of measure of the two processes with $s<t<T$ as

$$\begin{aligned} M_x^T(s,t) =& \Big(\frac{\sigma^2}{a^2} + \rho\frac{\sigma\eta}{ab}\Big)\Big[1-e^{-a(t-s)}\Big] - \frac{\sigma^2}{2a^2}\Big[e^{-a(T-t)} - e^{-a(T+t-2s)}\Big] \\ &- \frac{\rho\sigma\eta}{b(a+b)}\Big[e^{-b(T-t)} - e^{-bT-at+(a+b)s}\Big], \\ M_y^T(s,t) =& \Big(\frac{\eta^2}{b^2} + \rho\frac{\sigma\eta}{ab}\Big)\Big[1-e^{-b(t-s)}\Big] - \frac{\eta^2}{2b^2}\Big[e^{-b(T-t)} - e^{-b(T+t-2s)}\Big] \\ &- \frac{\rho\sigma\eta}{a(a+b)}\Big[e^{-a(T-t)} - e^{-aT-bt+(a+b)s}\Big]. \end{aligned}$$

Then $x(t)$ and $y(t)$ are normally distributed with conditional mean and conditional variance given by their bivariate distribution

$$\mathbb{E}^{\mathbb{Q}^T}[x(t)|x(s)] = x(s)e^{-a(t-s)} - M_x^T(s,t), \quad Var_s^{\mathbb{Q}^T}[x(t)] = \frac{\sigma^2}{2a}[1 - e^{-2a(t-s)}];$$
$$\mathbb{E}^{\mathbb{Q}^T}[y(t)|y(s)] = y(s)e^{-b(t-s)} - M_y^T(s,t), \quad Var_s^{\mathbb{Q}^T}[y(t)] = \frac{\eta^2}{2b}[1 - e^{-2b(t-s)}];$$
$$cov_s[x(t), y(t)] = \rho\frac{\sigma\eta}{a+b}[1 - e^{-(a+b)(t-s)}].$$

and

$$r(t) = x(t) + y(t) + \varphi(t), \quad r(0) = r_0.$$

Chapter 3

Least Squares Monte Carlo Simulation for American Options

3.1 A Dynamic Programming Approach

This section refers to the work of Longstaff and Schwartz [5], Glasserman [3], Glasserman and Yu [7]. The valuation and the determination of the optimal exercise date of derivatives with American-style exercise features is considered as one of the most challenging issue in option pricing theory. By definition of American option, at any exercise date the option holder has the right, but not the obligation, to exercise the option. Intuitively, what drives his decision is determined by the comparison between the payoff from immediate exercise and the expected payoff from continuation.

Monte Carlo simulation is a powerful numerical method to value any kind of financial derivative, however its forward-looking nature is not suited to address the valuation of American or Bermudan options because the optimal stopping problem embedded in the valuation of those kind of options makes this an unconventional and challenging problem for Monte Carlo. On the other hand, backward induction algorithm like binomial trees or finite difference methods becomes inefficient when we want to include a term structure driven by multiple risk factors.

Longstaff and Schwartz (2001) presented an efficient approach for the evaluation of such derivatives. The method provides approximate solutions by combining simulation, regression and a dynamic programming formulation of the problem. The application of Monte Carlo simulation allows the state variable to follow general stochastic processes such as jump-diffusions, HJM or forward market models (e.i. LMM) including other flexible possibilities.

Suppose to simulate the trajectories for the underlying state variable X_i with length m for $i = 1, \ldots, m$. Define $\tilde{h}_i$ as the payoff function for the exercise of an option written on the underlying X_i at t_i. Let $\tilde{V}_i(x)$ denote the value of the option at t_i given $X_i = x$, what we are interested in is the option value at t_0. The present value $\tilde{V}_0(X_0)$ is obtained recursively starting from the last exercise date and proceeding dynamically.

$$\begin{aligned}\tilde{V}_m(x) &= \tilde{h}_m(x),\\ \tilde{V}_{i-1}(x) &= \max\left\{\tilde{h}_{i-1}(x)\ ;\ \mathbb{E}[D_{i-1}(X_i)\tilde{V}_i(X_i) \mid X_{i-1} = x]\right\}.\end{aligned}$$

For $i = 1, \ldots, m.$ with $D_{i-1}(X_i)$ being the discount factor from t_{i-1} to t_i. Under a general formulation of the problem consider the simplified structure for deterministic discount factors, as already discussed this is something that can also be achieved also in practice taking the expectation under a consistent probability measure. Consider the following simplified framework

$$\begin{aligned}V_m(x) &= h_m(x),\\ V_{i-1}(x) &= \max\{h_{i-1}(x), E[V_i(X_i) \mid X_{i-1} = x]\};\end{aligned}$$

and let $D_{0,j}(X_j)$ be the deterministic non-negative discount factor function of X_j from time 0 to t_j such that $D_{0,0} = 1$ and $D_{0,i-1}(X_{i-1})D_{i-1,i}(X_i) = D_{0,i}(X_i)$

$$\begin{aligned}h_i(x) &= D_{0,i}(x)\tilde{h}_i(x),\\ V_i(x) &= D_{0,i}(x)\tilde{V}_i(x);\end{aligned}$$

for $i = 0,1, \ldots, m$. Set $V_0 = \tilde{V}_0$

$$\begin{aligned}V_m(x) &= h_m(x),\\ V_{i-1}(x) &= D_{0,i-1}(x)\tilde{V}_{i-1}(x),\\ &= D_{0,i-1}(x)\max\{\tilde{h}_{i-1}(x), \mathbb{E}[D_{i-1,i}(X_i)\tilde{V}_i(X_i) \mid X_{i-1} = x]\},\\ &= \max\{h_{i-1}(x), \mathbb{E}[D_{0,i-1}(x)D_{i-1,i}(X_i)\tilde{V}_i(X_i) \mid X_{i-1} = x]\},\\ &= \max\{h_{i-1}(x), \mathbb{E}[V_i(X_i) \mid X_{i-1} = x]\}.\end{aligned}$$

Under this framework the discounted values V_i is computed in a dynamic programming recursion for each time $i = 0,1,\ldots m$

3.1.1 Stopping Rule and Continuation Value

The continuation value of an American Option is given by the expectation of all future remaining cash flows conditional to the valuation date. In other words we can define the continuation value as the value of holding rather than exercising the option at the exercise date we are considering or, simply, its price.

$$C_i(x) = \mathbb{E}[V_{i+1}(X_{i+1})|X_i = x];$$

for $i = 0,\ldots,m-1$, and $C_m = 0$. At every exercise date i the option-holder compares the continuation value with the future payoff of the option in a dynamic programming procedure

$$C_i(x) = \mathbb{E}\big[\max\{h_{i+1}(X_{i+1}), C_{i+1}(X_{i+1})\}|X_i = x\big];$$

with $C_0(X_0)$ being the value of the option at t_0.

Define V_i as the value functions that determine the continuation values through conditional expectation

$$V_i(x) = \max\{h_i(x), C_i(x)\};$$

for $i = 1,\ldots,m$

According to Longstaff and Schwartz methodology, the continuation value is computed numerically from the "cloud" of points simulated at each step of the Monte Carlo. The issue of finding the best value within a parametric class is addressed by least squares regression, this method ease the computations and reduces the optimal stopping problem to a more tractable optimization problem.

The approximation is repeated recursively starting from the option value at expiry in a dynamic programming procedure as shown previously. Such method clearly restricts the number of possible exercise dates to be finite in discrete time.

The approximation provides a direct estimation of the conditional expectation function for each exercise date and gives the possibility to determine an optimal exercise strategy along each path. If the approximations satisfy $\hat{V}_i(x) =$

$\max\{h_i(x), \hat{C}_i(x)\}$ then we can determine the stopping rule $\hat{\tau}$

$$\hat{\tau} = \min\{i \in \{1, \ldots, m\} : h_i(X_i) \geq \hat{C}_i(X_i)\} \tag{3.1}$$

3.2 Longstaff and Schwartz

3.2.1 The Basic Algorithm

Consider an option V with payoff function $h(X)$ on an underlying X. The option we are considering has an American-style exercise feature, this means that at each exercise date $t_1, ..., t_M$ the option holder must choice to receive the payoff or to wait until the next exercise date. At maturity T the holder either exercises, or the option expires worthless.

The first issue we have to consider is to find an optimal stopping rule which would tell the option-holder when to exercise the option. To define such rule we need to compute the continuation value $C_i(x)$ as the expectation of V at t_{i+1} conditional to t_i

$$C_i(x) = \mathbb{E}[V_{i+1}(X_{i+1})|X_i];$$

this computation is obtained by approximation using a simple least squares regression and considering a set of basis functions ψ_r and constants $\beta_{i,r}$, with $r = 1, \ldots, M$.

$$\mathbb{E}[V_{i+1}(X_{i+1})|X_i = x] \approx \sum_{r=1}^{M} \beta_{i,r}\psi_r(x);$$

with C_i being the continuation value at time i, we can rewrite the equation as

$$C_i(x) = \beta_i^\top \psi(x),$$

with

$$\beta_i^\top = (\beta_{i1}, \ldots, \beta_{iM}), \quad \psi(x) = (\psi_1(x), \ldots, \psi_M(x))^\top.$$

Where the vector β_i of lenght M is given by

$$\beta_i = (E[\psi(X_i)\psi(X_i)^\top])^{-1} E[\psi(X_i)V_{i+1}(X_{i+1})] = B_\psi^{-1} B_{\psi V},$$

B_ψ is the MxM matrix and $B_{\psi V}$ the indicated vector of length M.

The optimal regression parameters $\beta_{i,r}$ are obtained minimizing the expected squared errors

$$\mathbb{E}\left[\left(\mathbb{E}[V(X_{i+1})|X_i = x] - \sum_{r=0}^{R} \beta_{i,r}\psi_r(x)\right)^2\right] \tag{3.2}$$

This approximations allow us to compute the continuation value of the option in an efficient way, as described in the following list of steps:

1. Generate N paths with M steps for the underlying
2. By no arbitrage set $V_M = h(X_M)$ as the terminal condition for each path.
3. Iterate backwards t = T - Δt
    - regress the $T_{t,i}$ against $S_{t,i}$, $i \in \{1, \dots, N\}$, given a certain number and kind of basis functions ψ_R
    - approximate the continuation value $C_{t,i}$ with the optimal regression coefficients $\beta_{t,R}$
    - set

    $$V_{t,i} = \begin{cases} h_t(S_{t,i}) & \text{if } h_t(S_{t,i}) > \hat{C}_{t,i} \\ \hat{C}_{t,i} & \text{if } h_t(S_{t,i}) \leq \hat{C}_{t,i} \end{cases}$$

    repeat iteration until $t = \Delta t$
4. At $t = 0$ calculate the LSM estimator

$$\hat{V}_0^{LSM} = e^{-r\Delta t}\frac{1}{N}\sum_{i=1}^{N} V_{\Delta t,i}$$

3.2.2 Number of Basis Functions

The LSM recursion uses a least squares regression to approximate continuation values, this assumption is justified if the evaluated conditional expectation of the option is an element of the square integrable functions L^2. By the property of

Hilbert spaces it is possible to represent the expectation as a function of the chosen basis functions[1]; the choice of such functions include the Laguerre, Hermite, Legendre polynomials and others, however the approximation produces satisfactory results also with simple powers of the state variable. The second and maybe more important issue that we have to consider before running the least squares regression concerns the choice of a state variable that efficiently represent both the evaluated payoff and the underlying process, this may seems trivial when the underlying variable follows a simple Brownian motion but the difficulty increases as we increase the number of factors and the complexity of our model.

In the following example the state variable is the Hull and White evolved short rate standardized by variance, [2] with basis functions defined as a series of simple powers of the state variable.

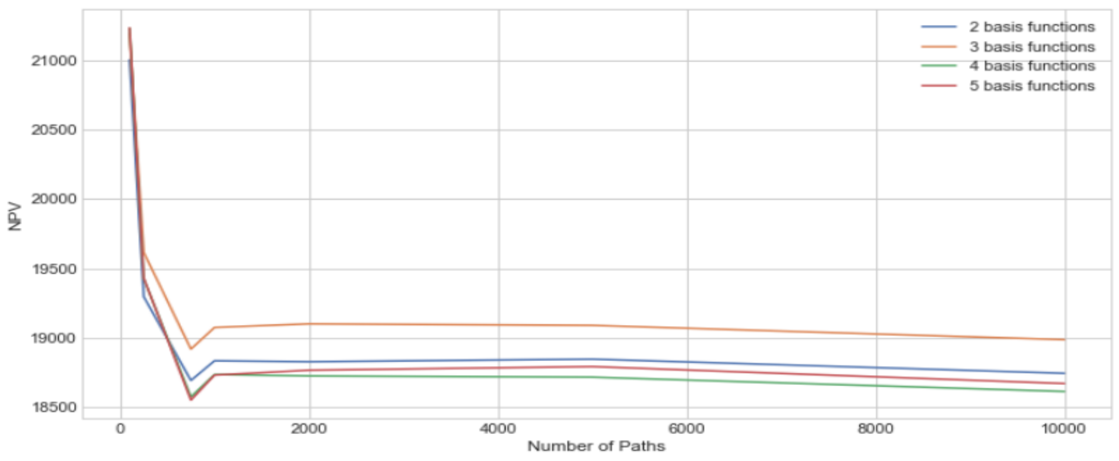


Figure 3.1: Price as function of the number of paths and number of basis function used for the regression
10Y ATM Bermudan swaption callable every 6MO priced on a 3% flat forward curve

The more we increase the number of the interpolating factors the more complex will result our regression function, however this does not immediately translate

[1]Longstaff and Schwartz [5]

[2]de-meaned and divided by variance, in this case standardization increase drastically the quality of the regression

into more accurate results as shown below in table (3.1): an excessively complex polynomial may not produce the desired result. In this case the use of a rank 4 polynomial is enough to obtain stable results.

Table 3.1: Price changes using 4 basis functions

N Paths	**Price**	**Runtime (seconds)**
100	21227.34	0.35
250	19425.22	0.74
750	18574.91	2.00
1000	18735.25	2.63
2000	18725.84	5.24
5000	18716.76	12.78
10000	18613.19	25.59

For any finite number of basis function M and exercise date K the least squares Monte Carlo approximation will produce smaller or equal results than the optimal value of the American option

$$V(x) \geq \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} LSM(M,K)$$

This is due to the structure of the algorithm that underestimate the value based on the optimal stopping rule. In general the quality of the regression always produce better performances when the underlying is around the strike, as the option considered is at the money the results are similar also with low number of basis functions.

