Credit Default Swaps - Practical Lesson 7

Matteo Sani
matteosan1@gmail.com

November 4, 2019

1 Catching up

1.1 Updating the `finmarkets` Module

Before going on talking about credit curves let's update our financial module by adding the objects we derived in the latest lectures:

- update the `generate_swap_dates` function for generic tenors
- `ForwardRateCurve` class
- `InterestRateSwap` class
- `InterestRateSwaption` class with two attributes (exercise date and irs instanciation) and two methods two compute npv with Black-Scholes formula or by MC simulation

```
In [1]: class InterestRateSwaption:

            def __init__(self, exercise_date, irs):
                self.exercise_date = exercise_date
                self.irs = irs

            def npv_bs(self, discount_curve, libor_curve, sigma):

                A = self.irs.annuity(discount_curve)
                S = self.irs.swap_rate(discount_curve, libor_curve)

                T = (self.exercise_date - discount_curve.today).days / 365

                d1 = (math.log(S/self.irs.fixed_rate) + 0.5 * sigma**2 * T) / (sigma * T**0.5)
                d2 = d1 - (sigma * T**0.5)

                npv = self.irs.notional * A * (S * scipy.stats.norm.cdf(d1) -
                                               self.irs.fixed_rate * scipy.stats.norm.cdf(d2))

                return npv

            def npv_mc(self, discount_curve, libor_curve, sigma, n_scenarios=10000):
```

```
                A = self.irs.annuity(discount_curve)
                S = self.irs.swap_rate(discount_curve, libor_curve)

                T = (self.exercise_date - discount_curve.today).days / 365
                discounted_payoffs = []

                for i_scenario in range(n_scenarios):
                    S_simulated = S * math.exp(-0.5 * sigma * sigma * T +
                                               sigma * math.sqrt(T) * numpy.random.normal())

                    swap_npv = self.irs.notional * (S_simulated - self.irs.fixed_rate) * A
                    discounted_payoffs.append(max(0, swap_npv))

                npv_mc = numpy.mean(discounted_payoffs)
                npv_error = 3 * numpy.std(discounted_payoffs) / math.sqrt(n_scenarios)

                return npv_mc, npv_error
```

```
In [2]: import finmarkets
        dir(finmarkets)
```

```
Out[2]: ['DiscountCurve',
         'ForwardRateCurve',
         'InterestRateSwap',
         'InterestRateSwaption',
         'OvernightIndexSwap',
         '__builtins__',
         '__cached__',
         '__doc__',
         '__file__',
         '__loader__',
         '__name__',
         '__package__',
         '__spec__',
         'generate_swap_dates',
         'math',
         'numpy',
         'relativedelta']
```

1.2 Credit curves

Just like a discount curve is a way of representing the underlying interest rates implicit in the market quotes of a collection of real-world interest rate products, **credit curves** are a way of representing the data implied by credit default swaps.

Credit default swaps (**CDS**) are instruments whose value depends on the likelihood that a given company (the curve's **issuer**) will suffer a credit event over a given period.

A **credit event** can be a default, the failure to make payments, the issuer entering into bankruptcy proceedings, or the occurence of other legal events. The exact definition of what con-

stitutes a credit event depends on a series of factors and is usually defined in some kind of ISDA (International Swaps and Derivatives Association) master agreement.

In any case, we will generically call a credit event a *default*, and talk about **non-default probabilities** (**NDP**), i.e. the probability that the issuer will not suffer a credit event before a given value date.

NDPs are the equivalent for credit curves of discount factors for discount curves. Just like discount curves, credit curves are built by specifying a pricing/observation date, a sequence of pillar dates and a sequence of NDPs. We will then implement a `CreditCurve` class that provides a method which log-linearly interpolates these pillar NDPs to return the NDP at an arbitrary value date between the pricing date and the last pillar date.

