Interest Rate Model Calibration and Pricing Caps and Floors with QuantLib

Financial Algorithm- Final Project

台科大 M10808050 呂偉丞

Contents

01

Hull White Model and Black Karasinski Model Calibration

02

Pricing Caps with Constant volatility

03

Pricing Caps with volatility surface

01 Hull White Model and Black Karasinski Model Calibration

首先會透過swaption 的market data 做calibration

WHY swaption?

會用swaption而不是cap/floor的原因有以下:

1. Swaption 流動性好
2. 交易量大
3. Information set 豐富

經過前面的環境設定，之後再define一個function建立swaptions

```
In [4]: def create_swaption_helpers(data, index, term_structure, engine):

            swaptions = []
            fixed_leg_tenor = ql.Period(1, ql.Years)
            fixed_leg_daycounter = ql.Actual360()
            floating_leg_daycounter = ql.Actual360()

            for d in data:
                vol_handle = ql.QuoteHandle(ql.SimpleQuote(d.volatility))
                helper = ql.SwaptionHelper(ql.Period(d.start, ql.Years),
                                           ql.Period(d.length, ql.Years),
                                           vol_handle,
                                           index,
                                           fixed_leg_tenor,
                                           fixed_leg_daycounter,
                                           floating_leg_daycounter,
                                           term_structure
                                           )
                helper.setPricingEngine(engine)
                swaptions.append(helper)
            return swaptions
```

01 Hull White Model and Black Karasinski Model Calibration

(二)Hull & White(1993)

- Hull & White 模型的短期利率連續極限如下，

$$dr = [\theta(t) - \alpha r]\cdot dt + \sigma \cdot dz \quad \text{......(8.4)}$$

- 在 HW 模型中只有二個參數，可由市場上交易的利率選擇權價格來推估。
  - 假設市場上有 M 個零息債券選擇權，其價格分別為 Market_i，I=1…M。令 $\text{Model}(\alpha，\sigma)_i$ 為由 HW 模型所求的的價格，則如下校準參數，

$$\min_{\alpha,\sigma}\sqrt{\sum_{i=1}^{M}\left(\frac{\text{model}_i(\alpha,\sigma)-\text{market}_i}{\text{market}_i}\right)^2}$$

define 一個function做calibrate

```
In [5]: def calibration_report(swaptions, data):

    print("-"*82)
    print("%15s %15s %15s %15s %15s" % \
    ("Model Price", "Market Price", "Implied Vol", "Market Vol", "Rel Error"))
    print("-"*82)


    cum_err = 0.0
    for i, s in enumerate(swaptions):
        model_price = s.modelValue()
        market_vol = data[i].volatility
        black_price = s.blackPrice(market_vol)
        rel_error = model_price/black_price - 1.0
        implied_vol = s.impliedVolatility(model_price,
                                          1e-5, 50, 0.0, 0.50)
        rel_error2 = implied_vol/market_vol-1.0
        cum_err += rel_error2*rel_error2

        print("%15.5f %15.5f %15.5f %15.5f %15.5f" % \
        (model_price, black_price, implied_vol, market_vol, rel_error))
    print("-"*82)
    print("Cumulative Error : %15.5f" % math.sqrt(cum_err))
```

01 Hull White Model and Black Karasinski Model Calibration

In [6]:
```
model = ql.HullWhite(term_structure);
engine = ql.JamshidianSwaptionEngine(model)
swaptions = create_swaption_helpers(data, index, term_structure, engine)

optimization_method = ql.LevenbergMarquardt(1.0e-8,1.0e-8,1.0e-8)
end_criteria = ql.EndCriteria(10000, 100, 1e-6, 1e-8, 1e-8)
model.calibrate(swaptions, optimization_method, end_criteria)

a, sigma = model.params()
print("alpha = %6.5f, sigma = %6.5f" % (a, sigma))
calibration_report(swaptions, data)
```

```
alpha = 0.04915, sigma = 0.00584
----------------------------------------------------------------------
    Model Price    Market Price     Implied Vol      Market Vol      Rel Error
----------------------------------------------------------------------
        0.00880         0.00951         0.10620         0.11480       -0.07488
        0.00967         0.01007         0.10632         0.11080       -0.04039
        0.00866         0.00871         0.10635         0.10700       -0.00606
        0.00650         0.00623         0.10644         0.10210        0.04234
        0.00354         0.00332         0.10659         0.10000        0.06561
----------------------------------------------------------------------
Cumulative Error :          0.11594
```

由圖可知經過calibration，得到所需參數

$$a = 0.04915$$

$$\sigma = 0.00584$$

得到Model Price、Market Price、Implied vol、Market vol

由此計算模型價格與市場價格誤差

01 Hull White Model and Black Karasinski Model Calibration

(四)Black & Karasinski(BK, 1991)