Chapter 4

EUR Market Implementation

4.1 Models Setup

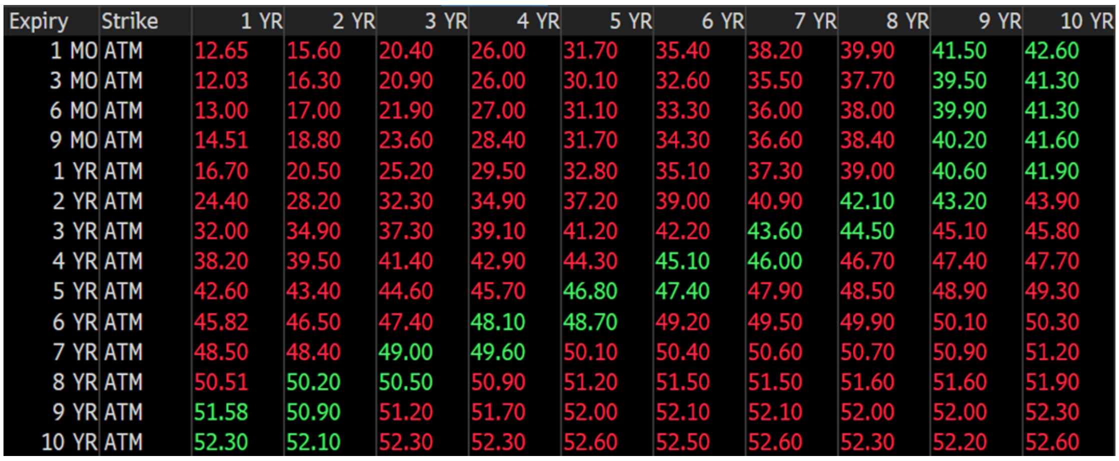
Early exercise derivatives like Bermudan swaptions can be efficiently priced combining Monte Carlo simulation and a backward induction algorithm. Following the methodology presented by Longstaff and Schwartz I implemented the Least squares Monte Carlo (LSM) for the valuation of continuation values and relative exercise probabilities of American options. Interest rates derivatives add another dimension in terms of pricing complexity, in this case it is not sufficient to evolve a single stochastic process to explain the underlying security's dynamics but it's necessary to setup a market model in order to explain the evolution of the whole term structure.

Consider a 5x5 Bermudan swaptions indexed to EURIBOR 6M, the structure is non callable for 5 years, then the holder has the possibility to enter annually into a swap with maturity 10 years, the swaption strike is at the money on 10 millions notional with notification dates fixed at 10 business days prior the fixed payment dates.

Table 4.1: exercise schedule
Daycount convention: Modified following
Calendar: TARGET

Notionals	Notification	Effective Dates	Exercise Strike
10,000,000.00	07/02/2025	21/02/2025	-0.06645 %
10,000,000.00	09/02/2026	23/02/2026	-0.06645 %
10,000,000.00	08/02/2027	22/02/2027	-0.06645 %
10,000,000.00	07/02/2028	21/02/2028	-0.06645 %
10,000,000.00	07/02/2029	21/02/2029	-0.06645 %

Before running the simulation it's necessary to calibrate the volatility parameters of the one factor (GSR) and the two factor (G2++) Gaussian short rate models to market quoted swaptions prices. Calibration represents the first tricky challenge of pricing exotic derivatives, short rate models must be calibrated to explain a satisfactory part of volatility surface, the model must reprice correctly the chosen calibration basket, giving good results also for a little portion of smile. A low calibration error is fundamental for price calculation, and even more when we want to estimate sensitivities.



Expiry	Strike	1 YR	2 YR	3 YR	4 YR	5 YR	6 YR	7 YR	8 YR	9 YR	10 YR
1 MO	ATM	12.65	15.60	20.40	26.00	31.70	35.40	38.20	39.90	41.50	42.60
3 MO	ATM	12.03	16.30	20.90	26.00	30.10	32.60	35.50	37.70	39.50	41.30
6 MO	ATM	13.00	17.00	21.90	27.00	31.10	33.30	36.00	38.00	39.90	41.30
9 MO	ATM	14.51	18.80	23.60	28.40	31.70	34.30	36.60	38.40	40.20	41.60
1 YR	ATM	16.70	20.50	25.20	29.50	32.80	35.10	37.30	39.00	40.60	41.90
2 YR	ATM	24.40	28.20	32.30	34.90	37.20	39.00	40.90	42.10	43.20	43.90
3 YR	ATM	32.00	34.90	37.30	39.10	41.20	42.20	43.60	44.50	45.10	45.80
4 YR	ATM	38.20	39.50	41.40	42.90	44.30	45.10	46.00	46.70	47.40	47.70
5 YR	ATM	42.60	43.40	44.60	45.70	46.80	47.40	47.90	48.50	48.90	49.30
6 YR	ATM	45.82	46.50	47.40	48.10	48.70	49.20	49.50	49.90	50.10	50.30
7 YR	ATM	48.50	48.40	49.00	49.60	50.10	50.40	50.60	50.70	50.90	51.20
8 YR	ATM	50.51	50.20	50.50	50.90	51.20	51.50	51.50	51.60	51.60	51.90
9 YR	ATM	51.58	50.90	51.20	51.70	52.00	52.10	52.10	52.00	52.00	52.30
10 YR	ATM	52.30	52.10	52.30	52.30	52.60	52.50	52.60	52.30	52.20	52.60

Figure 4.1: co-terminal swaptions normal volatility used for calibration (green)

The two models are calibrated on co-terminal at the money Euro swaptions normal volatilities in figure (4.1) recorded on 19 February 2020. The mean reversion and correlation parameters are chosen to be constants. The two factor model, due to the presence of the correlated process and an higher number of parameters, produces smaller calibration errors with respect to the one factor case. Moreover the choice of a time dependent parameter for volatility in a G2++ model produces almost perfect results.

Table 4.2: GSR and G2++ calibration

Start Date	**End Date**	**Vol σ_{GSR}**	**Vol σ_{G2}**	**Vol η_{G2}**
19/02/2020	19/03/2020	0.50 %	0.63 %	1.10 %
19/03/2020	19/05/2020	0.47 %	0.59 %	1.05 %
19/05/2020	19/08/2020	0.48 %	0.60 %	1.09 %
19/08/2020	19/11/2020	0.43 %	0.60 %	1.10 %
19/11/2020	19/02/2021	0.49 %	0.61 %	1.10 %
19/02/2021	21/02/2022	0.50 %	0.63 %	1.20 %
21/02/2022	20/02/2023	0.54 %	0.63 %	1.09 %
20/02/2023	19/02/2024	0.57 %	0.65 %	1.00 %
19/02/2024	19/02/2025	0.60 %	0.68 %	0.91 %
19/02/2025	19/02/2026	0.61 %	0.69 %	0.81 %
19/02/2026	19/02/2027	0.61 %	0.68 %	0.68 %
19/02/2027	21/02/2028	0.65 %	0.71 %	0.60 %
21/02/2028	19/02/2029	0.68 %	0.69 %	0.60 %
19/02/2029	-	0.71 %	0.78 %	0.60 %

4.2 Pricing Bermudan Swaptions with LSM

Before running the regression algorithm it's necessary to simulate future values of the underlying swap. Considering the previously calibrated GSR and G2++ models and setup a Monte Carlo simulation running on a certain date grid, by the properties of short rate model it's possible to run a exact simulation drawing values from the Gaussian distribution.

I choose to simulate all the processes under the T-forward measure to avoid discounting errors or further simulations as explained in Chapter 2, moreover time steps are fixed on the exercise dates as discretization is not a problem. Using the models properties we compute a new discount bond term structure for each time step to evaluate the remaining cash flows of the underlying swap in relation to the simulated short rate, then at each time step and for every path we will obtain a different "realized" value for the underlying swap. As we keep up with the simulation it's important to remember that the more we move forward the shorter will be the tenor of the swap, for at the money swaps the two legs future exposure will converge to zero as cash flows gets smaller.

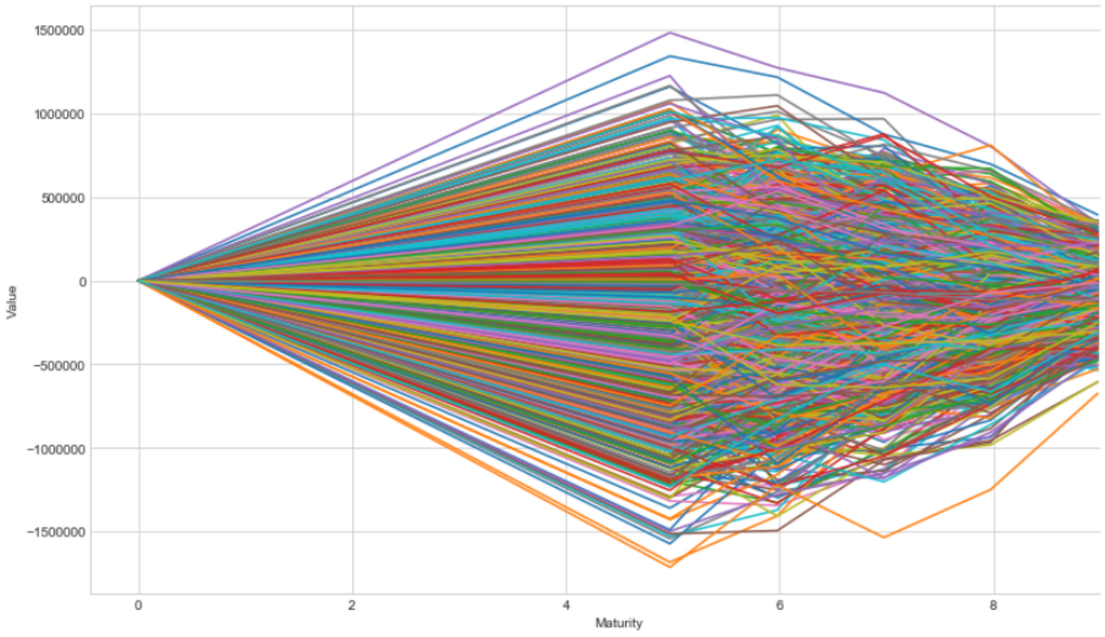


Figure 4.2: ATM swap value simulation (2000 path)

At inception the value of the swap is zero as the rate is equal to the swap fair rate. Then the initial time step will have high variance as the first call date is scheduled in 5 years from now. Define a Nxm matrix of swap future values where N denotes the number of paths and m the length of the time grid, the payoff of the swaption at every call date is

$$\mathbf{swaption} = max[\mathbf{swap}; 0]$$

then the payoff of the swaption is obtained by setting all negative paths of the swap equal to zero. Once generated the Monte Carlo trajectories of the underlying up to maturity it is possible to proceed with the Longstaff and Schwartz algorithm as described in the following pseudo-code: [1]

[1]see full Python code in Appendix A

```
m = length of the time grid array (starting time + exercise dates)
1) Generate the Monte Carlo trajectories for nPath and m time steps.
2) V[:, -1] = payoff at maturity
**for** i in (m-1, 0, step = -1) **do**
    a) deflate V[:,i+1] for the chosen discount factor at time[i]
    $b_1$) regress V[:, i+1] as a function of the state variable at time[i]
    $b_2$) Compute the approximated continuation values from the regression
    $c_1$) compare the continuation value with the discounted payoff of exercise
    $c_2$) define V[:,i] as the maximum between this two numbers
    Iterate until t[0]
    The expected value at $t_0$ is the npv of the Bermudan swaption.
**end for**
```

For the approximation I used the first 5 probabilists' Hermite polynomials, however also simple power of the state variable produces good results as we are pricing an ATM swaption.

$$\begin{aligned}
He_0(x) &= 1,\\
He_1(x) &= x,\\
He_2(x) &= x^2 - 1,\\
He_3(x) &= x^3 - 3x,\\
He_4(x) &= x^4 - 6x^2 + 3.
\end{aligned}$$

The state variable used for the approximation should link together the underlying process and the payoff. In this example the continuation values are approximated as a function of the standardized short rate. If the regression function fits poorly a solution could be to try other polynomial functions or a change of the state variable.

Typically the regression produces a good fit when the underlying value is close to the strike, to check the quality of the regression we can plot it over one cycle: here it's evident that the function doesn't produce good results on the extreme tails of the distribution.

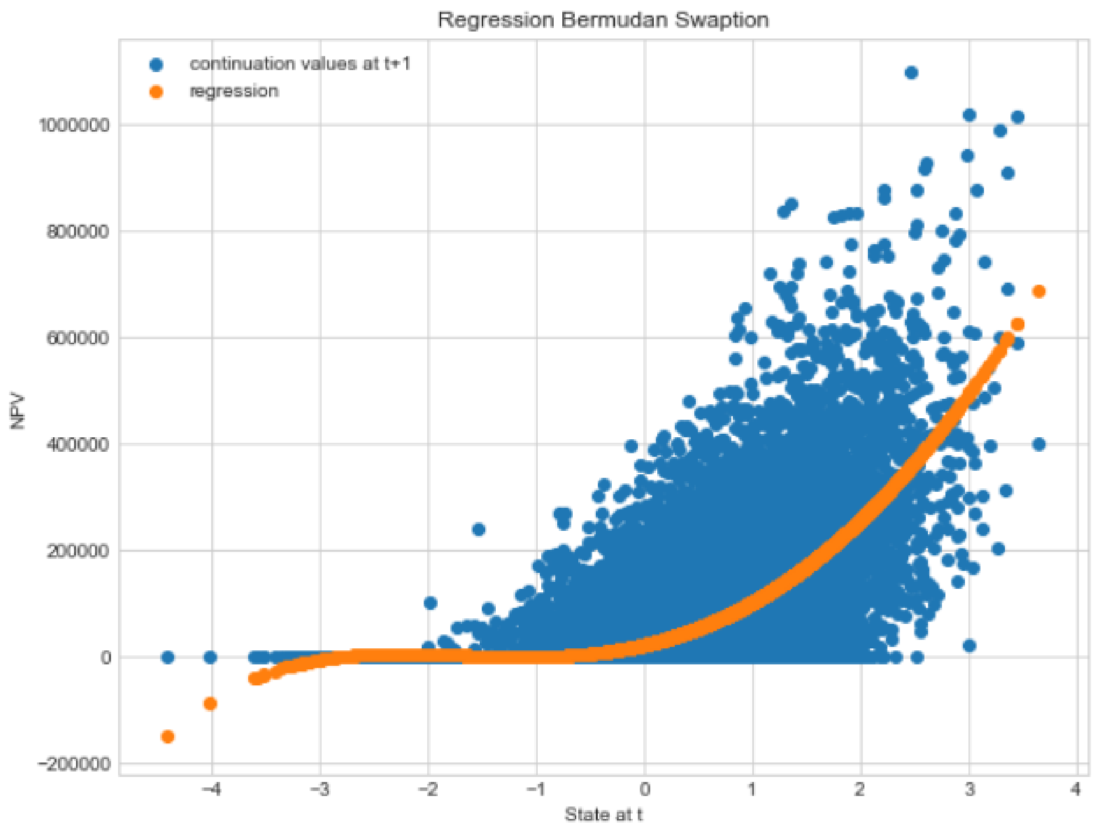


Figure 4.3: regression

In the figure we recognize the classical call option payoff of the payer swaption, the output of the least squares Monte Carlo pricing is reported below in table (4.4) in comparison with LMM from DLIB function of Bloomberg.

