UNIVERSITY PARIS-DAUPHINE

Python Project - M2 203

Equity Structured Products Backtester

Anthony NADJARI-FINCKELSTEIN

Matthieu GREEN

November 2022

Contents

Part I
Introduction

After several internships in Structured Products in Sales, Trading and Structuring, we had the opportunity to see how these three functions work altogether, but we also noticed that there were sometimes lack of communications regarding certain products. In fact, although Structuring is responsible of pricing structured products for sales, it is also responsible of producing backtest to be sent to the client. Due to the huge number of pricing requests, sales have obtained access to a pricer so they can price various things, and let the more complex structures to Structuring. However, there is also a huge demand in backtests, that the sales are not responsible of. They generally do not have access to a backtester, although its functioning is as simple as a pricer. To ease the flow in a trading floor, we thus decided to launch a very user friendly backtester to backtest a wide range of structured products.

Part II
Brief introduction of Structured Products

First, there are two big categories of structured products: bullet and autocallables. Bullet products are products that end at maturity, unlike autocallable products that can end whenever. Before introducing autocallable products, it is necessary to present particular products that are used in those structures.
The functioning of structured product is the following: the client gives the bank a nominal expressed in percentage (if the client wants to invest 5 millions of euros, then 5,000,000 will be 100%. At the end of the product, the owner of the structured product will receive back the nominal, plus or minus a certain amount depending on the product and the performance of the underlying.
An equity structured product is composed of two legs. The first leg is called the short leg, because it is composed of options that the client shorts, and the long leg, composed of options that the owner buys. Usually, the short leg is used to be able to finance (partially) the long leg: in a simple case scenario, the owner sells a structured put option to be able to finance digits, which are assimilated to coupons. The aim of this leg is also to ensure a certain safety, as we mentioned before. This safety is often called "Guaranteed Capital" (denominated "KG").

1 Short leg: the way to finance coupons

The short leg aims at selling an option to the bank that bets on the downside of the stock: the lower the stock (or the forward) will be, the more "in the

money"[1]" the underlying will be, and thus, the more expensive will the option be. Finally, this will lead to a bigger coupon, financed by a product called "digital", on which we will come back later.
There are different kinds of products that we can find on the short leg. As a reference, the level of the strike of these options is expressed in a percentage of the spot, which is the current level of the underlying: if the spot is at 203, a strike at 100(%) means that the value of the strike is 203. Usually, short leg options that we find in structured products are striked at 100%, because it is the best equilibrium. Having a lower strike would increase the guaranteed capital but decrease the possible gain, and having a higher strike would have the exact opposite effect.

1.1 Put options: European and American

A put option is the most classic option to bet on the downside of an underlying. The owner pays the price of the option called "the premium", and benefits in return of the negative performance of the underlying. There are two types of put options. The European**?** put option can be exercised only at maturity, whereas American put options can be exercised at any moment between the strike date (i.e. the beginning) and the maturity date.[2] The payoff[3] of this option at any time t is:

$$Payoff = (K - S_t)^+$$

where x^+ is the positive part of x (i.e. $max(0, x)$), K represents the strike, and S_T the value of the underlying at time T, the maturity. Both strike level and value of underlying can be represented either in percentage, or in value, but they must be of the same type.
The payoff can be represented as follows:

1.2 European Put Down-And-In

The European put down-and-in is part of a family of options called "barrier options". It is basically a put option, to which we add another parameter called the "barrier". There are 4 types of barriers. Once the barrier is breached (at maturity for a European option and during the lifetime of the product for an American option), the product either activates or deactivates. For and up-and-out barrier for instance, the product is deactivated ("out") if the underlying goes above the barrier ("up"). For a "down-and-in" barrier, the product is activated

[1] The moneyness of a put option depends on the value of

$$(K - S_t)$$

at any time t. If it is positive, the option is "in-the-money", if it is negative the option is "out-of-the-money", and if it equals 0, it is "at-the-money".

[2] In practice, American Put options can be exercised "only" every day.

[3] The payoff of an option represents the gain/loss at maturity. It does not take into account the premium.