◆ BK 模型的短期利率連續極限如下，

$$d\ln r = [\theta(t) - \alpha(t)\ln r]\cdot dt + \sigma(t)\cdot dz \quad (8.8)$$

➢ 令波動性為常數可簡化模型成為，

$$d\ln r(t) = [\theta(t) - \alpha\ln r]\cdot dt + \sigma\cdot dz$$

In [7]:
```
model = ql.BlackKarasinski(term_structure);
engine = ql.TreeSwaptionEngine(model, 100)
swaptions = create_swaption_helpers(data, index, term_structure, engine)

optimization_method = ql.LevenbergMarquardt(1.0e-8,1.0e-8,1.0e-8)
end_criteria = ql.EndCriteria(10000, 100, 1e-6, 1e-8, 1e-8)
model.calibrate(swaptions, optimization_method, end_criteria)

a, sigma =  model.params()
print("alpha = %6.5f, sigma = %6.5f" % (a, sigma))
```
```
alpha = 0.04162, sigma = 0.11782
```

In [8]:
```
calibration_report(swaptions, data)
```

Model Price	Market Price	Implied Vol	Market Vol	Rel Error
0.00874	0.00951	0.10550	0.11480	-0.08097
0.00967	0.01007	0.10633	0.11080	-0.04026
0.00867	0.00871	0.10655	0.10700	-0.00423
0.00651	0.00623	0.10665	0.10210	0.04443
0.00355	0.00332	0.10675	0.10000	0.06714

Cumulative Error : 0.12147

由圖可知經過calibration，得到所需參數

$$a = 0.04162$$

$$\sigma = 0.11782$$

得到Model Price、Market Price、Implied vol、Market vol

由此計算模型價格與市場價格誤差

02 Pricing Caps with Constant volatility

這之後會以定價Caps為主題，分別以兩種cases作探討

1. Pricing Caps with Constant volatility
2. Pricing Caps with volatility surface

WHY?

因為Caps是由多個caplets組成，每一個caplet都應該要有不同的volatility，因此會先以constant volatility再延生到volatility surface

02 Pricing Caps with Constant volatility

透過QuantLib評價Caps的步驟:

1. construct interest rate term structure for discounting
2. construct interest rate term structure for the floating leg
3. construct the pricing engine to value caps

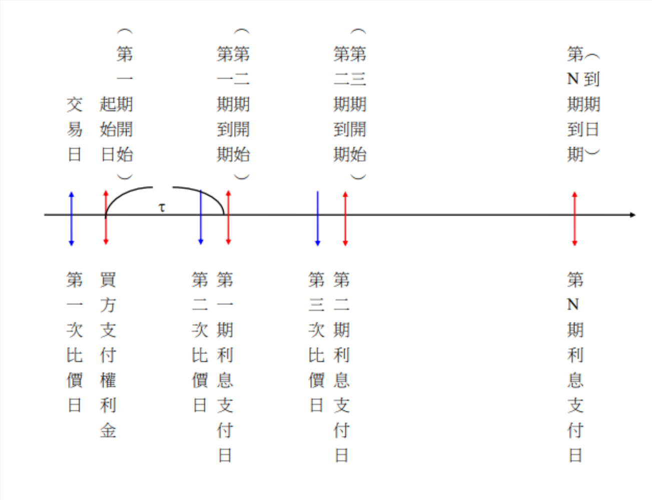
```
In [9]: calc_date = ql.Date(14, 6, 2021)
        ql.Settings.instance().evaluationDate = calc_date
```

```
In [10]: dates = [ql.Date(14,6,2021), ql.Date(14,9,2021),
                  ql.Date(14,12,2021), ql.Date(14,6,2022),
                  ql.Date(14,6,2024), ql.Date(14,6,2026),
                  ql.Date(15,6,2031), ql.Date(16,6,2036),
                  ql.Date(16,6,2041), ql.Date(14,6,2051)
                  ]
         yields = [0.000000, 0.006616, 0.007049, 0.007795,
                   0.009599, 0.011203, 0.015068, 0.017583,
                   0.018998, 0.020080]
         day_count = ql.ActualActual()
         calendar = ql.Taiwan()
         interpolation = ql.Linear()
         compounding = ql.Compounded
         compounding_frequency = ql.Annual

         term_structure = ql.ZeroCurve(dates, yields, day_count, calendar,
                                       interpolation, compounding, compounding_frequency)

         ts_handle = ql.YieldTermStructureHandle(term_structure)
```

