Swaps and Modules - Practical Lesson 5

Matteo Sani
matteosan1@gmail.com

October 8, 2019

1 Swaps and Modules

1.1 Recap

- basic Python (mostly not related directly to finance)
- how to implement a discount factor interpolation function
- qrapping 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.

2 Modules

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! Take a look at this video (https://www.youtube.com/watch?v=AqCl65wxikw) for an example of how it's done for Jupyter notebook.

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.

So first of all let's create a new file called finmarkets.py and copy into it the `DiscountCurve` class we wrote last time.

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

        curve = DiscountCurve(date(2019, 1, 1),
                              [date(2019, 1, 1),
                               date(2019, 6, 1),
                               date(2020, 1, 1)],
                          [1.0, 0.98, 0.82])
        curve.df(date(2019, 7, 1))

Out[1]: 0.9558151167629666
```

We will use this discount curve later in this lesson.

3 Overnight Index Swap

Overnight Index Swap (OIS) are products which pay a floating coupon, determined by overnight rate fixings over the reference periods, against a fixed coupon. We will always look at these products from the point of view of the **receiver of the floating leg**. Therefore 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

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 a 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 cashflows 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

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

where $D_{ois}(d)$ is the discount factor implied by OIS prices.

In a previous theory lesson we mentioned that the correct curve to use for discounting the flows of a collateralized contract 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

$$\text{NPV}_{\text{float}} = N\cdot\sum_{i=1}^{n}[D(d_{i-1}) - D(d_i)] = N\cdot[D(d_0) - D(d_n)]$$

Each cash flow of the fixed leg is equal to

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

so the NPV of the fixed leg is

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

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 function (or rather a class), which takes discount curve as the input and produces the net present value (NPV) of the OIS as the output. Next lesson we'll put this function inside a numerical optimizer to invert the process and hence to determine the implied discount factors from the prices.

```
In [2]: class OvernightIndexSwap(object):

            # 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(2019, 1, 1),
             date(2019, 4, 1),
             date(2019, 7, 1),
             date(2019, 10, 1),
             date(2020, 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. Let's now evaluate the NPV of the OIS.

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

```
Out[4]: 173824.80713628858
```

3.1 Exercises

3.1.1 Exercise 5.1

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.

3.1.2 Exercise 5.2

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

3.1.3 Exercise 5.3

In the next lesson we're going to build lots of `OvernightIndexSwap` objects, one for each market quote we have. The market quotes will 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

2016-11-17 start date 12 months → 2016-11-17, 2017-11-17 2016-11-17 start date 24 months → 2016-11-17, 2017-11-17, 2018-11-17

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

2016-11-17 start date 9 months → 2016-11-17, 2017-08-17 2016-11-17 start date 15 months → 2016-11-17, 2017-11-17, 2018-02-17

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

```python
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(2016, 11, 17), 12) == [date(2016, 11, 17),
                                                        date(2017, 11, 17)]
assert generate_swap_dates(date(2016, 11, 17), 24) == [date(2016, 11, 17),
                                                        date(2017, 11, 17),
                                                        date(2018, 11, 17)]

assert generate_swap_dates(date(2016, 11, 17), 9) == [date(2016, 11, 17),
                                                       date(2017, 8, 17)]
assert generate_swap_dates(date(2016, 11, 17), 15) == [date(2016, 11, 17),
                                                        date(2017, 11, 17),
                                                        date(2018, 2, 17)]
```