In addition, we'll also write a method which returns the **hazard rate** at an arbitrary value date. The hazard rate is the credit curve equivalent of the short rate or overnight rate for discount curves. It represents the instantaneous probability of the issuer defaulting conditioned on it not having defaulted until that moment - though in practice we'll calculate it numerically, and therefore it'll be the (annualized) conditional probability of the issuer defaulting between the value date and the day after.

Discount Curve
Credit Curve
Represents underlying rates implicit in market quotes of IR products
Represents default probability implied by credit default swaps
discount factors
non-default probabilities
short rate
hazard rate

1.2.1 Hazard Rate

Hazard rate is often called a *conditional failure rate* since it's expression is a direct application of the conditional probability concept.

Conditional probability answers to the question "how should you update probabilities of events when there is additional information available ?". To derive the general formula let's start with an example.

A fair die is rolled. Let A be the event that the outcome is an odd number ($A = 1, 3, 5$). Also let B be the event that the outcome is less than or equal to 3 ($B = 1, 2, 3$). What is the probability of A ($P(A)$) ? What is the probability of A given B ($P(A|B)$) ?

Being a simple example we can compute the result by hand:

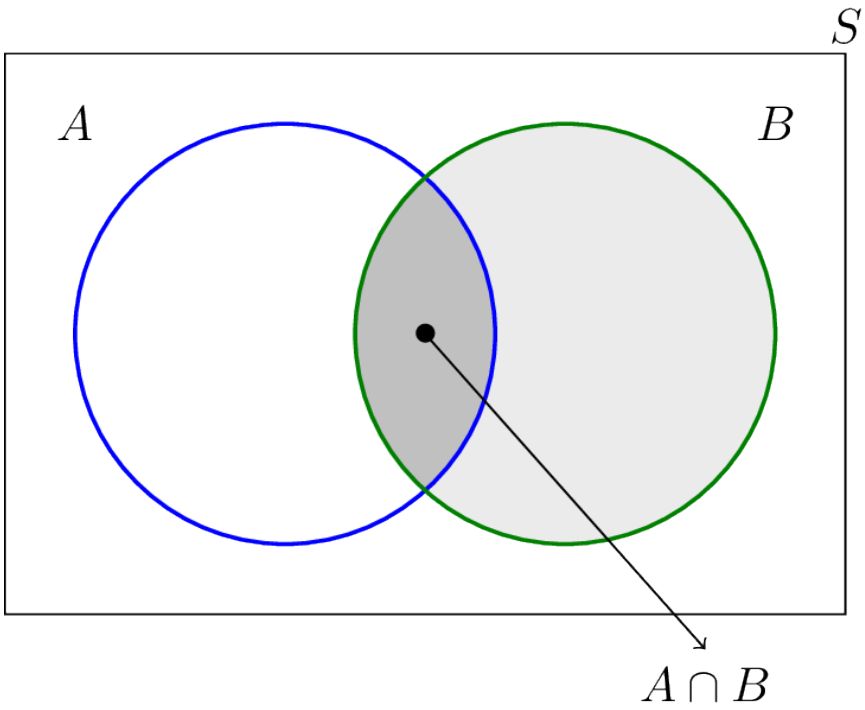
$$P(A) = \frac{|A|}{|S|} = \frac{|\{1,3,5\}|}{6} = \frac{1}{2} \text{ (where S is the entire sample space)}$$

Now let's find the conditional probability of A given that B occurred. If we know B has occurred, the outcome must be among $\{1, 2, 3\}$. For A to also happen the outcome must be in $A \cap B = \{1, 3\}$. Since all die rolls are equally likely, we argue that $P(A|B)$ must be equal to

$$P(A|B) = \frac{|A \cap B|}{|B|} = \frac{2}{3}$$

To generalize our example we can rewrite the calculation by dividing the numerator and denominator by the entire space of the events $|S|$ hence:

$$P(A|B) = \frac{|A \cap B|}{|B|} = \frac{\frac{|A \cap B|}{|S|}}{\frac{|B|}{|S|}} = \frac{P(A \cap B)}{P(B)}$$



$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

In formula if the non-default probability is indicated by N and the hazard rate by λ:

$$\lambda = -\frac{dN}{dt}\frac{1}{N(t_0, t_1)}$$

where the minus sign derives from the fact that N is a **non** default probability while the hazard rate is defined in terms of the probability of default.