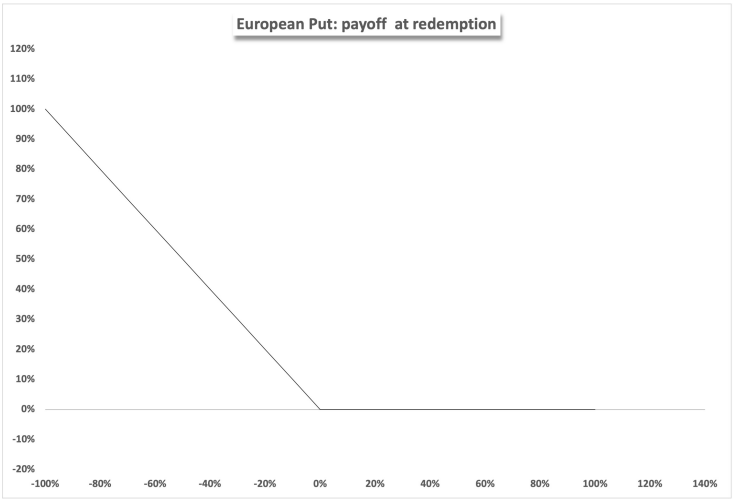


Figure 1: European Put Payoff at maturity

("in") if the underlying goes below ("down") the barrier. Thus, the European PDI (for "Put-Down-and-In") will activate the put option if the underlying is below the barrier. The payoff is given by the formula:

$$Payoff = \begin{cases} (K - S_T)^+ & \text{if} \quad S_T \leq B \\ 0 & \text{otherwise} \end{cases}$$

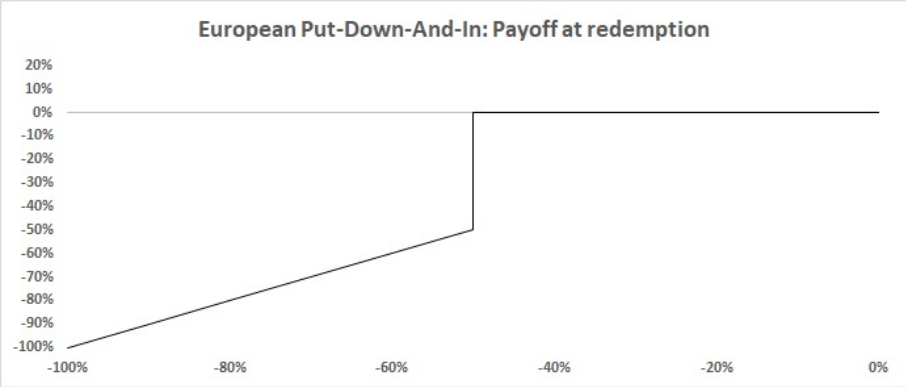


Figure 2: European Put Down-And-In Payoff

Compared to a simple put option, this product is less expensive : when the client is long PDI, he does not win anything as long as the barrier is not breached. However, this product is the most used because it allows the client to have its capital protected as long as the barrier is not breached.

1.3 Geared put: an alternative to a put option

Geared put is a common name for leveraged put. Before explaining the principle, let us first introduce the notion of "leverage". The objective of this product is to benefit from the performance of an At-The-Money put (we will denote that ATM Put), by financing only Out-Of-The-Money Puts (we will denote that OTM Put).
To illustrate that, let's take some concrete data: first, take an ITM Put, so a Put with a strike 100%. The maximum gain that the owner can have is called the cap: by default, the cap of a put is its strike, because the stock cannot go below 0 (theoretically):

$$\max_{S_T \in [0;+\infty[} (K - S_T)^+ = K$$

In comparison, the cap of a call is $+\infty$, because :

$$\max_{S_T \in [0;+\infty[} (S_T - K)^+ = +\infty$$

Then, take a put with a 85% gearing; the cap of the put is thus 85%. In order to get a 100% cap, we must then either buy one ATM put, or buy
100% / 85% = 1,25 OTM put. This number is called the leverage of the put. When the owner of a structured product shorts a geared put, he benefits from the cap of an ATM put, but also has his capital guaranteed until the stocks decreases

1.4 Put-spread: a way to drastically reduce risk

Another great alternative to both finance coupons and reduce risk exposure is the Put-spread. The principle of this product is to finance a put of Strike K_1 by short-selling another put of strike K_2, with $K_2 < K_1$. When an agent owns this strategy, it leads to the following result: the owner benefits from the negative performance of the underlying from K_1 to K_2, but his benefits are capped at $K_1 - K_2$ from K_2 to 0. However, using this strategy in a short leg allow the owner to have a maximum loss of $K_1 - K_2$. It is therefore one of the less risky strategies to use in a short leg, because it is the only one that guarantees a fixed capital at maturity compared to the other strategies we have seen before. However, it is not the one that will allow the user to finance the more coupons: since it is less risky, it is worth less. We can also see that, compared to a put, if we are short a put-spread, we buy a put with strike K_2, which reduces the amount we get.