Table 4.3: pricing results (in Euros)
2000 run-path

GSR	**G2++**	**LMM (DLIB)**
246,015.01	239,279.21	240,214.89

When we calibrate to swaption volatilities the results are coherent to the analysis made by Andersen and Andreasen [20], the price of a Bermudan swaption in the two factor model is lower than the price in the one factor model, due to the impact of rates de-correlation.

4.3 Hedging Callable I.R. Derivatives

After a general discussion on pricing, now it's important to turn our attentions to the ability/inability of short rate models to explain a complex volatility structure. Bermudan swaption are among the most liquid exotic derivatives traded in OTC markets, in recent times their volume has increased as they are used to mitigate specific interest rate exposures coming from sophisticated products settled to corporations and institutional investors, this market development led financial practitioners to prefer high performances and low-factors solutions (e.i. Hull and White PDE) to Monte Carlo simulations, losing all the information on forward rates correlation included in multi factor models.

The optionality embedded in Bermudan swaptions is basically a matter of choosing the most valuable swap among different dates, thus the main driver for volatility risk is represented by the intrinsic correlation of swap rates. Combining Monte Carlo simulation with the Longstaff and Schwartz regression produces good results in terms of speed/accuracy and gives the possibility to include forward rates de-correlation by the implementation of multi-factor models. However as we move towards hedging considerations we may get into trouble with risk sensitivities: which is the correct way to obtain accurate first order sensitivities from the simulation? How should I interpret my Vega exactly?

Delta is defined as the sensitivity to small interest rate changes, it can be computed considering parallel shifts of the initial yield curve or by bucketing independent movements of different portion of the curve. Some more complicated considerations must be made for Vega as we need to examine the sensitivity to changes of all the calibration basket volatility grid; when we calibrate a model like G2++ we specify a certain volatility structure, the main difficulty here is capturing all the unobservable features of forward volatilities and correlations dynamics. Swap rates can be decomposed into weighted combinations of forward rate, so performing calibration on co-terminal swaption seems to be the best way to include these opaque informations.

Calibration errors and complex algorithms may produce ambiguous results, therefore traders needs to *understand the interaction between mathematics and the* *reality of markets, data, regulations and human behaviour*[2] applying their knowledge to overcome any model inconsistency, making choices that may seems unbalanced in terms of model sensitivities implemented by risk management institutions.

4.3.1 Dynamic Delta Hedging with LSM

Consider a friction-less market where the only random variable is represented by a single stochastic process $S(t)$ with constant volatility. In this simplified framework dynamic delta hedging is the correct way to neutralize perfectly all the sensitivity of our portfolio. This assumptions are clearly violated in reality, however the implementation of an hedging strategy in benchmark models like Black and Scholes is useful to set up other, more complicated, hedging strategies.

In Black and Scholes world we assume all market to be defined by a stock $S(t)$ and a bond $B(t)$ representing the money market account

$$\begin{aligned} dS(t) &= rS(t)dt + \sigma S(t)dW^{\mathbb{Q}}(t), \\ dB(t) &= rB(t)dt. \end{aligned}$$

Every instruments in the market is reachable with combinations of other payoff and the market is complete. Consider an American-style put option written on the underlying variable $S(t)$, the model parameters are chosen to be equal to the first example of the Longstaff and Schwartz paper [3]. Set $S_0 = 36$, $K = 40$, $\sigma = 0.20$, $r = 0.06$, $T = 1$, the simulation runs on 100000 (50000 antithetic) paths with 50 call dates. The LSM algorithm for the option value at t_0 gives the following results

$$\begin{aligned} V_0 &= 4.47, \\ Std.Deviation &= 0.010. \end{aligned}$$

With exercise probabilities plotted below in (4.4)

[2]M. Morini[12]

[3]see Appendix C for the full table

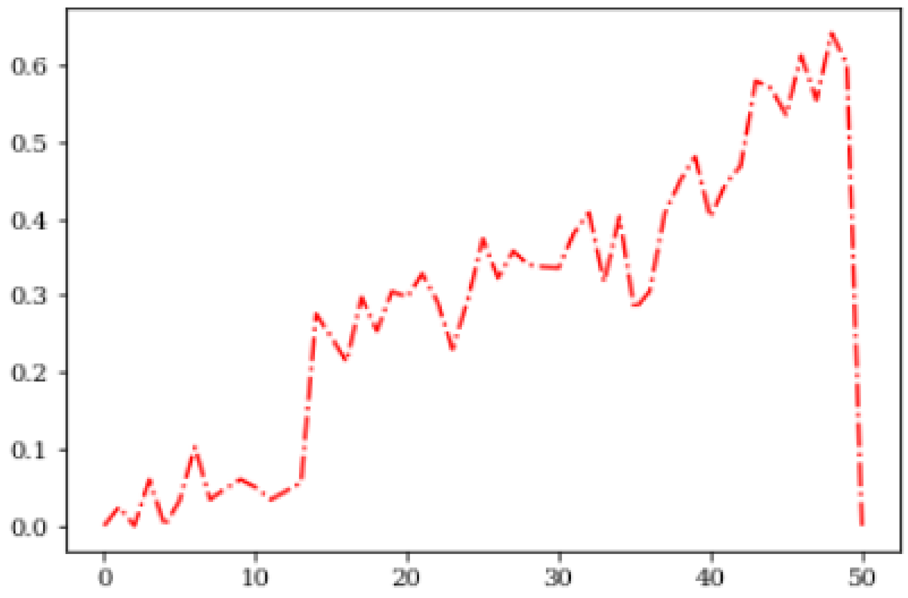


Figure 4.4: absolute exercise probabilities

Suppose to hold a portfolio composed by the put option: to avoid risk at t_0 we need to delta-hedge our position, then the delta of our option in a constant volatility world is given by

$$\Delta^V(t) = \frac{\partial V(t)}{\partial S(t)},$$

since the market is complete we can replicate the option value with a hedging portfolio composed by $\Delta^V(t)$ units of $S(t)$ and $V(t) - \Delta^V(t)S(t)$ units of money market account $B(t)$ such that at inception the portfolio value equals zero

$$V(t) = \Delta^V(t)S(t) + \big(V(t) - \Delta^V(t)S(t)\big)B(t)$$

The replicating strategy consists in a self-financing portfolio where at each rebalancing date t we are long on the put option and we neutralize the negative $\Delta^V(t)$ position combining the needed weights of the underlying and the bond as shown before.

Unwinding the position at the end of the simulation we obtain the P&L of the presented strategy

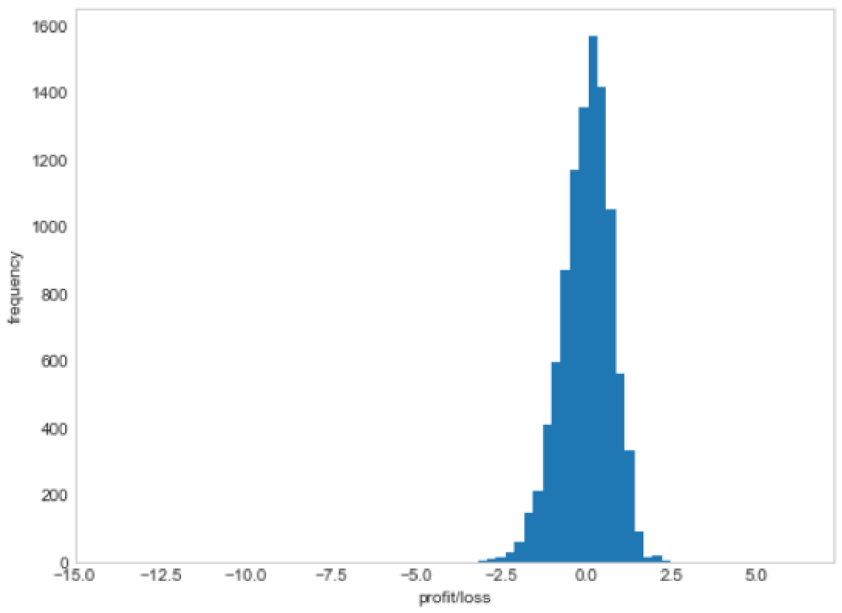


Figure 4.5: Profit and Loss distribution (10000 runs)
average: -0.005

The Black and Scholes framework gives the possibility to set up hedging strategies in a simplified market, even if the assumption are not very realistic the method is useful to understand the logic behind hedging portfolios since the basic methodology remains the same also when we increase the complexity of the model. The more we try to get a closer resemblance to reality the more we need market models capable to explain different empirical phenomenons, and even if the perfect models does not exist [4], setting up an advanced market model is never an easy task.

Just to see what happens when we try to add more dimensions making some more complicated assumptions, consider the schedule of the same 5x5 Bermudan payer swaption priced in the previous section with exercise dates starting in 5Y from now and expiring in 2030. The swaption is now priced with the one factor GSR on a 3% flat forward curve with fixed rate of 2.80%, the mean reversion and the volatility are fixed respectively to 0.03 and 0.0020. At inception we buy the Bermudan swaption paying $V(0)$ to enter in a long position on interest rates. To

[4]Rebonato [10]

neutralize the sensitivity we compute numerically the Delta on a million notional.

$$V_0 = 12691.76$$
$$\Delta_0^V = 281.24$$

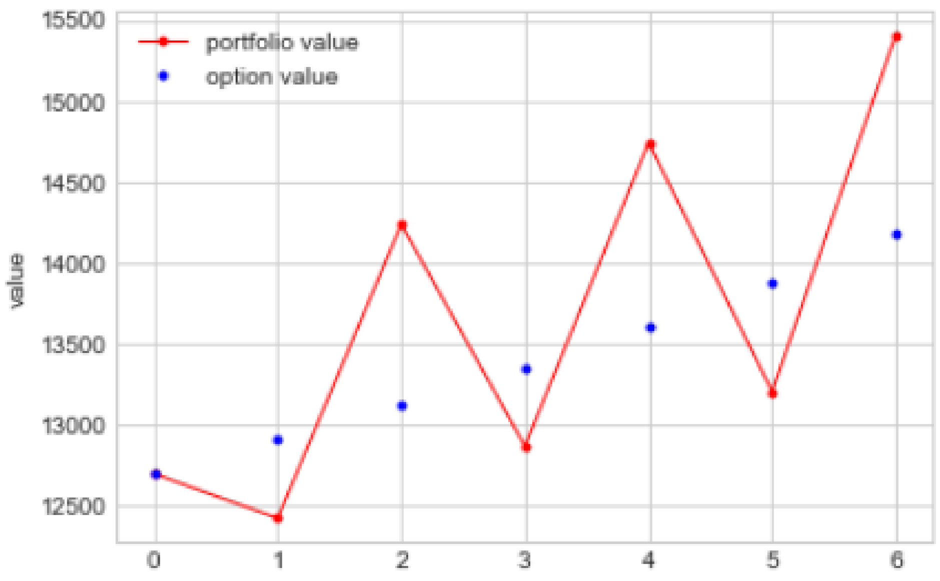
To hedge the Delta-position consider the same swap underlying the Bermudan swaption with notional weighted for the basis point value (BPV), we neutralize the sensitivity selling the payer swap and collecting the premium. The remaining cash value of the option to be paid is funded at the risk free rate.

```
Initial Portfolio
Option Value (buy): 12691.762
------------------------------------
Delta Hedging Portfolio
Hedging swap (sell): 6976.003
Bond Funding position: 5715.759
------------------------------------
Portfolio Cash Value at t0:    0.000
```

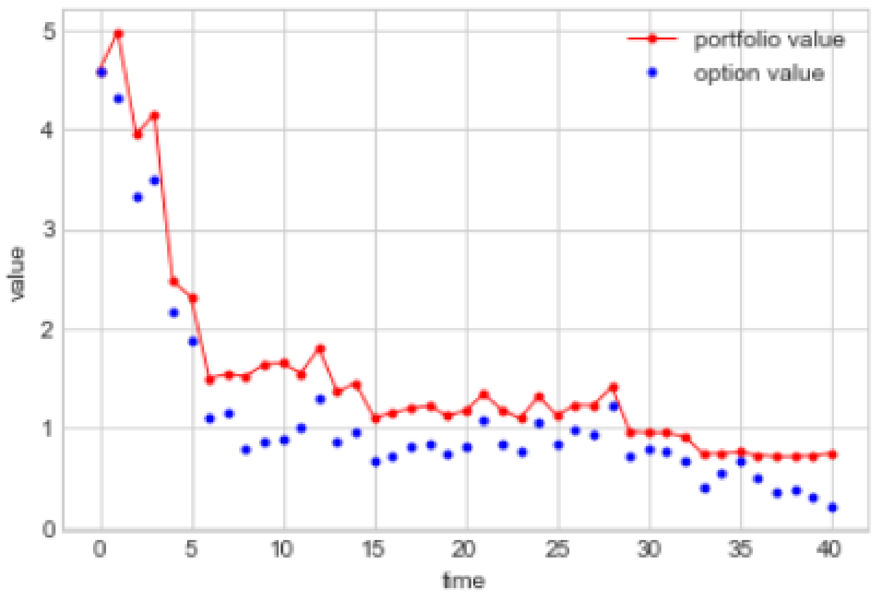
Define the swap weight needed to neutralize the Delta position of the option at the start of each period i as Δ_i^S, at inception we pay V_0 to buy the option and we go short on Δ_0^S units of swap, financing the remaining cash position with a risk-free bond B_0.

Table 4.4: Delta-hedging portfolio

Period	Option Value	Swap Position	Cash Position	Portfolio Value
0	(12691.76)	6976.01	5715.75	0.00
1	(12901.77)	6608.22	5808.58	484.97
2	(13122.94)	7039.92	7195.04	(1112.03)
3	(13353.40)	6684.33	6178.92	490.16
4	(13601.33)	7104.99	7635.35	(1139.01)
5	(13873.48)	6640.94	6559.85	672.69
6	(14184.70)	7107.69	8299.84	(1222.84)



(a) Swap Delta-Hedging



(b) Black and Scholes Delta-Hedging

Figure 4.6

For each step the option V_i it's repriced at the prevailing market conditions and the hedging portfolio is adjusted to neutralize the Delta position as

$$\Delta_i^V = \Delta_i^S, \qquad i = 0, 1, \ldots 6.$$

Whenever we rebalance our swap position cash it's received or paid, the resulting net cash balance is reinvested at the risk-free rate over each period. For example at t_1 the hedging portfolio will be composed by $\Delta_1^S + B_0 e^{r\tau} + cash$, where the *cash* component is defined as the quantity of swap that we need to buy or sell $\Delta_1^S - \Delta_0^S$. The resulting Delta hedging portfolio is visible in figure (4.6a), the slippage becomes evident as we step outside the Black and Scholes world.

