Advanced Mathematical Models in Finance 2023/2024

POLITECNICO
MILANO 1863

Final Project

Structured Bond Pricing

Group 1
Emanuele Bellini
Francesco Sforzini
Edgard Enrique Sotelo Escobedo

Abstract

This report aims to analyse and document the functionality of a Python code designed for pricing a structured bond and computing its sensitivities. The primary objective is to price the upfront percentage X% at the bond issuance date and determine key sensitivities: underlying-Delta, Vega, and parallel-DV01. These tasks are crucial for understanding the financial instrument's behaviour under various market conditions and for implementing effective hedging strategies.

The code employs Monte Carlo simulation to model the dynamics of the underlying asset, ENEL stock, using Geometric Brownian Motion. It calculates the average performance of the stock over specified monitoring periods to determine the coupon payments. These future cash flows are then discounted to their present value using relevant zero-rates, from which the upfront percentage is derived.

Sensitivity analysis is performed by perturbing initial conditions: stock price for Delta, volatility for Vega, and interest rates for DV01. The code calculates the respective sensitivities by observing the changes in the bond price resulting from these perturbations. Additionally, the code provides hedging strategies to offset these sensitivities, using ENEL stocks, European call options, and interest rate swaps.

The results from this code enable accurate pricing of the structured bond, comprehensive risk assessment, and formulation of effective hedging strategies, ensuring robust financial management and risk mitigation for the structured bond.

Contents

Imported libraries

The code starts by importing the required libraries for data manipulation, date handling, plotting, and financial calculations. These libraries include Pandas, NumPy, datetime, and custom financial libraries. Follows the explanation of the custom libraries imported:

Datetime and Dateutil

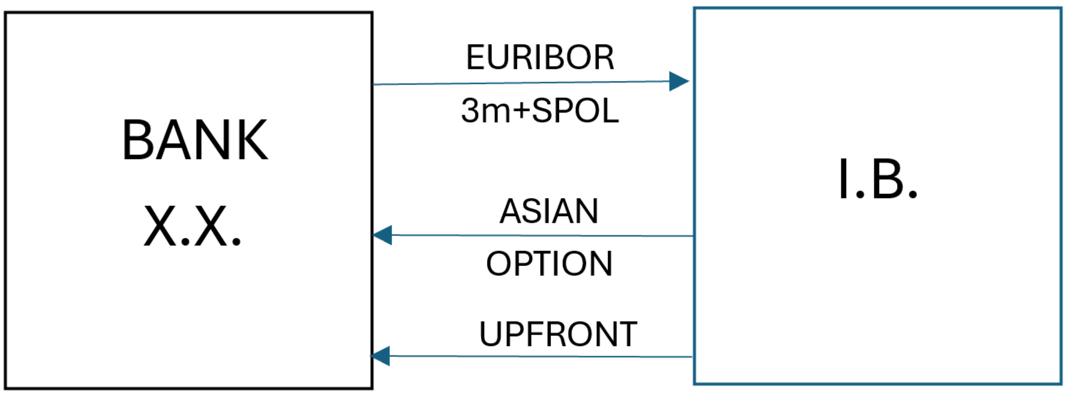
- **Date Definitions:** Defines trade, start, and maturity dates using datetime objects.
- **Relative Date Calculations:** Uses relativedelta to calculate the maturity date by adding a specific number of years to the start date.

Findates:

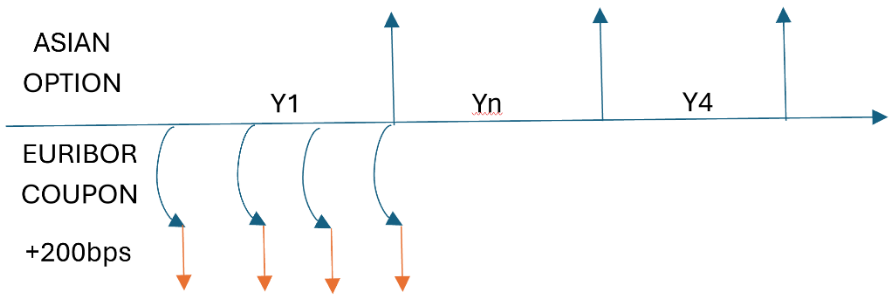
- **Year Fraction Calculation** calculates the year fraction between two dates using different day count conventions.
- **Date Rolling**: Adjusts dates to the next business day if they fall on a holiday.
- **Holiday Calendar**: Defines the holiday calendar for adjusting dates, ensuring accurate business day calculations.