2 Long Leg: benefiting from coupon

As we said earlier, the aim of the short leg is to finance coupons. The name of this product is a "digit", and is part of the family of barrier options like the Put Down-And-In. This time, the option is an Up-And-In option. If the underlying

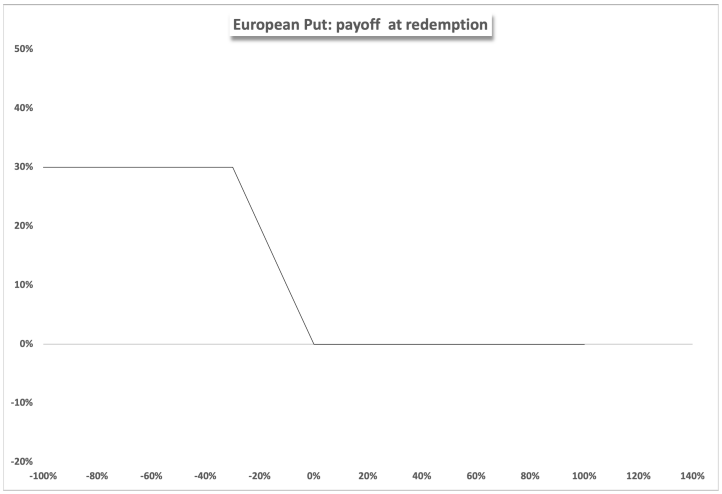


Figure 3: European Put Spread Payoff

goes above this barrier, it redeems a fixed coupon. Thus, the payoff can be expressed by this formula:

$$Payoff = \begin{cases} c & \text{if} \quad S_T \geq B \\ 0 & \text{otherwise} \end{cases}$$

It can also be written as:

$$Payoff = c * \mathbb{1}_{\{S_T \geq B\}}$$

3 Bullet Products: a first step to autocallable structures

Bullet products are structured products that do not have an autocallable feature. The observations can be at anytime from the strike date to the maturity date, but the payoff will always occur at maturity. To illustrate that, let us review a few products.

3.1 Reverse convertible: a simple structured product

The reverse convertible is probably the most classic and the simplest bullet structured product. At a given observation frequency, the client receives a fixed coupon regardless of the performance of the underlying. However, his invested capital is still exposed to the performance of the underlying. Indeed, the capital amount that is redeemed to the investor at maturity will be assessed by whether

or not the PDI barrier was activated. In other words, the PDI aims at financing the coupons. The payoff at any time before maturity is therefore c, and the payoff at maturity can be written as below:

$$Payoff = \begin{cases} c & \text{if} \quad S_T \geq B \\ c + K - S_T & \text{otherwise} \end{cases}$$

An additional barrier can be added for the coupon: it is not guaranteed as before, and will be redeemed if the underlying goes above the coupon barrier (which is then higher than the PDI barrier).
The payoff of a Barrier Reverse Convertible at maturity is then:

$$Payoff = \begin{cases} c & \text{if} \quad S_T \geq B_{coupon} \\ K - S_T & \text{if} \quad S_T \leq B_{PDI} \end{cases}$$

3.2 Twin-Win with European PDI: betting on volatility

The Twin-Win is the best product when we want to bet on the volatility of an underlying: when the owner thinks that the underlying will be moving a lot on the upside and/or on the downside, he could be inclined to buy either a put or a call, but buying both would be quite expensive. Therefore, the owner has the possibility to buy both by shorting a PDI. We say that we benefit from the "absolute performance" of the underlying. Therefore, the call will be uncapped, but the put will be capped from the PDI strike: it is therefore a Down-And-Out put, because it will deactivate if the PDI activates. The call will also deactivate if the PDI barrier is breached at or before maturity. Another variation would be to buy an Up-And-Out call: this would be less expensive than an uncapped call, and in case of medium-high volatility, it would also provide benefits if the underlying increases.