Conversly given the hazard rate the non-default probability can be determined as:

$$\lambda = -\frac{1}{dt} \cdot \frac{dN}{N} = -\frac{d(\log N)}{dt}$$

$$N(t_0, t) = e^{-\int_{t_0}^{t} \lambda dt}$$

```
In [3]: import math
        import numpy

        class CreditCurve(object):

            def __init__(self, pillar_dates, pillar_ndps):
```

```
            self.pillar_dates = pillar_dates

            self.pillar_days = [
                (pd - pillar_dates[0]).days
                for pd in pillar_dates
            ]

            self.log_ndps = [
                math.log(ndp)
                for ndp in pillar_ndps
            ]

        def ndp(self, value_date):

            value_days = (value_date - self.pillar_dates[0]).days

            return math.exp(
                numpy.interp(value_days,
                             self.pillar_days,
                             self.log_ndps))

        def hazard(self, value_date):
            ndp_1 = self.ndp(value_date)
            ndp_2 = self.ndp(value_date + relativedelta(days=1))
            delta_t = 1.0 / 365.0
            h = -1.0 / ndp_1 * (ndp_2 - ndp_1) / delta_t
            return h
```

```
In [4]: from datetime import date
        from dateutil.relativedelta import relativedelta
        from curve_data import pricing_date

        cc = CreditCurve(
            [pricing_date, pricing_date + relativedelta(years=2)],
            [1.0, 0.8]
        )
```

```
In [5]: cc.ndp(pricing_date + relativedelta(years=1))
```

```
Out[5]: 0.8942906859183223
```

```
In [6]: cc.hazard(pricing_date + relativedelta(years=1))
```

```
Out[6]: 0.11140214262993799
```

1.3 Credit Deafult Swaps

Once we have implemented a `CreditCurve` class which allows us to interpolate pillar non-default probabilities (NDPs) at arbitrary value dates, and also allows us to calculate the hazard rate at an arbitrary date, we can use it to price **credit default swaps** (CDSs).

CDSs are made up of two legs:

- the *default* leg: which pays $L = 1 - R$, known as the **loss given default** (LGD) if and when the credit event occurs, where $R = 40\%$ is a conventional **recovery value**;
- the *premium* leg: which pays S (*spread*) every 3 months until the credit event occurs.

1.3.1 Premium leg

Let's start with the premium leg, which is easier to calculate. We'll use the following notation:

- d today's date
- d_0 the start date of the CDS (could be different from d)
- $d_1, ..., d_n$ the payment dates of the premium leg, which occur at a 3-month frequency
- we assume that d_n is the end date of the CDS
- $D(d')$ the discount factor between d and d'
- $N(d')$ the non-default probability between d and d'
- τ the random variable representing the date of the credit event

At each payment date d_i, a flow of S is paid if and only if the credit event has not occurred before that date. Therefore the NPV of the each flow is

$$\tilde{\mathbb{E}}\left[S \times D(d_i) \times \mathbb{1}\{\tau > d_i\}\right] = S \times D(d_i) \times N(d_i)$$

therefore the NPV of the premium leg is simply the sum, over the payment dates, of these terms:

$$\text{NPV}_{premium} = \sum_{i=1}^{n} S \times D(d_i) \times N(d_i)$$

1.3.2 Default leg

The LGD $1 - R$ is paid out on the same date on which the credit event occurs, i.e. it can potentially be paid out on any date between d_0 and d_n. Mathematically, therefore, the NPV of the premium leg can be expressed as follows:

$$\tilde{\mathbb{E}}[(1 - R) \times D(\tau) \times \mathbb{1}\{\tau \leq d_n\}]$$