1. Pricing the Upfront

The calculation of the upfront price for the structured bond involves several steps. The process starts with reading and interpolating market data, simulating stock price paths, calculating the average performance of the underlying asset, discounting future cash flows, and finally deriving the upfront percentage. This chapter details each of these steps as implemented in the provided Python code, ensuring a comprehensive understanding of the methodology used.



The flows expected of this bond are modeled like the following graph:



In particular, to price the instrument we will find the price of the coupons paid between the parties A (Bank XX) and B (IB). Then, we know that the flows from each party should have the same value so there is no arbitrage. Hence, the difference between the flows of each party is the upfront payment (X).

First, we interpolated linearly the zero rate curve for everyday obtaining the new curve, in the code called 'zeroRates_interp'.

Data preparation

The first step in computing the upfront price is to prepare the market data, which involves importing the discount factor curve and the zero rate curve from CSV files. These curves are essential for discounting future cash flows to their present values. The discount factors and zero rates are read into Pandas DataFrames, with dates being converted to datetime objects for proper handling. The zero-rate curve is particularly important as it is interpolated to create a continuous curve, ensuring that zero rates are available for every day within the relevant period. This interpolation is achieved using Pandas' interpolate method, providing a seamless zero rate curve for accurate discounting.

Bond features and pricing

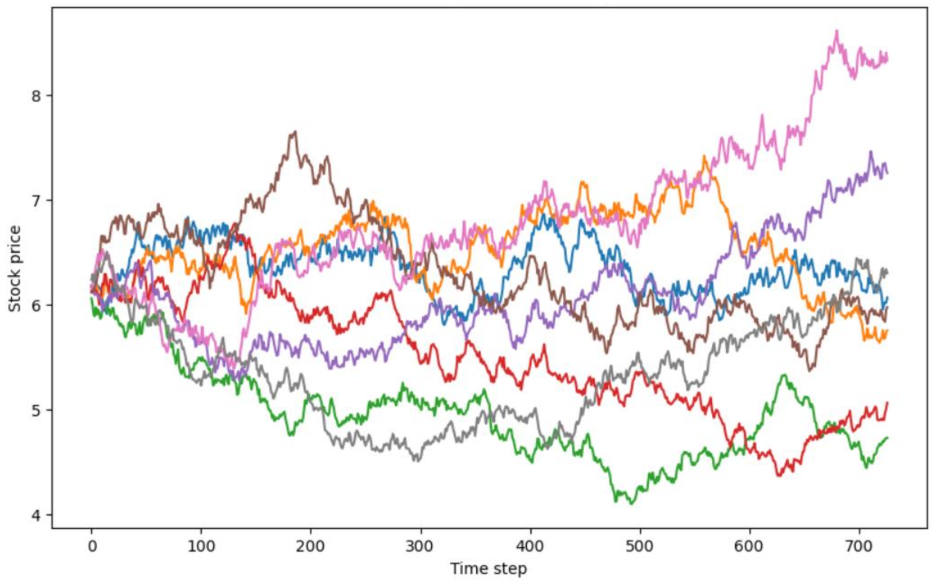
The bond characteristics, including the issue amount, trade date, start date, and maturity period, are defined next. The maturity date is calculated by adding the bond's maturity period to the start date. However, financial contracts typically require that all important dates, such as maturity dates, fall on business days. To ensure this, the rolldate function from the FinDates.busdayrule module is used. This function adjusts dates that fall on weekends or holidays to the nearest business day, following the 'follow' convention, which moves the date to the next business day if it falls on a non-business day.

Discount Factor Calculation

To discount future cash flows to their present value, the discount factor for the bond's maturity date is calculated. This involves retrieving the zero-rate corresponding to the maturity date from the interpolated zero rate curve. The zero rate is then converted to a discount factor using the year fraction calculated by the yearfrac function with the ACT/360 day count convention. The discount factor represents the present value of one unit of currency payable at the bond's maturity date.