4.3.2 A Primer on Volatility Skew

Using a broad definition we can define volatility as the level of dispersion within a series of values, a first characterization can be made between backward-looking and forward-looking volatility:

1. **Realized-Historical volatility:** past volatility recorded for a given time series

$$\sigma_R = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(S_i - \hat{S})^2}$$

where $\hat{S}$ denotes the sample mean for n observations.

2. **Forward-Future volatility:** defined as the value of σ_I that matches model to market prices (Black volatility). Implied volatility reflects the fair volatility expected by risk neutral investors in the market.

The Black and Scholes model, despite some well known restrictions and practical inconsistencies, represents the world market standard for option pricing. All model's parameters are directly observed in the market or set by the term sheet, volatility presents an exception as we pretend to be aware of a future unknown quantity.

The presence of a constant volatility constitutes the biggest shortcoming of Black and Scholes, however this imperfection has been turned into a strength by market players due to its practical tractability that allows to express prices in simple implied volatility quotes; the consequent challenge is therefore to introduce a strike dependency for volatilities in order to include some volatility skew. This inconsistency of the model suggests that implied volatility is nothing more than a convenient input parameter, and does not carry any information on the real underlying volatility process.

$$Opt(S, K, r, 0, T, \sigma) = \mathbf{Black}(S, K, r, T, \sigma_{blk})$$

where S, K, r, T are the underlying price, strike, risk free rate and time to maturity, with σ_{blk} being the market implied volatility. The formula allows to get option prices as functions of Black implied volatility for the given level of S and K observed at t_0

$$Opt(S, K, r, 0, T, \sigma) = \mathbf{Black}(S, K, r, T, \sigma_{blk}(S, K))$$

Following a famous contribution of Rebonato [10] the implied volatility σ_{blk} is just *the wrong number to put in the wrong formula to get the right price of plain-vanilla options* and does not carry any information about the real volatility process. As we try to hedge we need to account for the dependence of implied volatility to changes of the underlying variable

$$\begin{aligned}\Delta_{opt} &= \Delta_{blk} + \frac{\partial \mathbf{Black}(S, K, r, T, \sigma_{blk}(S, K))}{\partial \sigma_{blk}(S, K)} \frac{\partial \sigma_{blk}(S, K)}{\partial S} \\ &= \Delta_{blk} + \mathcal{V}_{blk} \frac{\partial \sigma_{blk}(S, K)}{\partial S}\end{aligned}$$

To solve this problem it's necessary to include some assumptions on the last term, as there are no informations available about the sensitivity of σ_{blk} to changes of the underlying variable, models produce two extreme cases

1. **Sticky smile:** the smile does not change its position, at time t_0 the option price is given by $\mathbf{Black}(S_0, K, \sigma_{blk}(S_0, K))$, as the underlying move to $S_0 + \delta S$ volatility is not affected. The new price is given by $\mathbf{Black}(S_0 + \delta S, K, \sigma_{blk}(S_0, K))$ and we can work with $\sigma = f(K)$

$$\Delta_{opt} = \Delta_{blk}$$

as the implied volatility does not react to changes in rates

$$\frac{\partial \sigma_{blk}(K)}{\partial S} = 0$$

2. **Floating smile:** The smile changes position as a function of the moneyness $\sigma_{blk}(S, K) = \sigma_{blk}(S - K)$, or log-moneyness $\sigma_{blk}(S, K) = \sigma_{blk}\big(\frac{log(S)}{log(K)}\big)$.

In the interest rates world empirical analysis shows that real smiles behaviour is something in between those two extreme scenarios, the implementation of a stochastic volatility model capable to define shape and movements of the smile is the best way to include volatility skew and avoid P&L swings of complex derivatives portfolio.

Price estimation of structured product is never an easy task as we need to evaluate exotic payoffs with reality-consistent market models. Traders take advantage of Black and Scholes practical tractability to establish a fast and convenient quoting system for volatilities, however reality hits back when it comes to estimate consistent sensitivities. Suppose that we want to estimate the sensitivities of a Bermudan swaption, we calibrate a model to an appropriate set of swaptions and then we compute the price using the LSM algorithm. Now we are interested in the sensitivity of the Bermudan swaption to small changes of volatility of each one of the calibration-swaption, the *direct*[5] method intuitively suggests to apply a small shock to the volatility of the calibration basket, re-calibrate the model, obtain the new price of the "bumped" swaption and compute Vega numerically. Suppose that our model is calibrated with a small error, say 0.01%, then the magnitude of the error is more or less equal to the bump we applied to volatilities, that's the reason why this method typically produces wrong results.

A possible solution to the problem is presented by Piterbarg [21], according to the methodology it's possible to obtain robust results using *indirect* sensitivities. The idea behind indirect sensitivities is to rule out the calibration noise by a reverse mapping of market/model volatilities, these sensitivities can be computed by bumping each swaption volatilities to minimize the errors between market Vega

[5]Using the definition given by Piterbarg [21]

(bumping the market volatilities) and the model Vega (bumping the model volatilities).

Building a solid risk management framework for Bermudan swaptions and for exotics Libor derivatives in general is a challenging task. As the complexity of our portfolio increases it's important to keep in mind three correlated issues:

- Is my model consistent with the market phenomena I am trying to explain? How can I calibrate my model to the prevailing volatility structure?
- Which is the best way to compute sensitivities? Are they consistent with reality or are they only good for the model?
- What portion of risk does my Greeks explain? Do they account for volatility-smile and rates de-correlation?

The research of adequate pricing techniques capable to account for all source of risk lead to the application of complex and unstable algorithms that create noisy results. Traders succeed in addressing this problem using their knowledge of the instruments, with a deep technical analyisis of the structure of the model.

Appendix A

Python code

The whole code is written as Jupyter notebook, following some tips of Balaraman and Ballabio[18] and Matthias Groncki blog, with an acknowledgement for Giuseppe Trapani for a general review of the code and for the essential contribution he gave for the realization of the whole thesis. The code runs using Numpy, Pandas, Matplotlib, QuantLib and Statsmodel libraries.

A.1 LSM pricing with GSR

Import libraries

```python
import numpy as np
import pandas as pd
import QuantLib as ql
import matplotlib.pyplot as plt
import datetime
import statsmodels.api as sm
```

Pricing functions definition

```python
def timeFromReferenceFactory(daycounter, ref):
    def impl(dat):
        return daycounter.yearFraction(ref, dat)
    return np.vectorize(impl)
```

```

def makeSwap(start, maturity, notional, fixedRate, index, tipo = ql.VanillaSwap.Payer):
    """
        Returns the Quantlib swap object
    """
    fixedLegTenor = ql.Period("1Y")
    fixedLegBDC = ql.ModifiedFollowing
    floatLegTenor = index.tenor()
    fixedLegDC = index.dayCounter()
    spread = 0.0
    fixedSchedule = ql.Schedule(start, maturity, fixedLegTenor, index.fixingCalendar(), fixedLeg
                                   ql.DateGeneration.Backward, False)
    floatSchedule = ql.Schedule(start, maturity, floatLegTenor, index.fixingCalendar(), index.bu
                                    index.businessDayConvention(), ql.DateGeneration.Backward,Fa
    swap = ql.VanillaSwap(tipo, notional,fixedSchedule,fixedRate,fixedLegDC,floatSchedule,index,
    return swap

def getfixed(swap, data):
    """
        Input:
        - swap: ql.VanillaSwap Object
        - data: reference date
        Returns:
        - array of payment times for the fixed leg
        - array of cash flows for the fixed leg
    """
    date_temp = calendar.advance(data, -callability)
    time_temp = ql.Actual365Fixed().yearFraction(today,date_temp)
    t = time_temp
    fixed_leg = swap.leg(0)
    n = len(fixed_leg)
    fixed_times=[]
    fixed_amounts=[]
    if n == len(date_grid):
        for i in range(n):
            refDate = calendar.advance(date_grid[i], callability)
            cf = fixed_leg[i]
            cf_time = daycounter.yearFraction(refDate, cf.date() )
            t_i = timeFromReference(cf.date())
            if t_i > t:
                fixed_times.append(t_i)
                fixed_amounts.append(cf.amount())
        fixed_times = np.array(fixed_times)
        fixed_amounts = np.array(fixed_amounts)
    else:
        for i in range(n):
            cf = fixed_leg[i]
            t_i = timeFromReference(cf.date())
            if t_i > t:
                fixed_times.append(t_i)
                fixed_amounts.append(cf.amount())
        fixed_times = np.array(fixed_times)
        fixed_amounts = np.array(fixed_amounts)
    return fixed_times, fixed_amounts


def getfloating(swap, data):
    """
        Input:
            - swap: ql.VanillaSwap Object
        - data: reference date

        Returns:
        - array --> payment times for the floating leg
        - array --> accrual between payments
        - array --> start of the accruals
        - array --> end of the accruals
        - array --> notionals
    """
    float_leg = swap.leg(1)
    t = ql.Actual365Fixed().yearFraction(today,data)
    n = len(float_leg)
    float_times = []
    float_dcf = []
    accrual_start = []
    accrual_end = []
    notionals = []
    for i in range(n):
        cf = ql.as_floating_rate_coupon(float_leg[i])
        value_date = cf.referencePeriodStart()
        t_fix_i = timeFromReference(value_date)
        t_i = timeFromReference(cf.date())
        if t_fix_i >= t:
            iborIndex = cf.index()
            period_end = cf.referencePeriodEnd()
            float_dcf.append(cf.accrualPeriod())
            accrual_start.append(t_fix_i)
            accrual_end.append(timeFromReference(period_end))
            float_times.append(t_i)
            notionals.append(cf.nominal())
    return np.array(float_times), np.array(float_dcf), np.array(accrual_start), np.array(accrual

def swapPathNPV(swap, data):
    """
        Input:
        - swap: ql.VanillaSwap Object
        - data: reference date

        Returns:
        function calc(x_t)
            input:
            - x_t --> level of the state variable (short rate)
    """
    fixed_times, fixed_amounts = getfixed(swap, data)
    float_times,float_dcf,accrual_start,accrual_end,notionals=getfloating(swap,data)
    t = ql.Actual365Fixed().yearFraction(today,data)
    df_times = np.concatenate([fixed_times, accrual_start, accrual_end, float_times])
    df_times = np.unique(df_times)
    fix_idx = np.in1d(df_times, fixed_times, True)
    fix_idx = fix_idx.nonzero()
    float_idx = np.in1d(df_times, float_times, True)
    float_idx = float_idx.nonzero()
    accrual_start_idx = np.in1d(df_times, accrual_start, True)
    accrual_start_idx = accrual_start_idx.nonzero()
    accrual_end_idx = np.in1d(df_times, accrual_end, True)
    accrual_end_idx = accrual_end_idx.nonzero()
    def calc(x_t):
        discount = np.vectorize(lambda T: model.zerobond(T, t, x_t))
        dfs = discount(df_times)
        fix_leg_npv = np.sum(fixed_amounts * dfs[fix_idx])
        index_fixings = (dfs[accrual_start_idx] / dfs[accrual_end_idx] - 1)
        index_fixings /= float_dcf
        float_leg_npv = np.sum(notionals * index_fixings * float_dcf * dfs[float_idx])
        npv = float_leg_npv - fix_leg_npv
        return npv ,float_leg_npv, fix_leg_npv
    return calc

def short_rate_path(time_grid):
    """
     Input:
    - time_grid: simulation times in years

    Returns:
    r_t --> one path for the short rate (Hull-White)
    r_std --> standardized variable (useful later for approximation)
    numeraire --> The numeraire for the T-measure (the ql.Gsr process is written under the T-For
    """
    m = len(time_grid)
    r_t = np.zeros(m)
    r_std = np.zeros(m)
    numeraire = np.zeros(m)
    for i in range(1, m):
        numeraire[0] = model.numeraire(0,0)
        t0 = time_grid[i-1]
        t1 = time_grid[i]
        e = process.expectation(t0, r_t[i-1], dt[i-1])
        std = process.stdDeviation(t0, r_t[i-1], dt[i-1])
        r_t[i] = np.random.normal(loc= e, scale= std)
        e_0_0 = process.expectation(0,0,t1)
        std_0_0 = process.stdDeviation(0,0,t1)
        r_std[i] = (r_t[i] - e_0_0) / std_0_0
        numeraire[i] = model.numeraire(t1, r_std[i])
    return r_t, r_std, numeraire

def swapSimulation(nPath):
    """
    Input:
    - nPath: number of paths for Monte Carlo

    Returns:
```

```
    - swapNPV: future values of the swap
    - flt: future values of the floating leg
    - fix: future values of the fixed leg
    - numeraires: numeraires for all paths
    - r_std: standardized short rate
    - r_t: short rate simulation
    """
    m = len(time_grid)
    r_t = np.zeros((nPath, m))
    r_std = np.zeros((nPath, m))
    numeraires = np.zeros((nPath, m))
    swapNPV = np.zeros((nPath, m))
    flt = np.zeros((nPath, m))
    fix = np.zeros((nPath, m))
    for i in range(nPath):
        r_t[i], r_std[i], numeraires[i]= short_rate_path(time_grid)
    for i in range(len(date_grid)):
        swapNPV[:,i],flt[:,i],fix[:,i] = np.vectorize(swapPathNPV(swap, date_grid[i]))(r_std[:,i
    return swapNPV, flt, fix, numeraires, r_std, r_t

def LSM_american(nPath):
    """
    Input:
    - nPath: number of paths

    Returns:
    - npv_amc: Least Squares Monte Carlo valuation
    - stDev_percentage: Standard deviation of the simulation
    - exerciseProbability: exercise probability of the option
    """
    swapNPV, flt, fix, numeraires, y, r = swapSimulation(nPath)
    payoff = swapNPV[:,-1]/numeraires[:,-1]
    payoff[payoff < 0] = 0
    V = np.zeros_like(y)
    exerciseProbability = np.zeros_like(y)
    V[:,-1] = payoff
    m = len(time_grid) -1
    ols_res = []
    for i in range(m-1, 0, -1):
        exercise_values = swapNPV[:,i] / numeraires[:,i]
        exercise_values[exercise_values < 0] = 0
        states = y[:, i]
        Y = np.column_stack((states, states**2, states**3, states**4, states**5))
        Y = sm.add_constant(Y)
        ols = sm.OLS(V[:,i+1], Y)
        ols_result = ols.fit()
        ols_res.append(ols_result)
        cont_value_hat = np.sum(ols_result.params * Y, axis=1)
        exerciseProbability[:,i]= np.where(exercise_values > cont_value_hat,1,0)
        V[:,i] = np.maximum(cont_value_hat, exercise_values)
    npv_amc =  np.sum(V[:,1]*numeraires[0,0]) / nPath
    stDev_simulation = np.sqrt((np.sum((V[:,1]*numeraires[0,0] - npv_amc)**2))/nPath)
    stDev_percentage = stDev_simulation / npv_amc
    exerciseProbability = np.sum(exerciseProbability, axis = 0)/nPath
    return npv_amc, stDev_percentage, exerciseProbability
```