02 Pricing Caps with Constant volatility

construct 一個十年期，每三個月比價一次的caps

```python
In [11]: start_date = ql.Date(14, 6, 2021)
         end_date = ql.Date(14, 6 , 2031)
         period = ql.Period(3, ql.Months)
         calendar = ql.Taiwan()
         buss_convention = ql.ModifiedFollowing
         rule = ql.DateGeneration.Forward
         end_of_month = False

         schedule = ql.Schedule(start_date, end_date, period,
                                calendar, buss_convention, buss_convention,
                                rule, end_of_month)
```



02 Pricing Caps with Constant volatility

```
In [12]: ibor_index = ql.USDLibor(ql.Period(3, ql.Months), ts_handle)
         ibor_index.addFixing(ql.Date(10,6,2021), 0.0065560)

         ibor_leg = ql.IborLeg([1000000], schedule, ibor_index)

In [13]: strike = 0.02
         cap = ql.Cap(ibor_leg, [strike])

         vols = ql.QuoteHandle(ql.SimpleQuote(0.547295))
         engine = ql.BlackCapFloorEngine(ts_handle, vols)

         cap.setPricingEngine(engine)

In [14]: cap.NPV()

Out[14]: 54408.95638684406
```

有了以上materials，即可計算出cap=54408.95

03 Pricing Caps with volatility surface

接著計算每個caplet都有不同的volatility

Construct a volatility matrix with different expires and strikes

In [15]:
```
strikes = [0.01,0.015, 0.02]
expiries = [ql.Period(i, ql.Years) for i in range(1,11)] + [ql.Period(12, ql.Years)]
vols = ql.Matrix(len(expiries), len(strikes))
data = [[47.27, 55.47, 64.07, 70.14, 72.13, 69.41, 72.15, 67.28, 66.08, 68.64, 65.83],
   [46.65,54.15,61.47,65.53,66.28,62.83,64.42,60.05,58.71,60.35,55.91],
   [46.6,52.65,59.32,62.05,62.0,58.09,59.03,55.0,53.59,54.74,49.54]
   ]

for i in range(vols.rows()):
    for j in range(vols.columns()):
        vols[i][j] = data[j][i]/100.0
```

03 Pricing Caps with volatility surface

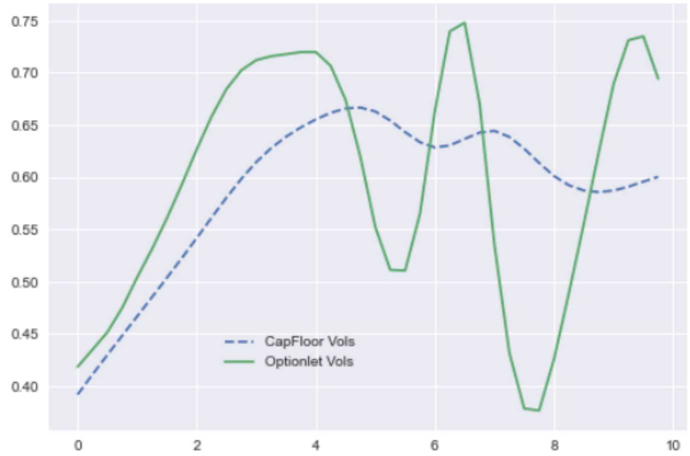
透過ql.OptionletStripper1 將每個caplet/floorlet volatility從capfloor vol中分開，再透過ql.StrippedOptionletAdapter組成新的optionlet vol才能建構term structure

透過matplotlib畫出capfloor vol and optionlet vol

In [17]:
```
optionlet_surf = ql.OptionletStripper1(capfloor_vol, ibor_index)
ovs_handle = ql.OptionletVolatilityStructureHandle(
    ql.StrippedOptionletAdapter(optionlet_surf)
)
```

In [18]:
```
tenors = np.arange(0,10,0.25)
strike = 0.015
capfloor_vols = [capfloor_vol.volatility(t, strike) for t in tenors]
opionlet_vols = [ovs_handle.volatility(t, strike) for t in tenors]

plt.plot(tenors, capfloor_vols, "--", label="CapFloor Vols")
plt.plot(tenors, opionlet_vols,"-", label="Optionlet Vols")
plt.legend(bbox_to_anchor=(0.5, 0.25))
```

Out[18]: <matplotlib.legend.Legend at 0x2076ebf8910>



03 Pricing Caps with volatility surface

```
In [19]: engine2 = ql.BlackCapFloorEngine(ts_handle, ovs_handle)
         cap.setPricingEngine(engine2)

In [20]: cap.NPV()

Out[20]: 54427.26567992578
```

即可將optionlet volatility surface評價caps or floors

由此可知cap=54427.26


Thanks