Simulating Stock Price Paths

The structured bond's payoff depends on the performance of the underlying ENEL stock, necessitating the simulation of its future price paths. The code employs Geometric Brownian Motion (GBM) to model the stock price dynamics. GBM is a widely used stochastic process for modelling stock prices, incorporating both the drift and volatility of the stock. The generate_paths function simulates multiple price paths over the bond's maturity period. It iterates through each time step, updating the stock price based on the risk-free rate, stock volatility, and random shocks drawn from a normal distribution.



Calculating Average Performance

The bond pays coupons based on the average performance of the underlying stock over predefined monitoring periods. The calculate_average_performance function calculates this average performance for each simulated path. It computes the average price of the stock over the monitoring dates and then calculates the performance relative to the initial stock price. This average performance is crucial for determining the bond's coupon payments.

Discounting Future Cash Flows

Once the average performance is calculated, the next step is to compute the future cash flows, including the coupon payments, and discount them to their present value. The discount_cash_flows function achieves this by multiplying each cash flow by its corresponding discount factor, which is derived from the interpolated zero rate curve. The discounted cash flows are then summed to obtain the present value of all future payments.

Calculating the Upfront Percentage

The final step is to calculate the upfront percentage X%, which represents the bond's price at issuance. This is derived from the net present value of the discounted cash flows, adjusted for the bond's notional amount and the spread over Libor. The upfront percentage is computed by summing the present value of the coupons and the notional amount adjusted by the spread, and then dividing by the bond's principal amount.

Results

The Python code comprehensively implements the steps necessary to compute the upfront price of the structured bond. Starting with the preparation of market data and interpolation of zero rates, the code simulates stock price paths using Geometric Brownian Motion, calculates the average performance of the underlying asset, discounts the future cash flows to present value, and finally derives the upfront percentage.

This detailed and systematic approach ensures the valuation of the structured bond, at the end of which it was possible to define the upfront price: **13.73%.**

2. Sensitivities

The computation of sensitivities for the structured bond involves determining how the bond's price reacts to changes in various market parameters. Specifically, this chapter focuses on the calculation of Delta, Vega, and DV01 (Dollar Value of 01), which measure the bond's sensitivity to changes in the underlying asset price, volatility, and interest rates, respectively. These sensitivities are crucial for risk management and hedging strategies. This chapter details the steps involved in computing each of these sensitivities as implemented in the provided Python code.

Market Data Preparation

As with the upfront pricing, the initial step involves preparing the market data. This includes importing the discount factor curve and the zero rate curve, which are essential for discounting future cash flows and computing sensitivities. The same process of reading the data from CSV files, converting date columns to datetime objects, and interpolating zero rates is followed to ensure continuous zero rate availability.

Simulating Stock Price Paths

The sensitivity calculations require multiple simulations of stock price paths, both with and without perturbations to the underlying parameters. The generate_paths function is used to simulate these paths based on the initial stock price, risk-free rate, and stock volatility. This simulation forms the foundation for calculating the bond’s sensitivities.

Delta Calculation

Delta measures the sensitivity of the bond price to changes in the price of the underlying asset. The computation involves perturbing the initial stock price by a small amount (denoted as h_S) and recalculating the bond price for these new stock prices. The difference in the bond prices before and after the perturbation, divided by the perturbation amount, gives the Delta.

Vega Calculation

Vega measures the sensitivity of the bond price to changes in the volatility of the underlying asset. This is calculated by perturbing the volatility (σ) by a small amount (denoted as h_σ) and recalculating the bond price. The difference in the bond prices before and after the perturbation, divided by the perturbation amount, gives the Vega.

DV01 Calculation

DV01 (Dollar Value of 01) measures the sensitivity of the bond price to changes in interest rates. The computation involves perturbing the zero rates by a small amount (typically one basis point) and recalculating the bond price. The difference in the bond prices before and after the perturbation gives the DV01.

Results

The Python code implements the steps necessary to compute the sensitivities of the structured bond, specifically Delta, Vega, and DV01. By simulating stock price paths, perturbing relevant parameters (stock price, volatility, and interest rates), and recalculating the bond price, the code determines how the bond's value reacts to changes in market conditions. Each sensitivity measure provides valuable insights into the bond's risk profile, enabling the formulation of effective hedging strategies. This detailed approach ensures that the bond’s sensitivities are accurately captured, facilitating robust risk management.

