Swaps and Bootstrapping - Practical Lesson 5

Matteo Sani
matteosan1@gmail.com

October 22, 2019

1 Swaps and Bootstrapping

1.1 Recap

- basic Python (mostly not related directly to finance)
- how to implement a discount factor interpolation function
- wrapping up functionality in classes in order to work with multiple data sets more easily
- libor forward rate calculator

1.2 Today's lesson

We're going to look at:

- modules, and start building up our library of finance-related functionality;
- implementing an Overnight Index Swap class for calculating the NPV of an OIS;
- bootstrapping technique to derive discount curve.

1.3 Modules (again)

An interactive session (e.g notebook or interactive shell) is great for quick testing and exploratory use, but once you have some code (i.e. functions or classes) which you'd like to reuse often, rather than copy/pasting it every time you need it, you can save it in a .py file and use it from your session (aka you can create your own library). These work just like the modules we have been importing up to now, except they're written by us...

Today we're going to start writing a module called **finmarkets**, and over the course of the remaining lessons we'll add functionality related to the theory lessons **(consider that you will be asked to use as much as possible this module for your final project !).**

According to your preferred way of working there are different instructions to write a module:

- take a look at this video for an example of how using your own module is done for *Jupyter notebook*;
- if you prefer to work with *repl.it* instead please look at this skeleton of project which has a finmarkets module example.

So to start our financial module let's create a new file called `finmarkets.py` and copy the `DiscountCurve` class you have written last time into it. As for any other module we will be able to import it and use its classes and functions. Let's try with an example:

```
In [1]: from datetime import date
        from finmarkets import DiscountCurve

        curve = DiscountCurve(date(2020, 1, 1),
                              [date(2020, 1, 1),
                               date(2021, 6, 1),
                               date(2022, 1, 1)],
                              [1.0, 0.98, 0.82])
        curve.df(date(2020, 7, 1))

Out[1]: 0.9929132520645648
```

2 Overnight Index Swap

Overnight Index Swaps (OIS) are products which pay a floating coupon, determined by overnight rate fixings over the reference periods, against a fixed coupon. Interest rate swaps are usually used to mitigate the risks of fluctuations of varying interest rates, or to benefit from lower interest rates. We will always look at these products from the point of view of the **receiver of the floating leg**. By definition an OIS is defined by:

- a notional amount N
- a start date d_0
- a sequence of payment dates $d_1, ..., d_n$
- a fixed rate K

(n.b. for simplicity we're assuming that the fixed and floating legs have the same notional and payment dates, although this is not necessarily always the case in practice).

At each payment date, the floating leg pays a cash flow determined as follows:

$$f_{\text{float, }i} = N\left\{\prod_{d=d_{i-1}}^{d=d_i-1}\left(1 + r_{o/n}(d)\cdot\frac{1}{360}\right) - 1\right\}$$

(This formula is valid for an EONIA swap, i.e. for OIS swaps in EUR, other currencies might have different conventions. The $\frac{1}{360}$ fraction appears because EONIA rates are quoted using the ACT/360 daycount convention and here we're making the simplifying assumption of ignoring weekends and holidays, so we assume that each overnight rate is valid for only one day.)

The sum of the discounted expected values of these cash flows is

$$\text{NPV}_{\text{float}} = \sum_{i=1}^{n} D(d_i)\mathbb{E}[f_{\text{float, }i}]$$

where $D(d)$ is the discount factor with expiry d. On the other hand, by definition (remember practical lesson 4 with forward rates), we also have the following relationship

$$\mathbb{E}[f_{\text{float, }i}] = N\cdot\left(\frac{D_{ois}(d_{i-1})}{D_{ois}(d_i)} - 1\right)$$

hence

$$\mathrm{NPV}_{\mathrm{float}} = N \cdot \sum_{i=1}^{n} D(d_i)\Big(\frac{D_{ois}(d_{i-1})}{D_{ois}(d_i)} - 1\Big)$$

where $D_{ois}(d)$ is the discount factor implied by OIS prices (we will see it better later).

The correct curve to use for discounting the flows of a collateralized contract, like OIS, is the one associated with the collateral. Since OIS contracts are collateralized with cash, and cash accrues daily interes at the overnight rate, the OIS curve is itself the correct curve with which to discount the flows of an OIS contract !