Curve and index definition, today's date is fixed at 19 February 2020 and the forward curve in this case is chosen to be flat at 3%.

```
today = ql.Date(19, ql.February, 2020)
ql.Settings.instance().setEvaluationDate(today)
rate = ql.SimpleQuote(0.03)
rate_handle = ql.QuoteHandle(rate)
daycount = ql.Actual365Fixed()
yieldTermStructure = ql.FlatForward(today, rate_handle, daycount)
yieldTermStructure.enableExtrapolation()
handleYieldTermStructure = ql.RelinkableYieldTermStructureHandle(yieldTermStructure)
t0_curve = ql.YieldTermStructureHandle(yieldTermStructure)
index = ql.Euribor6M(handleYieldTermStructure)
calendar = ql.TARGET()
daycounter = yieldTermStructure.dayCounter()
timeFromReference = timeFromReferenceFactory(daycounter, today)
```

QuantLib Gsr model setup, with constant mean reversion and time dependent σ calibrated to co-terminal swaptions as shown in Chapter 4

```
vola = np.array((0.0050,0.0050,
                 0.0054,0.0057,
```

```python
                 0.0060,0.0061,
                 0.0061,0.0065,
                 0.0068,0.0071,0.0071))

volahw = [ql.QuoteHandle(ql.SimpleQuote(v)) for v in vola]

calibrationDates = [today + ql.Period("1Y"),
                          today + ql.Period("2Y"),
                          today + ql.Period("3Y"),
                          today + ql.Period("4Y"),
                          today + ql.Period("5Y"),
                          today + ql.Period("6Y"),
                          today + ql.Period("7Y"),
                          today + ql.Period("8Y"),
                          today + ql.Period("9Y"),
                          today + ql.Period("40Y")]

meanRev = [ql.QuoteHandle(ql.SimpleQuote(0.03))]

model = ql.Gsr(handleYieldTermStructure, calibrationDates, volahw, meanRev, 31.)
process = model.stateProcess()
```

Underlying swap features

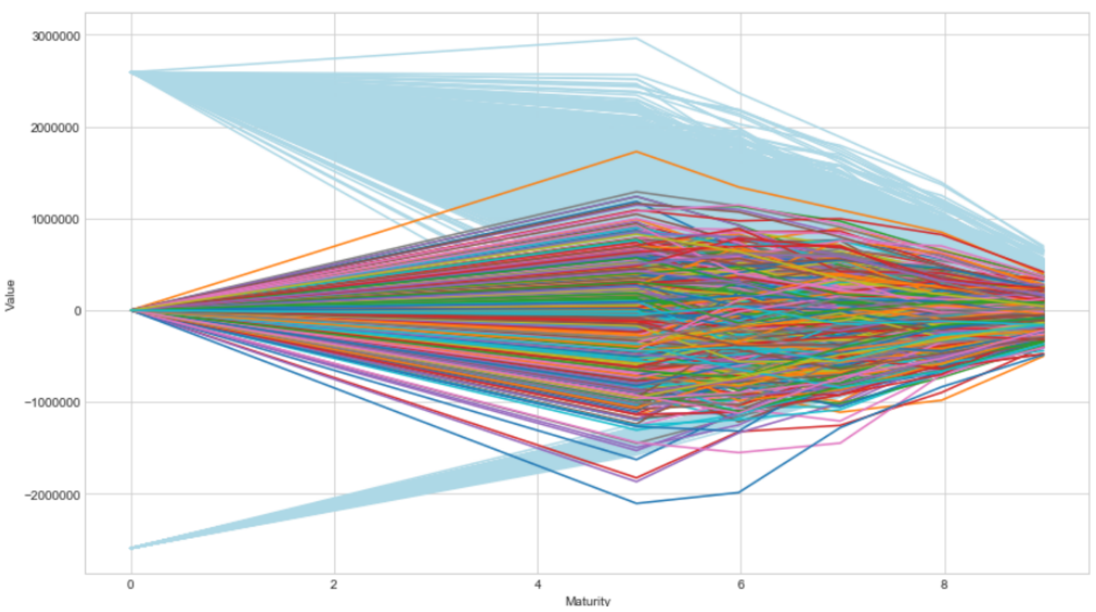
```python
notional = 10e6
fixedRate = 0.030480359077742216  #ATM rate
settlementDate = calendar.advance(today, ql.Period("2d"))
maturity = settlementDate + ql.Period("10Y")

swap = makeSwap(settlementDate, maturity, notional, fixedRate, index)
callability = ql.Period("-10d")
swap_fixed_schedule = list(swap.fixedSchedule())
calldates = [calendar.advance(d, callability) for d in swap_fixed_schedule[5:-1]]
date_grid = [today] + calldates
date_grid = np.unique(np.sort(date_grid))
time_grid = np.vectorize(lambda x: ql.Actual365Fixed().yearFraction(today, x))(date_grid)
dt = time_grid[1:] - time_grid[:-1]
# Check with QuantLib
engine = ql.DiscountingSwapEngine(handleYieldTermStructure)
swap.setPricingEngine(engine)
```

```
npv,_,_ = swapPathNPV(swap, date_grid[0])(0)
print("Swap NPV a t0: %.4f" % npv)
print("Quantlib NPV a t0: %.4f" % swap.NPV())
print("Errore : %.8f" % (npv - swap.NPV()))

[Out]: Swap NPV a t0: 0.0000
       Quantlib NPV a t0: 0.0000
       Errore : 0.00000000
```

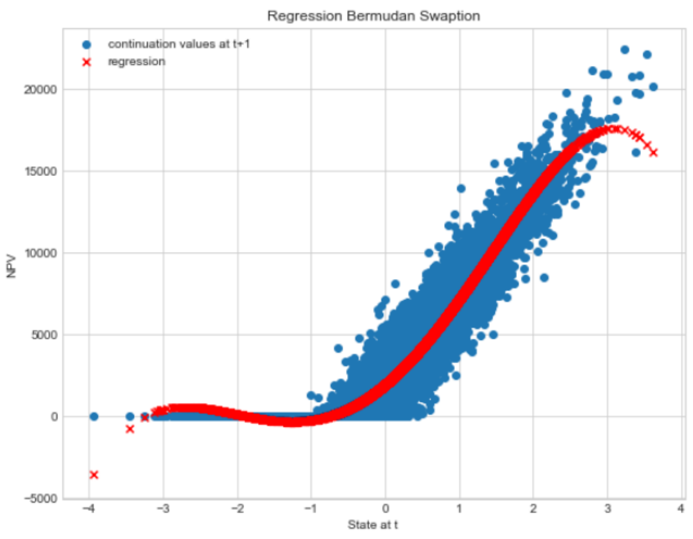
```
np.random.seed(1)
npv_amc, st_dev, exerciseProbability = LSM_american(1000)
print("NPV american option: %.2f" % npv_amc)
print("Standard deviation:  %.2f%%" % (st_dev))

[Out]: NPV american option: 144720.75
       Standard deviation:  1.33%
```

Plotted swap simulation

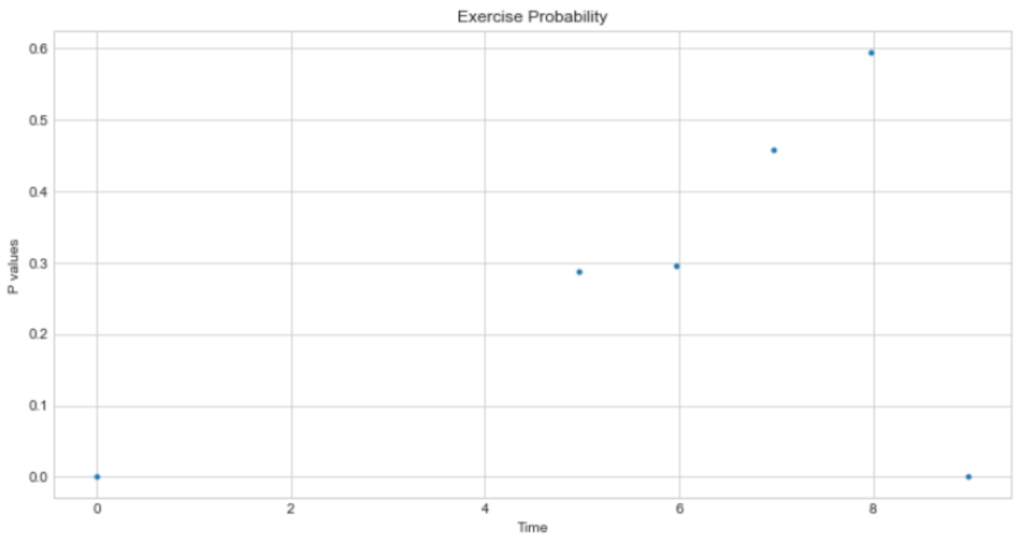
```
swapNPV, flt, fix,_,_,_  = swapSimulation(1000)
fig = plt.figure(figsize=(14,8))
plt.plot(time_grid, -fix.T, color = "lightblue")
plt.plot(time_grid, flt.T, color = "lightblue")
plt.plot(time_grid, swapNPV.T)
plt.xlabel("Maturity")
plt.ylabel("Value")
plt.show()
```


3000000
2000000
1000000
0
−1000000
−2000000
Value
0
2
4
6
8
Maturity

```python
swapNPV, flt, fix, numeraires, y, r = swapSimulation(10000)
payoff = swapNPV[:,-1]/numeraires[:,-1]
payoff[payoff < 0] = 0
V = np.zeros_like(y)
V[:,-1] = payoff
ols_res = []
payoff = swapNPV[:,-1]/numeraires[:,-1]
payoff[payoff < 0] = 0
states = y[:, -2]
Y = np.column_stack((states, states**2, states**3, states**4))
Y = sm.add_constant(Y)
ols = sm.OLS(payoff, Y)
ols_result = ols.fit()
cont_value_hat = np.sum(ols_result.params * Y, axis=1)

plt.figure(figsize=(9,7))
plt.scatter(states, payoff)
plt.scatter(states, cont_value_hat, marker='x', c='red')
plt.legend(["continuation values at t+1", "regression"])
plt.title("Regression Bermudan Swaption")
plt.xlabel("State at t")
plt.ylabel("NPV")
plt.show()
```



```python
plt.figure(figsize=(12,6))
plt.plot(time_grid, exerciseProbability, ".")
plt.title("Exercise Probability")
plt.xlabel("Time")
plt.ylabel("P values")
plt.show()
```



A.2 LSM pricing with G2++

```python
import xlwings as xw
import pandas as pd
import numpy as np
import QuantLib as ql
import statsmodels.api as sm
from scipy.stats import multivariate_normal
from numba import jit
```

Hermite polynomials

```python
def herm2(x):
    return (x*x) -1

def herm3(x):
    return (x*x*x) - 3*x

def herm4(x):
    return (x*x*x*x) - 6*(x*x) +3

def herm5(x):
    return (x**5) -10*(x**3) +15*x
```

Model and yield curve definition, today's date is fixed at 19 February 2020 and the forward curve is chosen to be flat at 3%

```python
a = 0.03
sigma = 0.0059
b = 0.50
eta = 0.0203
rho = -0.70

today = ql.Date(19, ql.February, 2020)
ql.Settings.instance().setEvaluationDate(today)
rate = ql.SimpleQuote(0.03)
rate_handle = ql.QuoteHandle(rate)
daycount = ql.Actual365Fixed()
```

```python
yieldTermStructure = ql.FlatForward(today, rate_handle, daycount)
yieldTermStructure.enableExtrapolation()
handleYieldTermStructure = ql.RelinkableYieldTermStructureHandle(yieldTermStructure)
t0_curve = ql.YieldTermStructureHandle(yieldTermStructure)
index = ql.Euribor6M(handleYieldTermStructure)
calendar = ql.TARGET()
daycounter = yieldTermStructure.dayCounter()
```

Swap functions

```python
def timeFromReferenceFactory(daycounter, ref):
    def impl(dat):
        return daycounter.yearFraction(ref, dat)
    return np.vectorize(impl)

def makeSwap(start, maturity, notional, fixedRate, index, tipo = ql.VanillaSwap.Payer):
    """
    Returns the Quantlib swap object
    """
    fixedLegTenor = ql.Period("6m")
    fixedLegBDC = ql.ModifiedFollowing
    floatLegTenor = index.tenor()
    fixedLegDC = index.dayCounter() #ql.Thirty360(ql.Thirty360.BondBasis)
    spread = 0.0
    fixedSchedule = ql.Schedule(start, maturity, fixedLegTenor, index.fixingCalendar(), fixedLeg
                                ql.DateGeneration.Backward, False)
    floatSchedule = ql.Schedule(start, maturity, floatLegTenor, index.fixingCalendar(), index.bu
                                 index.businessDayConvention(), ql.DateGeneration.Backward,Fa
    swap = ql.VanillaSwap(tipo, notional,fixedSchedule,fixedRate,fixedLegDC,floatSchedule,index,
    return swap
def getfixed(swap, data):
    ###### fixed leg #####
    date_temp = calendar.advance(data, -callability)
    time_temp = ql.Actual365Fixed().yearFraction(today,date_temp)
    t = time_temp
    fixed_leg = swap.leg(0)
    n = len(fixed_leg)
    fixed_times=[]
    fixed_amounts=[]
    if n == len(date_grid):
        for i in range(n):
            refDate = calendar.advance(date_grid[i], callability)
            cf = fixed_leg[i]
            cf_time = daycounter.yearFraction(refDate, cf.date() )
            t_i = timeFromReference(cf.date())
            if t_i > t:
                fixed_times.append(t_i)
                fixed_amounts.append(cf.amount())
        fixed_times = np.array(fixed_times)
        fixed_amounts = np.array(fixed_amounts)
    else:
        for i in range(n):
            cf = fixed_leg[i]
            t_i = timeFromReference(cf.date())
            if t_i > t:
                fixed_times.append(t_i)
                fixed_amounts.append(cf.amount())
        fixed_times = np.array(fixed_times)
        fixed_amounts = np.array(fixed_amounts)
    return fixed_times, fixed_amounts

def getfloating(swap, data):
    float_leg = swap.leg(1)
    t = ql.Actual365Fixed().yearFraction(today,data)
    n = len(float_leg)
    float_times = []
    float_dcf = []
    accrual_start_time = []
    accrual_end_time = []
    nominals = []
    for i in range(n):
        cf = ql.as_floating_rate_coupon(float_leg[i])
        value_date = cf.referencePeriodStart()
        t_fix_i = timeFromReference(value_date)
        t_i = timeFromReference(cf.date())
        if t_fix_i >= t:
            iborIndex = cf.index()
            index_mat = cf.referencePeriodEnd()
            float_dcf.append(cf.accrualPeriod())
            accrual_start_time.append(t_fix_i)
```

```
                accrual_end_time.append(timeFromReference(index_mat))
                float_times.append(t_i)
                nominals.append(cf.nominal())
        return np.array(float_times), np.array(float_dcf), np.array(accrual_start_time), np.array(ac
```