$$Payoff = \begin{cases} |S_T - K| & \text{if} \quad S_T \geq B \\ K - S_T & \text{otherwise} \end{cases}$$

4 Autocallable Structures

Now that we detailed the principles of an equity structured product, it is time to introduce autocalls, a nickname for "autocallable structures". These are structured products composed of two legs as we saw before, but with an additional feature: the product can end before its maturity date.
Additional features go along with additional barriers: the product automatically ends when a barrier is breached. This barrier is called "autocall barrier", and is also an important parameter to play with to perfect the product. Once the product is recalled, the owner usually receives a coupon but this also depends

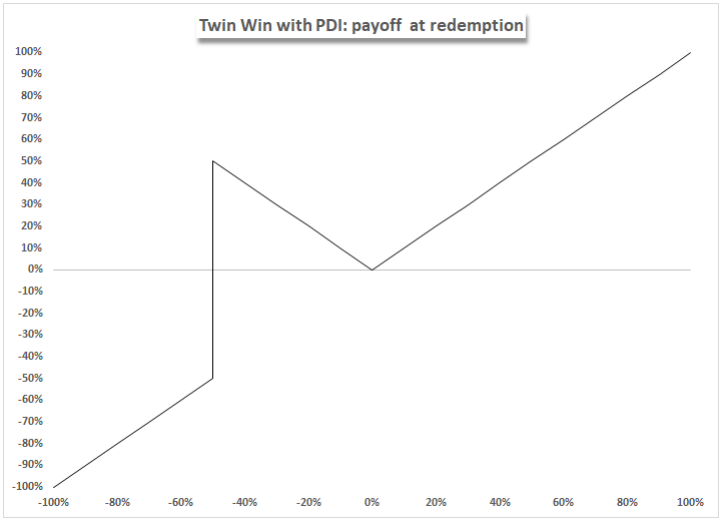


Figure 4: Twin-Win with PDI Payoff

on the product. For instance, the product can have a coupon barrier different from the autocall barrier. This feature is also very useful, because as soon as it is recalled, the PDI deactivates and the owner locks the coupon. However, it is very important to understand that this autocall features makes the product unable to be expressed with other vanilla / barrier options: since we do not know in advance when the product is going to be recalled, it is impossible to say that the product is made of digits. There is of course the notion of digitals, because the client is redeemed a coupon when recalled (or when above a barrier), but we do not know the maturity of these. This complexity makes the autocall un-pricable with Black-Scholes model: we cannot deduce, from the payoff, a perfect written formula. Hence, other methods have to be applied to be able to price these options. We will come back to these methods later in this thesis.
When an autocall is ordered, parameters are taken into account depending on the product.

4.1 Calibration of an Autocall: parameters

Firstly, one or multiple underlyings shall be chosen. In this case, the type of basket has to be determined too. There are three types of basket in the market:

- the worst-of basket: at each observation date, the underlying with the worst performance determines the level of the basket.
- the best-of basket: at each observation date, the underlying with the best performance determines the level of the basket.
- the equally-weighted basket: at each observation date, the level of the basket is determined by taking the weighted average of the performances of each underlying.

Then, the maturity has to be chosen: even if the product can be ended before maturity thanks to the autocallable feature, if no autocall event has happened, the product will end like a bullet product depending on its performance and structure.
Finally, the client should choose the frequency of the observations. It is usually daily, quarterly, semi-annually, annually or bi-annually.