In summary, $D = D_{ois}$ so the NPV simplifies to

$$\begin{aligned}\mathrm{NPV}_{\mathrm{float}} &= N \cdot \sum_{i=1}^{n} [D(d_{i-1}) - D(d_i)] = \\ &= N \cdot [(D(d_0) - D(d_1)) + (D(d_1) - D(d_2)) + ... + (D(d_{n-1}) - D(d_n))] \\ &= N \cdot [D(d_0) - D(d_n)]\end{aligned} \tag{1}$$

Each cash flow of the fixed leg is equal to

$$f_{\mathrm{fix},\,i} = N \cdot K \cdot \frac{d_i - d_{i-1}}{360}$$

so the NPV of the fixed leg is

$$\mathrm{NPV}_{\mathrm{fix}} = N \cdot K \cdot \sum_{i=1}^{n} D(d_i)\frac{d_i - d_{i-1}}{360}$$

Ultimately the aim will be to take a series of OIS quotations, and determine the discount factors implied by their prices. To do this we'll build a pricing `class`, which takes discount curve as the input and produces the net present value (NPV) of the OIS as the output. Then we'll put this function inside a numerical optimizer to invert the process to determine the implied discount factors from their prices (market quotes).

```
In [2]: class OvernightIndexSwap:

            # this method is called to build the instance,
            # we take some data arguments and save them as
            # attributes of self
            # n.b.: payment_dates should be a list of dates,
            # including the start date as the first element
            def __init__(self, notional, payment_dates, fixed_rate):
                self.notional = notional
                self.payment_dates = payment_dates
                self.fixed_rate = fixed_rate

            # this method takes a discount curve and calculates
            # the NPV of the floating leg using that curve
            def npv_floating_leg(self, discount_curve):
                # self.payment_date s[0] is the start date of the swap
                # self.payment_date s[-1] is the last payment date of the swap
            return self.notional * (discount_curve.df(self.payment_dates[0]) -
                                    discount_curve.df(self.payment_dates[-1]))

        # this method takes a discount curve and calculates the NPV
        # of the fixed leg using that curve
        def npv_fixed_leg(self, discount_curve):
            npv = 0
            # we loop from i=1 up to but not including the length of the date list
            for i in range(1, len(self.payment_dates)):
                # we can do i-1, because the loop starts with i=1
                start_date = self.payment_dates[i-1]
                end_date = self.payment_dates[i]
                tau = (end_date - start_date).days / 360
                df = discount_curve.df(end_date)
                npv = npv + df * tau
            return self.notional * self.fixed_rate * npv

        # this method calculates the NPV of the OIS swap
        # n.b.: inside this method we call the other two
        # methods of the class on the same instance 'self',
        # using self.npv_XXX_leg(...), and we pass the
        # discount_curve we received as an argument
        def npv(self, discount_curve):
            float_npv = self.npv_floating_leg(discount_curve)
            fixed_npv = self.npv_fixed_leg(discount_curve)
            return float_npv - fixed_npv
```

```
In [3]: from datetime import date

        ois = OvernightIndexSwap(
            # the notional, one million
            1e6,
            # the list of product dates,
            # i.e. the start date then the payment dates
            [date(2020, 1, 1),
             date(2020, 4, 1),
             date(2020, 7, 1),
             date(2020, 10, 1),
             date(2021, 1, 1)],
            # the fixed rate, 2.5%
            0.025
        )
```

We can now use the curve we have prepared at the beginning of the lesson and that we stored in a variable called curve to check if the class is working by evaluating the NPV of the OIS.

```
In [4]: ois.npv(curve)

Out[4]: -10990.364227052869
```