G2++ process simulation

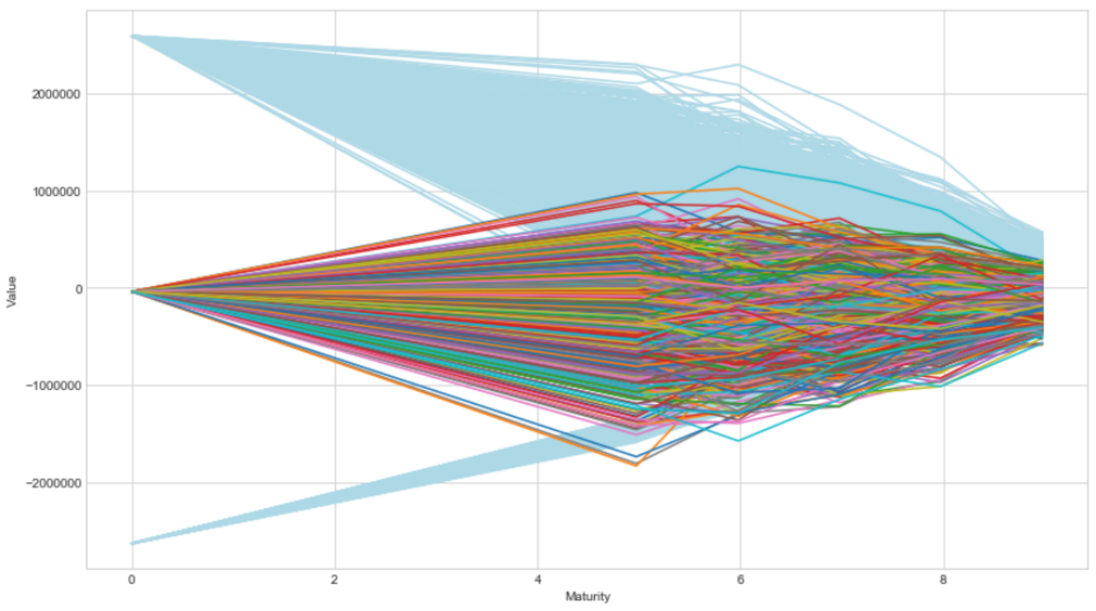
```
@jit(nopython=True)
def phi(f, t):  # pagina 146 Brigo-Mercurio
    exp_at = 1- np.exp(-a*t)
    exp_bt = 1- np.exp(-b*t)
    phi_t = f + (sigma**2/(2*(a*a))) * (exp_at**2)
    + ((eta*eta)/(2*(b*b))) * (exp_bt**2) + rho *((sigma*eta)/(a*b))*exp_at*exp_bt
    return phi_t

@jit(nopython=True)
def computeMx(s, t, T):
    efactor = 1 - np.exp(-a*(t-s))
    M = (((sigma*sigma)/(a*a)) + rho*((sigma*eta)/(a*b)))*efactor
    M += -((sigma*sigma)/(2*a*a))*(np.exp(-a*(T-t)) - np.exp(-a*(T + t - 2*s)))
    M += -((rho*sigma*eta)/(b*(a+b)))*(np.exp(-b*(T-t)) - np.exp(-b*T - a*t + (a+b)*s))
    return M

@jit(nopython=True)
def computeMy(s, t, T):
    dt = t - s
    efactor = 1 - np.exp(-b*(t-s))
    M = (((eta*eta)/(b*b)) + rho*((sigma*eta)/(a*b)))*efactor
    M += -((eta*eta)/(2*b*b))*(np.exp(-b*(T-t)) - np.exp(-b*(T + t - 2*s)))
    M += -((rho*sigma*eta)/(a*(a+b)))*(np.exp(-a*(T-t)) - np.exp(-a*T - b*t + (a+b)*s))
    return M

@jit(nopython=True)
def B(s, t, z):
    tau = t - s
    b = (1-np.exp(-z*tau))/z
    return b

@jit(nopython=True)
def E_xt(s, t, T, x_s):
    dt = t - s
    mean = x_s * np.exp(-a*dt) - computeMx(s,t,T)
    return mean

@jit(nopython=True)
def var_xt(s,t):
    var = (sigma*sigma)*B(s, t, 2*a)
    return var

@jit(nopython=True)
def E_yt(s, t, T, y_s):
    dt = t-s
    mean = y_s * np.exp(-b*dt) - computeMy(s,t,T)
    return mean

@jit(nopython=True)
def var_yt(s,t):
    var = (eta*eta)*B(s, t, 2*b)
    return var

@jit(nopython=True)
def cov_xy(s, t):
    cov = rho*sigma*eta*B(s, t, a+b)
    return cov
@jit(nopython = True)
def compute_V(t,T):
    dt = T-t
    v1 = ((sigma*sigma)/(a*a))*(dt +(2/a)*np.exp(-a*dt) - (1/(2*a))*np.exp(-2*a*dt) - 3/(2*a))
    v2 = ((eta*eta)/(b*b)) * (dt + (2/b)*np.exp(-b*dt) - (1/(2*b))*np.exp(-2*b*dt) - 3/(2*b))
    v12 = 2*rho*((sigma*eta)/(a*b))*(dt + (np.exp(-a*dt)-1)/a + (np.exp(-b*dt)-1)/b + (np.exp(-(
    v_t_T = v1 + v2 + v12
    return v_t_T

@jit(nopython = True)
def computeA(t, T, P0T, P0t):
    A_t_T = (P0T/P0t) * np.exp(0.5*(compute_V(t,T) - compute_V(0,T) + compute_V(0,t)))
    return A_t_T

@jit(nopython = True)
def computeB(z, t, T):
    B = (1 - np.exp(-z*(T-t)))/z
    return B

@jit(nopython = True)
def ZCB_price(t, T, P0T, P0t, x_t, y_t):
    P_t_T = computeA(t, T, P0T, P0t) * np.exp(- computeB(a,t,T)*x_t -  computeB(b,t,T)*y_t)
    return P_t_T

def numeraireGSR(t, x_t, y_t):
    P0t = t0_curve.discount(t, True)
    P0T = t0_curve.discount(31, True)
    num = ZCB_price(t, 31, P0T, P0t, x_t, y_t)
    return num

def simul(nPath, time_grid, phi_pmt):
    m = len(time_grid)
    r = np.zeros((nPath, m))
    x = np.zeros((nPath, m))
    y = np.zeros((nPath, m))
    x_std = np.zeros((nPath, m))
    y_std = np.zeros((nPath, m))
    r_std = np.zeros((nPath, m))
    r[:,0] = phi_pmt[0]
    T = 31
    for n in range(0,nPath):
        for i in range(1, m):
            s = time_grid[i-1]
            t = time_grid[i]
            E_x = E_xt(s, t, T, x[n, i-1])
            var_x = var_xt(s, t)
            E_y = E_yt(s, t, T, y[n, i-1])
            var_y = var_yt(s, t)
            cov = cov_xy(s, t)
            means = [E_x, E_y]
            cov_matrix = [[var_x, cov], [cov, var_y]]
            x[n,i], y[n,i] = np.random.multivariate_normal(means, cov_matrix)
            r[n,i] = x[n,i] + y[n,i] + phi_pmt[i]
            e_x0 = E_xt(0, t, T, 0)
            stdev_x0 = np.sqrt(var_xt(0, t))
            e_y0 = E_yt(0, t, T, 0)
            stdev_y0 = np.sqrt(var_yt(0, t))
            x_std[n,i] = (x[n,i] - e_x0)/stdev_x0
            y_std[n,i] = (y[n,i] - e_y0)/stdev_y0
            r_std[n,i] = x_std[n,i] + y_std[n,i] + phi_pmt[i]
    return x, y, r, x_std, y_std, r_std

def shortRateSimul(nPath, time_grid):
    m = len(time_grid)
    numeraire = np.zeros((nPath,m))
    phi_pmt = np.zeros(m)
    for i in range(0, len(time_grid)):
        t = time_grid[i]
        f = yieldTermStructure.forwardRate(t, t+0.002739726, ql.Continuous).rate()
        phi_pmt[i] = phi(f,t)
    m = len(time_grid)
    x, y, r, x_std, y_std, r_std = simul(nPath, time_grid, phi_pmt)
    numeraire[:,0] = numeraireGSR(0,0,0)
    for n in range(0,nPath):
        for i in range(1, m):
            numeraire[n,i] = numeraireGSR(t, x[n,i], y[n,i])
    return x, y, r, x_std, y_std, r_std, numeraire
```

```
def swapPathNPV2(swap, data):
    fixed_times, fixed_amounts = getfixed(swap, data)
    float_times, float_dcf, accrual_start_time, accrual_end_time, nominals = getfloating(swap, d
    t = ql.Actual365Fixed().yearFraction(today,data)
    df_times = np.concatenate([fixed_times, accrual_start_time, accrual_end_time, float_times])
    df_times = np.unique(df_times)
    fix_idx = np.in1d(df_times, fixed_times, True)
    fix_idx = fix_idx.nonzero()
    float_idx = np.in1d(df_times, float_times, True)
    float_idx = float_idx.nonzero()
    accrual_start_idx = np.in1d(df_times, accrual_start_time, True)
    accrual_start_idx = accrual_start_idx.nonzero()
    accrual_end_idx = np.in1d(df_times, accrual_end_time, True)
    accrual_end_idx = accrual_end_idx.nonzero()
    P0t = t0_curve.discount(t, True)
    def calc(x_t, y_t):
        dfs = np.zeros(len(df_times))
        for i, time in enumerate(df_times):
            P0T = t0_curve.discount(time, True)
            dfs[i] = ZCB_price(t, time, P0T, P0t, x_t, y_t)
        fix_leg_npv = np.sum(fixed_amounts * dfs[fix_idx])
        index_fixings = (dfs[accrual_start_idx] / dfs[accrual_end_idx] - 1)
        index_fixings /= float_dcf
        float_leg_npv = np.sum(nominals * index_fixings * float_dcf * dfs[float_idx])
        npv = float_leg_npv - fix_leg_npv
        return npv, float_leg_npv, fix_leg_npv
    return calc
def swapSimulation(nPath, time_grid):
    x, y, r, x_std, y_std, r_std, numeraire = shortRateSimul(nPath, time_grid)
    m = len(time_grid)
    swapNPV = np.zeros((nPath, m))
    flt = np.zeros((nPath, m))
    fix = np.zeros((nPath, m))
    for i in range(len(time_grid)):
        swapNPV[:,i], flt[:,i], fix[:,i] = np.vectorize(swapPathNPV2(swap, date_grid[i]))(x[:,i]
    return swapNPV, flt, fix, r, r_std, numeraire

def LSM_american(nPath, time_grid):
    swapNPV, flt, fix, r, r_std, numeraire = swapSimulation(nPath, time_grid)
    payoff = swapNPV[:,-1]/numeraire[:,-1]
    payoff[payoff < 0] = 0
    V = np.zeros_like(r)
    exerciseProbability = np.zeros_like(r)
    V[:,-1] = payoff
    m = len(time_grid) -1
    ols_res = []
    for i in range(m-1, 0, -1):
        exercise_values = swapNPV[:,i]/ numeraire[:,i]
        exercise_values[exercise_values < 0] = 0
        states = r_std[:, i]
        Y = np.column_stack((states, herm2(states), herm3(states), herm4(states)))
        Y = sm.add_constant(Y)
        ols = sm.OLS(V[:,i+1], Y)
        ols_result = ols.fit()
        ols_res.append(ols_result)
        cont_value_hat = np.sum(ols_result.params * Y, axis=1)
        exerciseProbability[:,i]= np.where(exercise_values > cont_value_hat,1,0)
        V[:,i] = np.maximum(cont_value_hat, exercise_values)
    npv_amc =  np.sum(V[:,1]*numeraire[0,0]) /nPath
    stDev_simulation = np.sqrt((np.sum((V[:,1]*numeraire[0,0] - npv_amc)**2))/nPath)
    stDev_percentage = stDev_simulation / npv_amc
    exerciseProbability = np.sum(exerciseProbability, axis = 0)/nPath
    return npv_amc, stDev_percentage, exerciseProbability
```

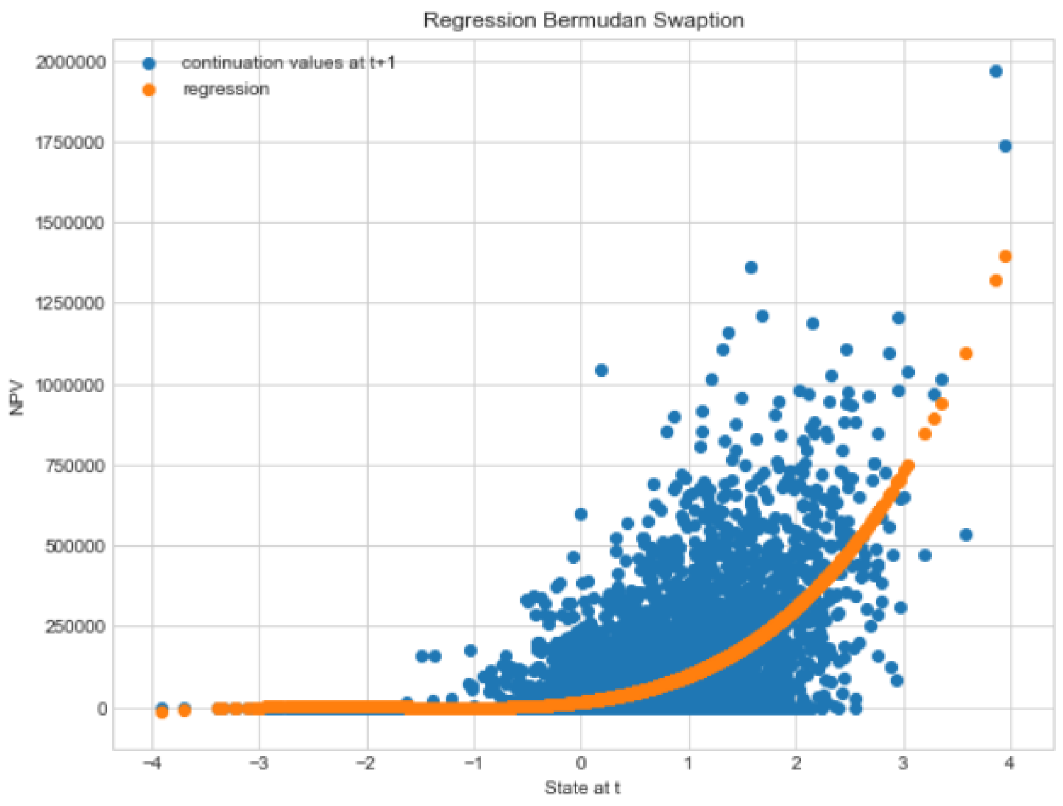
```
np.random.seed(1)
npv_amc, st_dev, exerciseProbability = LSM_american(1000, time_grid)
print("NPV american option: %.2f" % npv_amc)
print("Simul Standard deviation:  %.6f%%" % (st_dev))

[Out]: NPV american option: 113623.14
       Simul Standard deviation:  1.238026\%
```