- Delta: **38.68**
- Vega: **208.51**
- DV01: **382983€**

3. Hedging

Hedging is a crucial aspect of managing the risks associated with holding a structured bond. The primary goal of hedging is to offset the sensitivities of the bond to various market factors, such as changes in the price of the underlying asset, volatility, and interest rates. In this chapter, we will outline a framework for hedging the structured bond, focusing on Delta, Vega, and DV01 hedging. Although the original Python code does not provide a hedging strategy, we will discuss how these hedging strategies can be implemented conceptually and practically.

Delta Hedging and Vega Hedging

Delta measures the sensitivity of the bond's price to changes in the price of the underlying asset. To hedge Delta, we need to create an offsetting position in the underlying asset (ENEL stock). To hedge the bond's Delta, we should take an opposite position in the underlying asset. For instance, if the bond's Delta is positive, short the ENEL stock; if negative, buy the ENEL stock. Then, regularly rebalance the hedge to maintain a neutral Delta, especially as the price of the underlying asset changes or as time progresses.

Vega measures the sensitivity of the bond's price to changes in the volatility of the underlying asset. To hedge Vega, use options on the underlying asset, such as European call or put options. To hedge the bond's Vega, take an opposite position in options on the underlying asset. Typically, at-the-money (ATM) options are used for this purpose. Then, regularly rebalance the hedge to maintain a neutral Vega as market conditions change.

For hedging against delta and vega, we defined a linear system as follows

$$\begin{pmatrix} 1 & \delta_{\text{call}} \\ 0 & \nu_{\text{call}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -\delta \\ -\nu \end{pmatrix}$$

Where:

- X_1 is the Number of ENEL stocks needed to hedge Delta.
- X_2 is Number of 4-year ATM European call options needed to hedge Vega.
- Delta call and Vega call are defined based on a call computed with Black&Scholes formula.
- Delta and vega are the values computed before, the ones we want to achieve.

DV01 Hedging

DV01 (Dollar Value of 01) measures the sensitivity of the bond's price to changes in interest rates. To hedge DV01, use interest rate derivatives such as interest rate swaps (IRS) or bonds with a known DV01. To hedge the bond's DV01, take an opposite position in an interest rate swap or a bond with similar duration. Then, regularly rebalance the hedge to maintain a neutral DV01 as interest rates and market conditions change.

In our case, to hedge the EURIBOR bond using a IR SWAP we calculate the forward interest rate using:

$$S_0 = \frac{1 - B(0, T_{M+1})}{\delta \sum_{i=1}^{M+1} B(0, T_i)}$$

And the price of the swap will be:

$$NominalAmount \cdot S_0 \delta \sum_{i=1}^{M+1} B(0, T_i)$$

Results

The underlying-Delta via the ENEL stock (number of stocks): 28M

Vega via a 4y ATM-Fwd European Call on the underlying (number of Calls): 5257 (short)

Interest rate risk via a 4y IR swap vs Euribor 3m (Notional Amount): 12.29M

The structured bond's sensitivities can be effectively hedged using Delta, Vega, and DV01 hedging strategies. By creating offsetting positions in the underlying asset, options, and interest rate swaps, the bondholder can mitigate the risks associated with changes in the price of the underlying asset, volatility, and interest rates. Although the original code does not provide a hedging strategy, the framework discussed here outlines the essential steps and considerations for implementing an effective hedge, ensuring robust risk management and financial stability.

Moreover, practical implementation could require more careful consideration of the following:

- Transaction Costs: Costs associated with trading stocks, options, and swaps can impact the effectiveness of the hedge.
- Liquidity: Ensure that the chosen hedging instruments have sufficient liquidity to avoid large bid-ask spreads.
- Model Risk: The accuracy of the hedging strategy depends on the models used for calculating Delta, Vega, and DV01.
- Monitoring and Rebalancing: Continuous monitoring and periodic rebalancing are essential to maintain effective hedges, especially in volatile markets.

4. Comments on the results

From the results we realised that we should use a bigger number of simulation because the variance is relevant due to the randomness of the processes.

Moreover, by considering the notional amount, which is very high, we found that the greeks are relevant values and we see that are high. However, applying the right concept we got those values.

5. References

Some of the formulas and the procedures that you will find in the code are based on the following texts:

Monte Carlo methods in financial engineering -- Paul Glasserman

Musiela and Rutkowski - 2007 - Martingale methods in financial modelling

Hull - 2022 - Options, futures, and other derivatives