2.1 Bootstrapping

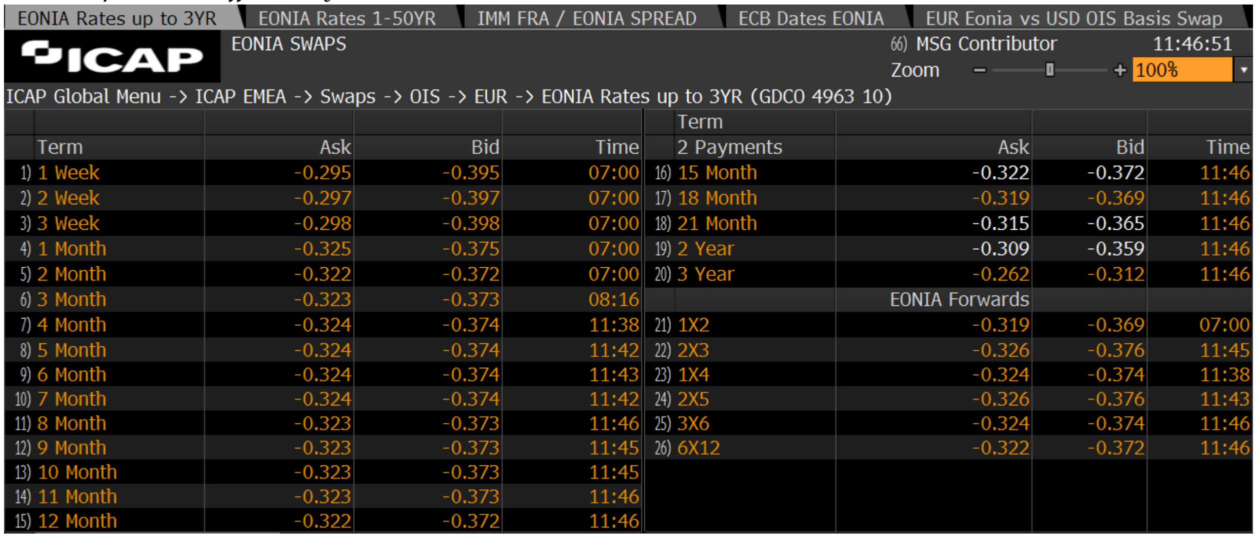
Now we're going to look at how extract a discount curve from OIS market data, via a process called *bootstrapping*. This is the ABC of financial mathematics, since you almost always need a discount curve to price any contract, especially if you're interested in its NPV. We're going to concentrate on EONIA swaps in order to build an EUR discount curve.

2.1.1 Getting the data

The first problem is actually getting the data (swap market quotes) from somewhere, and this is not actually as simple as it sounds.

The issue is that the EONIA swap market is Over The Counter (OTC) and it's not straightforward to access it. Unlike (some) listed futures, where anyone with a retail brokerage account can view and apply realtime prices, to trade in the EONIA swap market you have to be a financial institution or at least a large company and have an agreement with a broker which operates in the market. One of the main brokers in the OIS market is ICAP.

Though there exist some electronic platform in which market participants post bids and offers and other participants can apply them, in practice a lot of trading is still done over "voice", i.e. by phone or more commonly over chat. For convenience, however, Bloomberg provides a service which displays indicative realtime rates as provided by a selection of relevant brokers. (*n.b. interest rate swap quotes vary from standard price quotes of commonly traded instruments, they can appear puzzling because the quotes are effectively interest rates*)



ICAP Global Menu -> ICAP EMEA -> Swaps -> OIS -> EUR -> EONIA Rates up to 3YR (GDCO 4963 10)

Term	Ask	Bid	Time	Term 2 Payments	Ask	Bid	Time
1) 1 Week	-0.295	-0.395	07:00	16) 15 Month	-0.322	-0.372	11:46
2) 2 Week	-0.297	-0.397	07:00	17) 18 Month	-0.319	-0.369	11:46
3) 3 Week	-0.298	-0.398	07:00	18) 21 Month	-0.315	-0.365	11:46
4) 1 Month	-0.325	-0.375	07:00	19) 2 Year	-0.309	-0.359	11:46
5) 2 Month	-0.322	-0.372	07:00	20) 3 Year	-0.262	-0.312	11:46
6) 3 Month	-0.323	-0.373	08:16		EONIA Forwards		
7) 4 Month	-0.324	-0.374	11:38	21) 1X2	-0.319	-0.369	07:00
8) 5 Month	-0.324	-0.374	11:42	22) 2X3	-0.326	-0.376	11:45
9) 6 Month	-0.324	-0.374	11:43	23) 1X4	-0.324	-0.374	11:38
10) 7 Month	-0.324	-0.374	11:42	24) 2X5	-0.326	-0.376	11:43
11) 8 Month	-0.323	-0.373	11:46	25) 3X6	-0.324	-0.374	11:46
12) 9 Month	-0.323	-0.373	11:45	26) 6X12	-0.322	-0.372	11:46
13) 10 Month	-0.323	-0.373	11:45				
14) 11 Month	-0.323	-0.373	11:46				
15) 12 Month	-0.322	-0.372	11:46				

As part of our Quants duties we have set up an Excel spreadsheet which acquires this data from Bloomberg in realtime. From this spreadsheet, it's easy to export the data into a Python file - I have done this and saved the data in a file (`ois_data.py`).