Plotted swap simulation

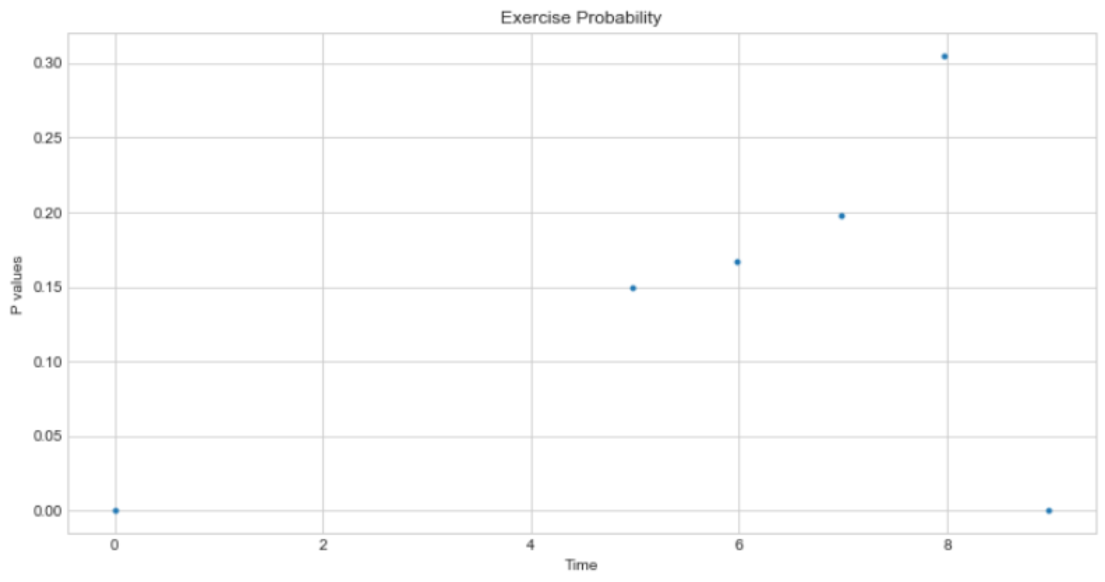
```
swapNPV, flt, fix, r, r_std, numeraire = swapSimulation(1000, time_grid)
fig = plt.figure(figsize=(14,8))
plt.plot(time_grid, -fix.T, color = "lightblue")
plt.plot(time_grid, flt.T, color = "lightblue")
plt.plot(time_grid, swapNPV.T)
plt.xlabel("Maturity")
plt.ylabel("Value")
plt.show()
```



```
swapNPV, flt, fix, r, r_std, numeraires = swapSimulation(10000, time_grid)
payoff = swapNPV[:,-1]/numeraires[:,-1]
payoff[payoff < 0] = 0
V = np.zeros_like(r)
V[:,-1] = payoff
payoff = swapNPV[:,-1]/numeraires[:,-1]
payoff[payoff < 0] = 0
states = r_std[:, -2]
Y = np.column_stack((states, herm2(states), herm3(states), herm4(states)))
Y = sm.add_constant(Y)
ols = sm.OLS(payoff, Y)
ols_result = ols.fit()
cont_value_hat = np.sum(ols_result.params * Y, axis=1)
plt.figure(figsize=(9,7))
plt.scatter(states, payoff)
plt.scatter(states, cont_value_hat)
plt.legend(["continuation values at t+1", "regression"])
plt.title("Regression Bermudan Swaption")
plt.xlabel("State at t")
plt.ylabel("NPV")
plt.show()
```



```python
plt.figure(figsize=(12,6))
plt.plot(time_grid, exerciseProbability, ".")
plt.title("Exercise Probability")
plt.xlabel("Time")
plt.ylabel("P values")
plt.show()
```


Exercise Probability
P values
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Time
0
2
4
6
8

A.3 Dynamic Delta Hedging with LSM

```python
import numpy as np
import pandas as pd
import QuantLib as ql
import matplotlib.pyplot as plt
import datetime
import statsmodels.api as sm
```

Use the same swap functions of the GSR model

```python
def short_rate_path(time_grid, dt):
    m = len(time_grid)
    r_t = np.zeros(m)
    r_std = np.zeros(m)
    numeraire = np.zeros(m)
    for i in range(1, m):
        numeraire[0] = model.numeraire(0,0)
        t0 = time_grid[i-1]
        t1 = time_grid[i]
        e = process.expectation(t0, r_t[i-1], dt[i-1])
        std = process.stdDeviation(t0, r_t[i-1], dt[i-1])
        r_t[i] = np.random.normal(loc= e, scale= std)
        e_0_0 = process.expectation(0,0,t1)
        std_0_0 = process.stdDeviation(0,0,t1)
        r_std[i] = (r_t[i] - e_0_0) / std_0_0
        numeraire[i] = model.numeraire(t1, r_std[i])
    return r_t, r_std, numeraire

def swapSimulation(nPath, swap, date_grid, time_grid, dt):
    m = len(time_grid)
    r_t = np.zeros((nPath, m))
    r_std = np.zeros((nPath, m))
    numeraires = np.zeros((nPath, m))
    swapNPV = np.zeros((nPath, m))
    flt = np.zeros((nPath, m))
    fix = np.zeros((nPath, m))
    for i in range(nPath):
        r_t[i], r_std[i], numeraires[i]= short_rate_path(time_grid, dt)
    for i in range(len(date_grid)):
        swapNPV[:,i],flt[:,i],fix[:,i],_ = np.vectorize(swapPathNPV(swap, date_grid[i]))(r_std[:
    return swapNPV, flt, fix, numeraires, r_std, r_t

def LSM_american(nPath, swap, date_grid, time_grid, dt):
    swapNPV, flt, fix, numeraires, y, r = swapSimulation(nPath, swap, date_grid, time_grid, dt)
    payoff = swapNPV[:,-1]/numeraires[:,-1]
    payoff[payoff < 0] = 0
    V = np.zeros_like(y)
    exerciseProbability = np.zeros_like(y)
    V[:,-1] = payoff
    m = len(time_grid) -1
    ols_res = []
    for i in range(m-1, 0, -1):
        exercise_values = swapNPV[:,i] / numeraires[:,i]
        exercise_values[exercise_values < 0] = 0
        states = y[:, i]
        Y = np.column_stack((states, states**2, states**3, states**4))
        Y = sm.add_constant(Y)
        ols = sm.OLS(V[:,i+1], Y)
        ols_result = ols.fit()
        ols_res.append(ols_result)
        cont_value_hat = np.sum(ols_result.params * Y, axis=1)
        exerciseProbability[:,i]= np.where(exercise_values > cont_value_hat,1,0)
        V[:,i] = np.maximum(cont_value_hat, exercise_values)
    npv_amc =  np.sum(V[:,1]*numeraires[0,0]) / nPath
    stDev_simulation = np.sqrt((np.sum((V[:,1]*numeraires[0,0] - npv_amc)**2))/nPath)
    stDev_percentage = stDev_simulation / npv_amc
    exerciseProbability = np.sum(exerciseProbability, axis = 0)/nPath
    return npv_amc, stDev_percentage, exerciseProbability

def deltaNum(nPath, date_grid, time_grid, dt):
    np.random.seed(1000)
    spreadUP = ql.SimpleQuote(1e-4)
    handleYieldTermStructure.linkTo(ql.ZeroSpreadedTermStructure(t0_curve, ql.QuoteHandle(spread
    index = ql.Euribor6M(handleYieldTermStructure)
    model = ql.Gsr(handleYieldTermStructure, calibrationDates, volahw, meanRev, 31.)
    process = model.stateProcess()
    swap = makeSwap(date_grid[0], maturity, notional, fixedRate, index)
    npv_up, _,_ = LSM_american(nPath, swap, date_grid, time_grid, dt)
```

```python

    np.random.seed(1000)
    spreadDOWN = ql.SimpleQuote(-1e-4)
    handleYieldTermStructure.linkTo(ql.ZeroSpreadedTermStructure(t0_curve, ql.QuoteHandle(spread
    index = ql.Euribor6M(handleYieldTermStructure)
    model = ql.Gsr(handleYieldTermStructure, calibrationDates, volahw, meanRev, 31.)
    process = model.stateProcess()
    swap = makeSwap(date_grid[0], maturity, notional, fixedRate, index)
    npv_down, _,_ = LSM_american(nPath, swap, date_grid, time_grid, dt)
    handleYieldTermStructure.linkTo(yieldTermStructure)
    return (npv_up - npv_down)/2
```

Initial portfolio setup, Chapter 4 swap hedging portfolio.

```python
today = ql.Date(19, ql.February, 2020)
ql.Settings.instance().setEvaluationDate(today)
rate = ql.SimpleQuote(0.03)
rate_handle = ql.QuoteHandle(rate)
daycount = ql.Actual365Fixed()
yieldTermStructure = ql.FlatForward(today, rate_handle, daycount)
yieldTermStructure.enableExtrapolation()
handleYieldTermStructure = ql.RelinkableYieldTermStructureHandle(yieldTermStructure)
t0_curve = ql.YieldTermStructureHandle(yieldTermStructure)
index = ql.Euribor6M(handleYieldTermStructure)
calendar = ql.TARGET()
daycounter = yieldTermStructure.dayCounter()
timeFromReference = timeFromReferenceFactory(daycounter, today)

volahw = [ql.QuoteHandle(ql.SimpleQuote(0.0020)), ql.QuoteHandle(ql.SimpleQuote(0.0020)) ]
calibrationDates = [today + ql.Period("1Y")]
meanRev = [ql.QuoteHandle(ql.SimpleQuote(0.03))]
model = ql.Gsr(handleYieldTermStructure, calibrationDates, volahw, meanRev, 31.)
process = model.stateProcess()

notional = 1e6
fixedRate = 0.028
settlementDate = calendar.advance(today, ql.Period("2d"))
maturity = settlementDate + ql.Period("10Y")

swap = makeSwap(settlementDate, maturity, notional, fixedRate, index)
```

```python
callability = ql.Period("-10d")
swap_fixed_schedule = list(swap.fixedSchedule())
calldates = [calendar.advance(d, callability) for d in swap_fixed_schedule[5:-1]]
date_grid = [today] + calldates
date_grid = np.unique(np.sort(date_grid))
time_grid = np.vectorize(lambda x: ql.Actual365Fixed().yearFraction(today, x))(date_grid)
dt = time_grid[1:] - time_grid[:-1]

hedging_frequency = ql.Period("6m")
first_hedge = settlementDate + hedging_frequency
HedgingSchedule = ql.Schedule(first_hedge, settlementDate + ql.Period("3y"),
                              hedging_frequency, ql.TARGET(), ql.ModifiedFollowing,
                                   ql.ModifiedFollowing, ql.DateGeneration.Forward, False)
hedgingDates = list(HedgingSchedule)
hedgingDates = [today] + hedgingDates
hedgingTimes = [daycounter.yearFraction(today, d) for d in hedgingDates]
hedgingTimes = np.array(hedgingTimes)
dt_hedge = hedgingTimes[1:] - hedgingTimes[:-1]
risk_free = 0.030454533953516938
nPath = 750
t = len(hedgingTimes)
delta = np.zeros(t)
hedging_portfolio = np.zeros(t)
option_value = np.zeros(t)
bpv_path = np.zeros(t)
swap_path = np.zeros(t)
salva_bond = np.zeros(t)
salva_cash = np.zeros(t)
np.random.seed(1000)
option_value[0],_,ex = LSM_american(nPath, swap, date_grid, time_grid, dt)

np.random.seed(1000)
delta[0] = deltaNum(nPath, date_grid, time_grid, dt)
swap_bpv = swapPathNPV(swap, hedgingDates[0])(0)[-1]*0.0001
hedge_weight = delta[0]/swap_bpv
bpv_path[0] = swap_bpv*hedge_weight
hedging_swap = makeSwap(settlementDate, maturity, hedge_weight, fixedRate, index)
initial_hedge_pv = swapPathNPV(hedging_swap, hedgingDates[0])(0)[0]
swap_path[0] = initial_hedge_pv
bond = option_value[0]- initial_hedge_pv
```

```
salva_bond[0] = bond
salva_cash[0] = bond
hedging_portfolio[0] = initial_hedge_pv + bond
## Long on 5x5 bermudan swaption, short on the same underlying swap
print( "Initial Portfolio")
print( "Option Value (buy): %8.3f" % option_value[0])
print(33*"-")
print("Delta Hedging Portfolio")
print( "Hedging swap (sell): %8.3f" % initial_hedge_pv)
print("Bond Funding position: %8.3f" % bond)
print(33*"-")
print("Portfolio Cash Value at t0: %8.3f" % (-option_value[0] + hedging_portfolio[0]))

Out:     Initial Portfolio
         Option Value (buy): 12691.762
         ---------------------------------
         Delta Hedging Portfolio
         Hedging swap (sell): 6976.003
         Bond Funding position: 5715.759
         ---------------------------------
         Portfolio Cash Value at t0:    0.000
```

```
short_rate, std_rate, _ = short_rate_path(hedgingTimes, dt_hedge)
cash_balance = 0
for i in range(1, t, 1):
    hedgeDate = hedgingDates[i]
    settlementDate = calendar.advance(hedgeDate, ql.Period("2d"))

    swap = makeSwap(settlementDate, maturity, notional, fixedRate, index)
    callability = ql.Period("-10d")
    date_grid = [hedgeDate] + calldates
    date_grid = np.unique(np.sort(date_grid))
    time_grid = np.vectorize(lambda x: ql.Actual365Fixed().yearFraction(hedgeDate, x))(date_grid
    dt = time_grid[1:] - time_grid[:-1]
    np.random.seed(1000)
    option_value[i] = LSM_american(nPath, swap, date_grid, time_grid, dt)[0]
    #delta
    np.random.seed(1000)
    hedging_swap = makeSwap(settlementDate, maturity, hedge_weight, fixedRate, index)
    swap_hedge,_,_,swap_bpv = swapPathNPV(hedging_swap, hedgeDate)(short_rate[i])
    hedging_portfolio[i] = swap_hedge + bond*np.exp(risk_free*dt_hedge[i-1]) # stesso portafogli
    delta[i] = deltaNum(nPath, date_grid, time_grid, dt)
    swap_bpv = swapPathNPV(swap, hedgeDate)(short_rate[i])[-1]*0.0001
    hedge_weight = delta[i]/swap_bpv
    bpv_path[i] = swap_bpv*hedge_weight
    swapPath = makeSwap(settlementDate, maturity, hedge_weight, fixedRate, index)
    swap_path[i] = swapPathNPV(swapPath, hedgeDate)(short_rate[i])[0]
    cash_balance = swap_path[i] - swap_hedge
    salva_cash[i] = bond*np.exp(risk_free*dt_hedge[i-1]) + cash_balance
    hedging_portfolio[i] = swap_path[i] + bond*np.exp(risk_free*dt_hedge[i-1]) + cash_balance
    bond = option_value[i]- swap_path[i]
    salva_bond[i]= bond

hedging_errors = option_value - hedging_portfolio

print ("Average Error %7.3f"% (np.sum(hedging_errors) / len(hedging_errors)))
print ("Total P&L %7.3f" % (np.sum(hedging_errors)))

Out:    Average Error -260.864
        Total P&L -1826.046
```

Appendix B

Basic Theory

Definition 1. ***$\mathcal{L}^2$ Space***
In the $\mathcal{L}^2(X, \mathbb{C})$ space the norm is induced by the interior product:

$$\langle f, g \rangle = \int_X f(x)g(x)dx$$

$\mathcal{L}^2$ space is the only Hilbert space among the $\mathcal{L}^p$ class. Hilbert spaces have orthonormal vector basis[1].