Using the law of total probability, we can break this down into the sum of "daily NPVs" calculated as a function of the daily forward default probabilities:

$$\begin{aligned}
\tilde{\mathbb{E}}[(1 - R) \times D(\tau) \times \tau \leq d_n] &= \sum_{d'=d_0}^{d_n} \tilde{\mathbb{E}}[(1 - R) \times D(\tau) \mid \tau = d']\mathbb{P}[\tau = d'] \\
&= (1 - R) \sum_{d'=d_0}^{d_n} D(d') \left(\mathbb{P}[\tau \geq d'] - \mathbb{P}[\tau \geq d' + 1]\right) \\
&= (1 - R) \sum_{d'=d_0}^{d_n} D(d') \left(N(d') - N(d' + 1)\right)
\end{aligned}$$

In [7]:
```
from finmarkets import generate_swap_dates

class CreditDefaultSwap:

    def __init__(self, notional, start_date, nyears, fixed_spread, recovery=0.4):

        self.notional = notional
        self.payment_dates = generate_swap_dates(start_date, nyears*12, 3)
        self.fixed_spread = fixed_spread
        self.recovery = recovery

    def premium_leg_npv(self, discount_curve, credit_curve):

        npv = 0

        for i in range(1, len(self.payment_dates)):
            npv += (
                self.fixed_spread *
                discount_curve.df(self.payment_dates[i]) *
                credit_curve.ndp(self.payment_dates[i])
            )

        return npv * self.notional

    def default_leg_npv(self, discount_curve, credit_curve):

        npv = 0
        d = self.payment_dates[0]

        while d <= self.payment_dates[-1]:

            npv += discount_curve.df(d) * (
                credit_curve.ndp(d) -
                credit_curve.ndp(d + relativedelta(days=1))
            )

            d += relativedelta(days=1)

        return npv * self.notional * (1 - self.recovery)

    def npv(self, discount_curve, credit_curve):

        return self.default_leg_npv(discount_curve, credit_curve) - \
               self.premium_leg_npv(discount_curve, credit_curve)
```

In [8]:
```
from curve_data import discount_curve, pricing_date
from dateutil.relativedelta import relativedelta
```

```
        credit_curve = CreditCurve([pricing_date, pricing_date + relativedelta(months=36)],
                                   [1.0, 0.7])

        cds = CreditDefaultSwap(1e6, pricing_date, 3, 0.03)
        cds.premium_leg_npv(discount_curve, credit_curve)
```

```
Out[8]: 300505.06774906756
```

```
In [9]: cds.default_leg_npv(discount_curve, credit_curve)
```

```
Out[9]: 181369.30417135215
```

```
In [10]: cds.npv(discount_curve, credit_curve)
```

```
Out[10]: -119135.7635777154
```

1.4 Exercises

1.4.1 Exercise 7.1

Applying a bootstrapping technique (outlined in lesson 5) derive the a credit curve from the following CDS market quotes:

```
pricing_date = date(2019, 11, 6)

cds_quotes = [
    {'maturity': 12, 'spread':0.0149},
    {'maturity': 24, 'spread':0.0165},
    {'maturity': 36, 'spread':0.0173},
    {'maturity': 69, 'spread':0.0182},
    {'maturity': 120, 'spread':0.0183},
    {'maturity': 240, 'spread':0.0184},
]
```

Hint

- create a CDS contract for each input market quote;
- implement an objective function to minimize the squared sum of the CDS npvs, the function has to implement also a `CreditCurve` with the unknown (to be detrmined by the bootstrap survival probabilities) and a list of pillars corresponding to the CDS maturities;
- remember to set initial guesses and boundary conditions for the unknown parameters (the first survival probability has to be set to 1 since no default happened !, the others free to move between [0.01 to 1]);
- using the `scipy.optimize.minimize` find the curve.

1.4.2 Exercise 7.2

Using the above `Credit Curve` and the `DiscountCurve` already defined in lesson 5, price the following CDS:

```
cds_to_price = [
    {'nominal': 5000000, 'maturity': 18, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 30, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 42, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 72, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 108, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 132, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 160, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 184, 'spread': 0.02},
    {'nominal': 5000000, 'maturity': 210, 'spread': 0.02}
]
```