We now use this dataset to derive a discount curve, let's check how it looks like:

```
In [5]: import ois_data
        print (type(ois_data.quotes))
```

```
<class 'list'>
```

```
In [6]: ois_data.quotes[0]
```

```
Out[6]: {'months': 1, 'rate': -0.35}

In [7]: ois_data.quotes[-1]

Out[7]: {'months': 720, 'rate': 0.997}

In [8]: ois_data.observation_date

Out[8]: datetime.date(2019, 10, 23)
```

2.1.2 Building OIS instances

Let's say we want to build a 15 months swap instance, we have to use data contained in `ois_data` file:

```
In [9]: # first check the 15 months rate
        ois_data.quotes[12]

Out[9]: {'months': 15, 'rate': -0.35}
```

We can build the swap instance like this:

```
ois = OvernightIndexSwap(1e6,
                          [date(2019, 10, 23),
                           date(2020, 10, 23),
                           date(2020, 1, 23)],
                          ois_data.quotes[12]['rate']*0.01
                         )
# print the last payment date (15 months after obs date)
ois.payment_dates[-1]
```

Clearly to use the `npv` method to calculate the OIS' NPV we need a discount curve with which to evaluate it and here comes to hand the bootstrapping technique !

2.1.3 Exercise 5.1

Today and in the next lessons we're going to build lots of `OvernightIndexSwap` objects, one for each market quote we have (market quotes, as we have seen, consist of fixed strikes for 1M, 2M, 3M, ..., 12M, 15M, 18M, 2Y, 3Y, ..., 30Y and 40Y swaps).

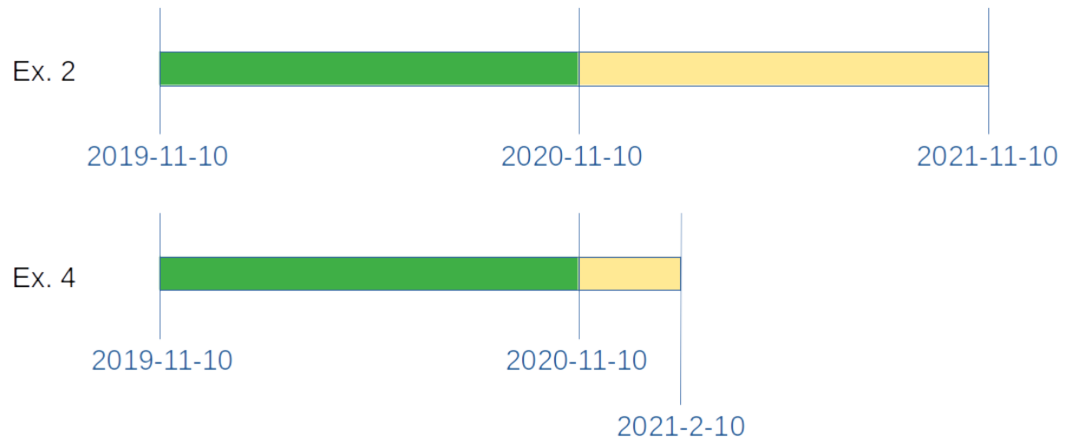
It would be very boring to write a long list of payment dates for each one of these, plus they'd need to be updated every day. Write a function which given a start date and the number of months, returns a list of dates of **annual** frequency starting from the start date and ending after the specified number of months.