4.2 Athena Autocall

Let us introduce this structure by giving a classic example: the Athena autocall. This structure is the most basic autocall that we find on the market, but also the most traded among Equity Structured Products. The short leg of the autocall is composed of a PDI. When the underlying hits the autocall barrier, the products ends and the owner receives the nominal and a coupon. The coupon is said to be "snowball", because it increases as time goes.
For instance, take a classic Athena which has a maturity of n years, and yearly observations. The coupon that the autocall delivers can be written as:

$$\forall t \in [\![1\ ;\ n]\!], c_t = t * c_1 = c_{t-1} + c_1 \text{ (assuming } c_0 = 0)$$

The payoff cannot be expressed in one formula, but it is possible to express the payoff depending on the scenario that happens. Taking an athena with maturity n, observed yearly with an annual coupon of c gives us:

$$\forall t \in [\![1\ ;\ n-1]\!],\ Payoff_t = (100\% + c_t) * \mathbb{1}_{\{S_t \geq B_{autocall}\}}$$

Then, at maturity, the payoff ends up being:

$$Payoff = \begin{cases} 100\% + n * c & \text{if} \quad S_T \geq B_{autocall} \\ 100\% - (K - S_T) & \text{if} \quad S_T \leq B_{PDI} \end{cases}$$

(Note that $K = 100\%$ for almost every PDI)

5 Presentation of the file codes

5.1 Global interface file

The file "main.py" contains all the Dash Plotly general interface. This allows us to create a userfriendly programme. This particular file is used to generate the navigation sidebar and top bar. To add the content, we create a file for each corresponding tab, and in our case, this corresponds to the next file.

5.2 Tickers file

The Json file is a dictionary containing the keys (what is displayed in the Dash Dropdown to select the tickers) and their values (that we use to scrap the data on the website).

5.3 Webscrapping of the data

The inputs necessary for our back tester are obviously the historical daily prices of the basket's stocks for the selected period. To make our user interface more accessible, we decide to limit the user's choice to a list of stocks that are usually found as underlyings for structured products. However, the web scrapper is not limited to above mentioned stocks, as it must be able to scrap prices (i) for any stock or index, (ii) for any timeframe, and (iii) from any exchange.

The first requirement wasn't an issue as there exists hundreds of websites that publish stock and index prices. The second one was trickier, and ultimately, the reason why we opted for a combination of Selenium and BeautifulSoup to extract the prices. Indeed, when dealing with large datasets, websites usually present the data in the form of multiple-pages tables. In some cases, each page has its own unique URL, but in others, it triggers AJAX calls. This was the case with Boursorama, for which we would simulate clicks with Selenium to navigate through the different pages. Although this process worked fine, Boursorama would only have 3 years of price history. This eventually led us to choose ADVFN, as the only website with enough historical data, tracing back more than 10 years. Although ADVFN does have unique URLs for each table page, the website returns "403 Forbidden" responses to our GET requests. We tried setting headers with no luck. Once again, we had to choose the less efficient Selenium over Beautiful Soup. Although the latter is still used to parse the HTML data. Finally, it is important to note that ADVFN specifies the exchange on which the stock is listed in the URL. Assuming that the user might not know where the stocks are listed, we decided to use the website's suggestion box to fetch the exchange.

Inputs

- Ticker

- Start Date
- End Date

Outputs

- Table of prices for the selected period
- CSV file
  - Number of Rows = number of observations (i.e., days)
  - Number of Columns = 4 (Close / Open / High / Low)

Data Collected (daily basis)

- Close
- Open
- High
- Low

Code Breakdown

1. Extract stored CSV data if it exists
   - If all is available in stored CSV we go straight to the next ticker
   - If the stored data is incomplete, we will only scrap missing prices
2. Launch the web driver
3. Find corresponding exchange (NASDAQ, NYSE, EURONEXT, etc.)
   - Enter ticker in search box
   - Extract the exchange from the first suggestion
4. Enter the start and end date
5. Loading Historical Prices
6. Loop through Table pages
7. Saving Data in an individual CSV file (ex: AAPL.csv)

Our main concern regarding the scrapping process is the efficiency. Opening a web driver and looping through pages (ex: 43 for 10 years of prices) is time consuming and requires significant processing power. We aim to reduce as much as possible such time delay by loading prices from the previously stored CSV files. Every time the user needs prices outside of the stored data, the web scrapper will complete the stored data with freshly scrapped prices. For instance, let's imagine we have stored the prices for AAPL from March 2014

to May 2020 from a previous run of the code. If the user now needs the prices for the period from June 2015 to July 2019, it will return the requested prices without launching the web driver, hence saving a significant amount of time. On the other hand, if the user needs prices starting in January 2011 and ending in November 2022, it will (i) scrap prices from January 2011 to March 2014 and (ii) scrap prices from July 2019 to November 2022. It will therefore complete existing stored prices with the new ones, save them and return the table to the user.