Definition 2. ***Stochastic Integral***
Consider a Wiener process W and a general stochastic process g, the process g belongs to the square-integrable class $\mathcal{L}^2[a, b]$ if:

- $\int_a^b \mathbb{E}[g^2(s)]ds < \infty$
- *g is adapted to $\mathcal{F}_t^W$*

The process g belongs to the general $\mathcal{L}^2$ if $g \in \mathcal{L}^2[0, t]$ for all $t > 0$.
Then the following equalities holds

$$\mathbb{E}\left[\int_a^b g(s)dW(s)\right] = 0$$

$$\mathbb{E}\left[\left(\int_a^b g(s)dW(s)\right)^2\right] = \int_a^b \mathbb{E}[g(s)]ds$$

[1] *a finite set of elements in the vector space, each element of the vector space can be rewritten as a finite linear combination of the elements of the basis*

and $\int_a^b g(s)dW(s)$ is $\mathcal{F}_b^W$ measurable

Definition 3. ***Martingale***

A general stochastic process X adapted to the filtration $\mathcal{F}_t$ is a martingale if for each $T > t$

$$\mathbb{E}[X(T)|\mathcal{F}_t] = X(t)$$

Definition 4. ***Equivalent Probability Measures***

Two probability measures are equivalent if they assign the same probability to certain and impossible events, let $\mathbb{P}$ and $\mathbb{Q}$ be probability measures defined on the probability space $(\Omega,\, \mathcal{F})$

$$\mathbb{P}(A) = 1 \iff \mathbb{Q}(A) = 1$$

for all $A \in \mathcal{F}$ an event that has strictly positive probability in $\mathbb{P}$ must have strictly positive probability also in $\mathbb{Q}$.

Definition 5. ***Girsanov Theorem***

Consider a normal random variable $z \sim N(0,1)$ with density function $f(z_t)$ and probability measure $\mathbb{P}$

$$d\mathbb{P}(z_t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(z_t)^2} dz_t$$

define the function

$$\xi(z_t) = e^{z_t\mu - \frac{1}{2}\mu^2}$$

that multiplied by $d\mathbb{P}(z_t)$ produces a new probability measure

$$\begin{aligned} d\mathbb{P}(z_t)\xi(z_t) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(z_t)^2 + \mu z_t - \frac{1}{2}\mu^2} dz_t \\ d\mathbb{Q}(z_t) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(z_t - \mu)^2} dz_t \end{aligned}$$

Definition 6. ***Simple Least Squares Regression***

Given a set of points $x_1, \ldots, x_m$ of an independent variable, define a model function $y = f(x, \beta)$ that best fit the data with respect to the parameter β_j

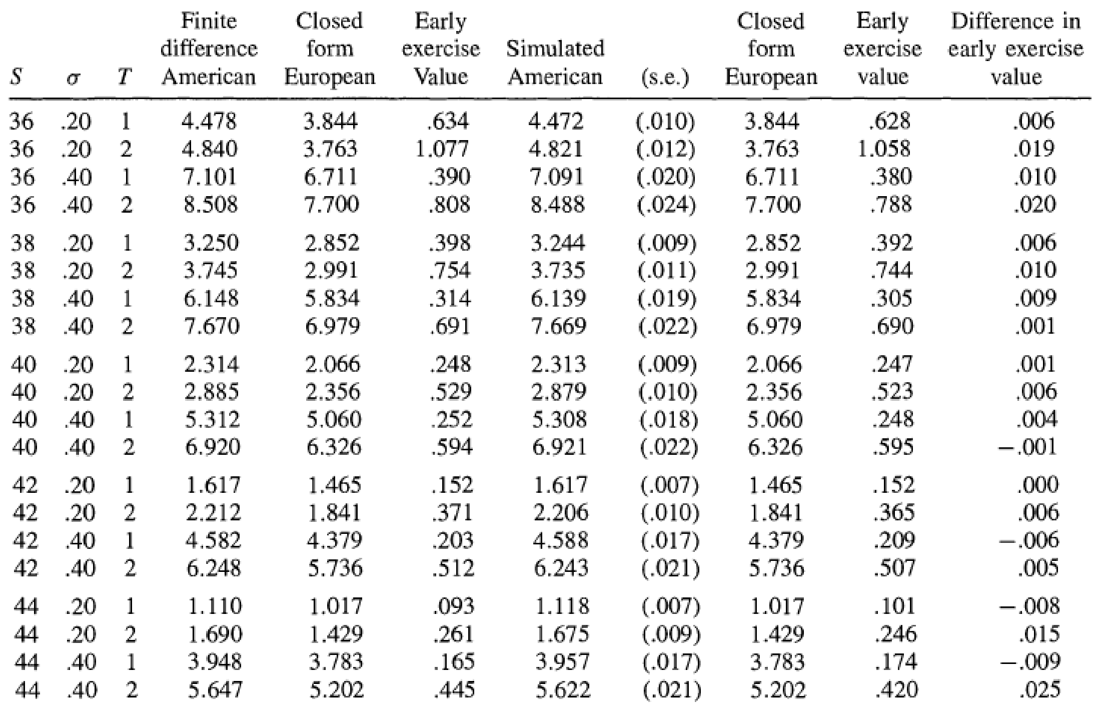
$$f(x, \beta) = \sum_{j=1}^{n} \beta_j \phi_j(x)$$

for a n polynomial function ϕ_j minimize the squared residuals

$$\sum_{i=1}^{m}(y_i - f(x_i, \beta))^2$$

Appendix C

List of Tables



S	σ	T	Finite difference American	Closed form European	Early exercise Value	Simulated American	(s.e.)	Closed form European	Early exercise value	Difference in early exercise value
36	.20	1	4.478	3.844	.634	4.472	(.010)	3.844	.628	.006
36	.20	2	4.840	3.763	1.077	4.821	(.012)	3.763	1.058	.019
36	.40	1	7.101	6.711	.390	7.091	(.020)	6.711	.380	.010
36	.40	2	8.508	7.700	.808	8.488	(.024)	7.700	.788	.020
38	.20	1	3.250	2.852	.398	3.244	(.009)	2.852	.392	.006
38	.20	2	3.745	2.991	.754	3.735	(.011)	2.991	.744	.010
38	.40	1	6.148	5.834	.314	6.139	(.019)	5.834	.305	.009
38	.40	2	7.670	6.979	.691	7.669	(.022)	6.979	.690	.001
40	.20	1	2.314	2.066	.248	2.313	(.009)	2.066	.247	.001
40	.20	2	2.885	2.356	.529	2.879	(.010)	2.356	.523	.006
40	.40	1	5.312	5.060	.252	5.308	(.018)	5.060	.248	.004
40	.40	2	6.920	6.326	.594	6.921	(.022)	6.326	.595	−.001
42	.20	1	1.617	1.465	.152	1.617	(.007)	1.465	.152	.000
42	.20	2	2.212	1.841	.371	2.206	(.010)	1.841	.365	.006
42	.40	1	4.582	4.379	.203	4.588	(.017)	4.379	.209	−.006
42	.40	2	6.248	5.736	.512	6.243	(.021)	5.736	.507	.005
44	.20	1	1.110	1.017	.093	1.118	(.007)	1.017	.101	−.008
44	.20	2	1.690	1.429	.261	1.675	(.009)	1.429	.246	.015
44	.40	1	3.948	3.783	.165	3.957	(.017)	3.783	.174	−.009
44	.40	2	5.647	5.202	.445	5.622	(.021)	5.202	.420	.025

Figure C.1: Table 1 from Longstaff and Schwartz paper

Table C.1: quoted instruments used for EONIA bootstrapping, (6 Feb. 2020, source: Bloomberg)

Term	**Rate Type**	**Mid**
1 DY	CASH	**-0.520**
2 DY	CASH	**-0.520**
1 WK	OIS SWAP	**-0.452**
2 WK	OIS SWAP	**-0.452**
3 WK	OIS SWAP	**-0.452**
1 MO	OIS SWAP	**-0.452**
2 MO	OIS SWAP	**-0.454**
3 MO	OIS SWAP	**-0.455**
4 MO	OIS SWAP	**-0.457**
5 MO	OIS SWAP	**-0.460**
6 MO	OIS SWAP	**-0.463**
7 MO	OIS SWAP	**-0.465**
8 MO	OIS SWAP	**-0.468**
9 MO	OIS SWAP	**-0.470**
10 MO	OIS SWAP	**-0.472**
11 MO	OIS SWAP	**-0.474**
12 MO	OIS SWAP	**-0.476**
15 MO	OIS SWAP	**-0.479**
18 MO	OIS SWAP	**-0.481**
21 MO	OIS SWAP	**-0.481**
2Y	OIS SWAP	**-0.480**
3Y	OIS SWAP	**-0.466**
4Y	OIS SWAP	**-0.440**
5Y	OIS SWAP	**-0.307**
6Y	OIS SWAP	**-0.364**
7Y	OIS SWAP	**-0.217**
8Y	OIS SWAP	**-0.263**
9Y	OIS SWAP	**-0.206**
10Y	OIS SWAP	**-0.149**
11Y	OIS SWAP	**-0.093**
12Y	OIS SWAP	**-0.039**
15Y	OIS SWAP	**0.105**
20Y	OIS SWAP	**0.247**
25Y	OIS SWAP	**0.299**
30Y	OIS SWAP	**0.303**

Table C.2: quoted instruments used for EURIBOR 6M bootstrapping, (6 Feb. 2020, source: Bloomberg)

Term	**Rate Type**	**Mid**
6 MO	CASH	**-0.343**
1 MO X 7 MO	SERIAL_SPOT_FRA	**-0.349**
2 MO X 8 MO	SERIAL_SPOT_FRA	**-0.355**
3 MO X 9 MO	SERIAL_SPOT_FRA	**-0.359**
4 MO X 10 MO	SERIAL_SPOT_FRA	**-0.363**
5 MO X 11 MO	SERIAL_SPOT_FRA	**-0.365**
6 MO X 12 MO	SERIAL_SPOT_FRA	**-0.367**
7 MO X 13 MO	SERIAL_SPOT_FRA	**-0.369**
8 MO X 14 MO	SERIAL_SPOT_FRA	**-0.368**
9 MO X 15 MO	SERIAL_SPOT_FRA	**-0.367**
10 MO X 16 MO	SERIAL_SPOT_FRA	**-0.365**
11 MO X 17 MO	SERIAL_SPOT_FRA	**-0.363**
12 MO X 18 MO	SERIAL_SPOT_FRA	**-0.359**
2Y	OIS SWAP	**-0.356**
3Y	OIS SWAP	**-0.334**
4Y	OIS SWAP	**-0.301**
5Y	OIS SWAP	**-0.262**
6Y	OIS SWAP	**-0.219**
7Y	OIS SWAP	**-0.172**
8Y	OIS SWAP	**-0.120**
9Y	OIS SWAP	**-0.064**
10Y	OIS SWAP	**-0.009**
12Y	OIS SWAP	**0.098**
14Y	OIS SWAP	**0.193**
15Y	OIS SWAP	**0.234**
20Y	OIS SWAP	**0.366**
25Y	OIS SWAP	**0.407**
30Y	OIS SWAP	**0.402**
35Y	OIS SWAP	**0.380**
40Y	OIS SWAP	**0.347**
50Y	OIS SWAP	**0.276**

Bibliography

[1] L. Andersen, V. Piterbarg (2010) Interest Rate Models. Vol.1, Vol.2, Vol.3.

[2] E. Derman, M.B. Miller, D. Park (2016) The Volatility Smile

[3] Glasserman,P. (2003). Monte Carlo Methods in Financial Engineering.

[4] D. Brigo, F. Mercurio (2006) Interest Rate Models - Theory and Practice (2nd edition).

[5] F.A Longstaff, E.S. Schwartz (2001) Valuing American Options by Simulation: A Simple Least-Squares Approach.

[6] C. Pacati (2016) A note on joint simulation of spot rates and stochastic discount factors in the G2 part of the G2++ model.

[7] P. Glasserman, B. Yu (2005) Number of paths versus number of basis functions in American Option pricing, Columbia University.

[8] Y. Hilpisch (2015) Derivatives Analytics with Python : Data Analysis, Models, Simulation, Calibration and Hedging.

[9] Quantitative Analytics Bloomberg L.P. (2018) Overview of Bloomberg Models for Interest Rate Derivatives in DLIB.

[10] R. Rebonato (2004) Volatility and Correlation 2nd Edition, The Perfect Hedger and the Fox.

[11] F. M. Ametrano, M. Bianchetti (2009) Bootstrapping the illiquidity, multiple yield curves construction for market coherent forward rates estimation.

[12] M. Morini (2011) Understanding and Managing Model Risk. A Practical Guide for Quants, Traders and Validators

[13] Ametrano F. M. Bianchetti M. (2013) Everything You Always Wanted to Know About Multiple Interest Rate Curve Bootstrapping But Were Afraid To Ask.

[14] P. Glasserman, B. Yu (2006) Simulation for American Options: Regression Now or Regression Later?

[15] P. Caspers (2013) One factor Gaussian short rate model implementation.

[16] F. Mercurio (2009) Interest Rates and The Credit Crunch: New Formulas and Market Models.

[17] J. H. Cochrane (2005) Asset pricing, Princeton University Press.

[18] G. Balaraman, L. Ballabio (2019) QuantLib Python Cookbook.

[19] R. Rebonato (2018) Bond pricing and yield-curve modelling, a structural approach.

[20] L. Andersen and J. Andreasen (2001) Factor dependence of Bermudan swaptions: fact or fiction?

[21] V. Piterbarg (2003) A Practitioner's Guide to Pricing and Hedging Callable Libor Exotics in Forward Libor Models.