For example

- 2019-11-10 start date 12 months → 2019-11-10, 2020-11-10
- 2019-11-10 start date 24 months → 2019-11-10, 2020-11-10, 2021-11-10

Note that if the number of months is not a multiple of 12, the last period should simply be shorter than 12 months. For example

- 2019-11-10 start date 9 months → 2019-11-10, 2020-08-10
- 2019-11-10 start date 15 months → 2019-11-10, 2020-11-10, 2021-02-10



Here's some skeleton code to help you get started:

```
from dateutil import relativedelta

def generate_swap_dates(start_date, n_months):
    dates = []
    # your code here which adds all the relevant dates to the dates list
    return dates

# some tests to check if the function is working correctly
from datetime import date

assert generate_swap_dates(date(2019, 11, 10), 12) == [date(2019, 11, 10),
                                                       date(2020, 11, 10)]
assert generate_swap_dates(date(2019, 11, 10), 24) == [date(2019, 11, 10),
                                                       date(2020, 11, 10),
                                                       date(2021, 11, 10)]

assert generate_swap_dates(date(2019, 11, 10), 9) == [date(2019, 11, 10),
                                                      date(2020, 8, 10)]
assert generate_swap_dates(date(2019, 11, 10), 15) == [date(2019, 11, 10),
                                                       date(2020, 11, 10),
                                                       date(2021, 2, 10)]
```

2.1.4 The Bootstrapping Technique

In the next we are going to somehow reverse what we have previously done (i.e. we determined the OIS's NPV given a certain discount curve).

The general idea here is to get the discount curve such that it prices correctly each OIS, or at least as well as possible, by minimizing the sum of the square NPVs:

$$\min_{curve}\left\{\sum_{i=1}^{n}\text{NPV}(\text{ois}_i,\text{curve})^2\right\}$$

A discount curve is characterized by pillar dates and the corresponding discount factors. The description of the problem we have given above does not, in theory, specifies any constraint on the pillar dates of the discount curve. However, the pillar dates determine the number of unknown variables (i.e. the dimensionality n of the optimization problem). A curve with n pillar dates has n discount factors (note that the first discount factor with value date equal to the today date, is constrained to 1). **In practice, therefore, it makes sense to choose the pillar dates in such a way that there are exactly the right number of degrees of freedom in the optimization to match data.** So the natural choice is to choose the pillar dates of the discount curve equal to the set of expiry dates of the swaps.

The reason for this is that each market quote will determine exactly one *free* discount factor which is not already determined by the other market quotes - this can be seen by considering the mathematical expression for calculating the fixed leg of the OIS swaps ($f_{\text{fix},\ i} = N\cdot K\cdot D(d_i)\cdot\frac{d_i-d_{i-1}}{360}$), and the way that the payment date schedules are constructed. Therefore, once we've fixed $\vec{d}$ to be a vector of pillar dates equal to the expiry dates of the OIS swaps, and we use the notation $\vec{x}$ to represent the vector of pillar discount factors, then the problem becomes:

$$\min_{\vec{x}}\left\{\sum_{i=1}^{n}\text{NPV}(\text{ois}_i,\text{curve}(\tilde{\text{d}},\tilde{\text{x}}))^2\right\}$$

In practice this is an optmization problem (**to find the minimum of the above expression as a function of $\vec{x}$**) so we can just use one of the available numerical optimization routines.

So let's start by defining a set of OIS objects to cover all the maturities defined by the market data we have collected in the `ois_data.py` file.

```
In [10]: from finmarkets import DiscountCurve, generate_swap_dates
         import ois_data_eric as ois_data

         pillar_dates = [ois_data.observation_date]

         swaps = [] # container of the OIS objects

         for quote in ois_data.quotes:
             swap = OvernightIndexSwap(
                 # notional - doesn't really matter what we put here
                 1e6,

                 # payment dates
                 generate_swap_dates(
                     ois_data.observation_date,
                quote['months']
            ),

            # the fixed rate (in the file is expressed in percent)
            0.01 * quote['rate']
        )
        swaps.append(swap)
        pillar_dates.append(swap.payment_dates[-1])

    pillar_dates = sorted(pillar_dates)
    n_df_vector = len(pillar_dates)
```

```
In [11]: type(pillar_dates), len(pillar_dates), pillar_dates[0], pillar_dates[-1]
```

```
Out[11]: (list, 34, datetime.date(2016, 11, 23), datetime.date(2076, 11, 23))
```

How Does the Minimization Algorithm Work ?