5.3.1 get_price_table()

Inputs

- List of tickers (ex: ['AAPL', ..., 'BNP'])
- Start Date
- End Date

Outputs

- Concatenated table of close prices for all tickers
- Individual CSV files
- CSV file for concatenated table
  - Data = Close Prices
  - Number of Rows = number of common observations for the list of tickers
    - Accounting for different bank holidays depending on the exchange
  - Number of Columns = number of tickers

Code Breakdown

- Loop through tickers
- Extract individual close prices
- Saving concatenated table with all the tickers (all_data.csv)

5.4 Backtester interface

We will start by describing the parameters to input, before giving more details about them. To begin with, the user shall input the range of dates to compute the backtest. All the other fields should be in percentage, including the barriers. For non-autocallability, the user can input a huge number such as 9999. It is highly recommended to put a very high number, because in some cases, a 999%

increase can happen! The user can just click on submit and the backtester will produce a graph that is exportable / zoomable.

The next file is the most important, because it explains all the important details that make the backtester work.

5.5 Backtester algorithm

The principle of the backtest is simple. We compute the final payoff of a product from a predefined start date, for a big number of start dates. This gives a good approximation and idea of the value of the product, without having to compute its price with models. It also helps forecasting whether or not the launch of the product today would be a success. // However, there are some things to specify regarding autocallable products. As we know, these products have a fixed maturity, but can be autocalled before. A 10-year product which started in 2015 can be autocalled, or still be alive. In the latter case, should we include them? The answer is yes. We thus distinguish 3 cases:

- the maturity of the product ends before the backtest end date (that the user inputs): there is no problem to compute the result.
- the maturity of the product ends after the backtest end date (that the user inputs): there are 2 cases:
  - the product has already autocalled: we keep the result and save it
  - the product is still alive: we do not take it into account and it is considered as "None" in the script

5.6 Download the explanatory file

You perhaps already know how to do this since you are reading the file... The second tab of the backtester redirects the user to a GitHub page where the download of this LaTeX file is automatic.

5.7 Displaying the data

The last tab allows the user to display the data that we previously webscrapped. The user shall input the range date of the data and the tickers. The programme will automatically source the data from the individual files containing the Open, Close, Lowest and Highest prices of each day of the range date.

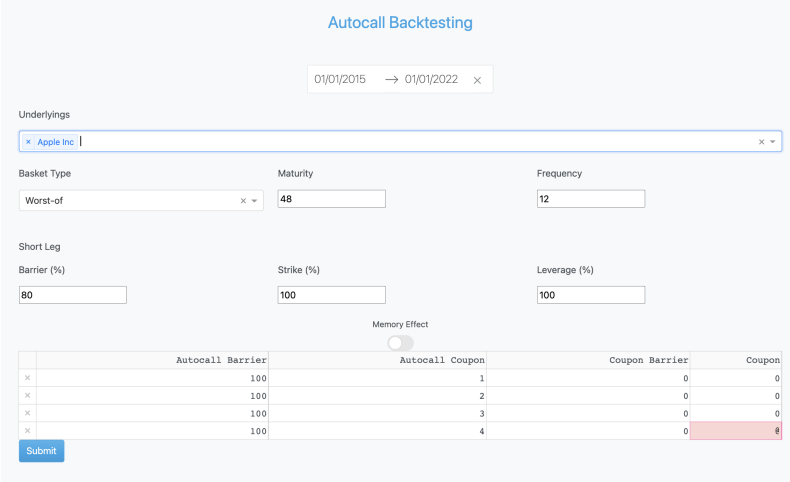
5.8 Quick example

We will test the following, as this can be used as a template.

- We pick Apple stock
- We backtest from 2013 to 2022

- Since it is only one stock, it does not matter if we choose Worst-of or Best-of
- **Make sure that $Maturity/Frequency$ is an integer and corresponds to the number of barriers!** We will backtest a 4-year product of frequency 1 year
- We choose a PDI 80% Barrier (100% strike and leverage, most basic case)
- We input 100% Barriers and an incremental coupon of 1%
- The result (in percentage) corresponds to the amount that the client receives after an investment of 100% (60% thus corresponss to a loss).

Here is the backtester configuration:



Here are the results!