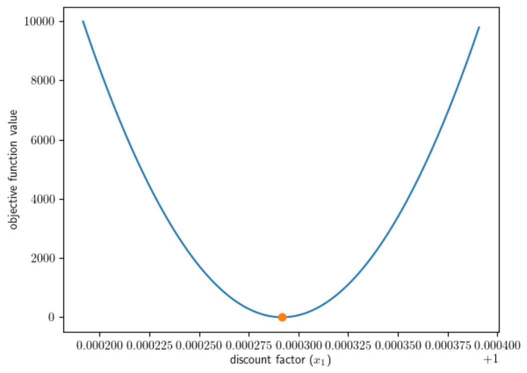
- Define an *objective function* i.e. the function that is actually minimized to reach our goal. In our case we want to find the discount curve which minimize the sum of the squared NPVs (swap quotes are considered their fair-values);
- set the intial value of the unknown parameters and their range of variability. We will set all the discount factors to 1 with a range of $[0.01, 100]$ (of course the first element of the list, today's discount factor will be set constant to 1);
- the *minimizer* will compute the objective function value;
- then will move the parameter values in such a way to find a smaller value of the objective function (e.g. *following* the derivative w.r.t. each parameter);
- the last two steps will be repeated until further variations of the x values won't change significantly the objective function (i.e. we have found a minimum of the function so the minimization process is completed !).

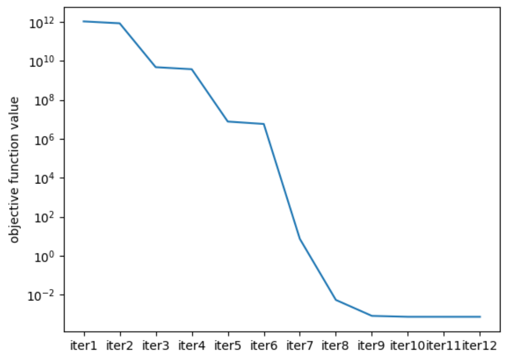
```
In [12]: def objective_function(x):

             curve = DiscountCurve(
                 ois_data.observation_date,
                 pillar_dates,
                 x
             )

             sum_sq = 0.0

             for swap in swaps:
                 sum_sq += swap.npv(curve) ** 2

             return sum_sq
```

To optimize our $\vec{x}$ we can use the `minimize` algorithm defined in `scipy.optimize`.

```
In [13]: from scipy.optimize import minimize

         # initialize to 1 the x vector (random choice)
         x0 = [1.0 for i in range(n_df_vector)]

         # set wide constraints on the discount factors
         # in the minimization problem the value of each x_i
         # will be bound between these limits
         bounds = [(0.01, 100.0) for i in range(n_df_vector)]

         # in addition we have an additional constraint:
         # we want the first pillar to be 1 (fixed)
         # (because it has pillar date = today)
         bounds[0] = (1.0, 1.0)

         # finally we run the minimization
         result = minimize(objective_function, x0, bounds=bounds)
```

Let's look at some diagnostic plots to check if the minimization was successful:





On the left the objective function value as a function of discount factor (x_1), on the right the value of objective function at each iteration.

```
In [14]: # print the diagnostic of the minimization problem
         result

Out[14]:       fun: 0.0007370890117814888
         hess_inv: <34x34 LbfgsInvHessProduct with dtype=float64>
              jac: array([ 6.40061365e+05, -4.14161280e+01, -1.93984741e+01,  5.37030079e+00,
                3.02530223e+01,  5.95243457e+01,  9.05547884e+01,  1.24363295e+02,
                1.59125629e+02,  1.97071574e+02,  2.36973096e+02,  2.76028231e+02,
               -9.72901535e+02, -3.95831915e+02, -3.67814737e+02, -3.29982597e+02,
               -3.11447882e+02,  1.31701062e+02,  5.96919251e+02,  9.85290168e+02,
          1.18797301e+03,  1.09656582e+03,  7.03858626e+02,  7.86777868e+01,
         -6.55082634e+02, -1.36397212e+03, -1.89121870e+02,  1.93487828e+03,
          5.40226051e+02, -1.65947564e+02, -5.36852034e+02, -2.47750882e+03,
         -1.84414691e+02,  1.90090790e+03])
  message: b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 875
      nit: 12
   status: 0
  success: True
        x: array([1.        , 1.00029175, 1.00058831, 1.00089012, 1.00116802,
         1.00147021, 1.00176786, 1.00207128, 1.00236508, 1.00266885,
         1.00297281, 1.0032578 , 1.00356124, 1.00445968, 1.00529983,
         1.00614269, 1.00693061, 1.00906201, 1.0093198 , 1.00710112,
         1.0018986 , 0.99379504, 0.9833297 , 0.97101001, 0.95723164,
         0.9426886 , 0.92772535, 0.88314869, 0.8178113 , 0.76554845,
         0.71988664, 0.64350636, 0.59281978, 0.54547324])
```

```
In [15]: # objective function value with starting point parameters
         objective_function(x0)
```

```
Out[15]: 1055841619695.9585
```

```
In [16]: # objective function value with final values
         objective_function(result.x)
```

```
Out[16]: 0.0007370890117814888
```

```
In [17]: # define the discount curve object using the
         # resulting discount factors (result.x)
         curve = DiscountCurve(ois_data.observation_date, pillar_dates, result.x)

         from datetime import date
         curve.df(date(2059, 11, 23))
```

```
Out[17]: 0.6278698804291626
```

```
In [18]: # 50 years rate
         import math
         -math.log(curve.df(date(2059, 11, 23))) / 50
```

```
Out[18]: 0.009308446615020075
```

```
In [19]: list(result.x)
```

```
Out[19]: [1.0,
          1.0002917467402102,
          1.000588313127234,
          1.0008901199538132,
          1.0011680243827574,
          1.0014702089411056,
          1.0017678648937525,
          1.0020712764009663,
          1.0023650754871605,
          1.0026688489207223,
          1.0029728065322203,
          1.0032577961650246,
          1.003561243417754,
          1.004459676763001,
          1.0052998330195508,
          1.0061426892577996,
          1.006930607427503,
          1.009062012124004,
          1.0093198010277291,
          1.0071011202466504,
          1.00189860229147,
          0.9937950392360159,
          0.9833296977458316,
          0.9710100057905164,
          0.9572316430523317,
          0.9426886049809743,
          0.9277253536938298,
          0.8831486876894581,
          0.817811304643707,
          0.7655484543652669,
          0.7198866407284839,
          0.643506361425598,
          0.5928197767210946,
          0.5454732370629979]
```

```
In [20]: pillar_dates
```

```
Out[20]: [datetime.date(2016, 11, 23),
          datetime.date(2016, 12, 23),
          datetime.date(2017, 1, 23),
          datetime.date(2017, 2, 23),
          datetime.date(2017, 3, 23),
          datetime.date(2017, 4, 23),
          datetime.date(2017, 5, 23),
          datetime.date(2017, 6, 23),
          datetime.date(2017, 7, 23),
          datetime.date(2017, 8, 23),
          datetime.date(2017, 9, 23),
          datetime.date(2017, 10, 23),
          datetime.date(2017, 11, 23),
          datetime.date(2018, 2, 23),
          datetime.date(2018, 5, 23),
          datetime.date(2018, 8, 23),
 datetime.date(2018, 11, 23),
 datetime.date(2019, 11, 23),
 datetime.date(2020, 11, 23),
 datetime.date(2021, 11, 23),
 datetime.date(2022, 11, 23),
 datetime.date(2023, 11, 23),
 datetime.date(2024, 11, 23),
 datetime.date(2025, 11, 23),
 datetime.date(2026, 11, 23),
 datetime.date(2027, 11, 23),
 datetime.date(2028, 11, 23),
 datetime.date(2031, 11, 23),
 datetime.date(2036, 11, 23),
 datetime.date(2041, 11, 23),
 datetime.date(2046, 11, 23),
 datetime.date(2056, 11, 23),
 datetime.date(2066, 11, 23),
 datetime.date(2076, 11, 23)]
```

2.2 Exercises

2.2.1 Exercise 5.2

Take the `OvernightIndexSwap` class from the lesson and add a new method called fair_value_strike which takes a discount curve object and returns the fixed rate which would make the OIS have zero NPV.

Hints:

- first take the formulas for the NPV of the fixed leg and the NPV of the floating leg, put one equal to the other and solve for K;
- then implement that in Python.

2.2.2 Exercise 5.3

Take the `OvernightIndexSwap` class, add it to `finmarkets.py` and try importing and using it.