1 Modeling Fixed Rate Bonds Using QuantLib

Let's consider a hypothetical bond with a par value of 100, that pays 6% coupon semi-annually issued on January 15th, 2015 and set to mature on January 15th, 2016. The bond will pay a coupon on July 15th, 2015 and January 15th, 2016. The par amount of 100 will also be paid on the January 15th, 2016.

To make things simpler, lets assume that we know the spot rates of the treasury as of January 15th, 2015. The annualized spot rates are 0.5% for 6 months and 0.7% for 1 year point. Lets calculate the fair value of this bond.

1.1 refs

- source materials of above codes
  - [http://gouthamanbalaraman.com/blog/quantlib-bond-modeling.html (http://gouthamanbalaraman.com/blog/quantlib-bond-modeling.html)](http://gouthamanbalaraman.com/blog/quantlib-bond-modeling.html)
  - [https://letyourmoneygrow.com/2018/04/14/quantlib-python-twisting-a-snake-to-fit-a-yieldcurve/ (https://letyourmoneygrow.com/2018/04/14/quantlib-python-twisting-a-snake-to-fit-a-yieldcurve/)](https://letyourmoneygrow.com/2018/04/14/quantlib-python-twisting-a-snake-to-fit-a-yieldcurve/)
- a improved machine learning method to estimate term structure
  - [https://arxiv.org/pdf/1703.01536.pdf (https://arxiv.org/pdf/1703.01536.pdf)](https://arxiv.org/pdf/1703.01536.pdf)

1.2 quantlib installation

```
brew install boost
brew install quantlib
pip install QuantLib-Python
```

In [1]:
```python
3/pow(1+0.005, 0.5) + (100 + 3)/(1+0.007)
```

Out[1]:
```
105.27653992490681
```

Lets calculate the same using QuantLib

In [2]:
```python
import QuantLib as ql
```

In [3]:
```python
todaysDate = ql.Date(15, 1, 2015)
ql.Settings.instance().evaluationDate = todaysDate
spotDates = [ql.Date(15, 1, 2015), ql.Date(15, 7, 2015), ql.Date(15, 1, 2016)]
spotRates = [0.0, 0.005, 0.007]
dayCount = ql.Thirty360()
calendar = ql.UnitedStates()
interpolation = ql.Linear()
compounding = ql.Compounded
compoundingFrequency = ql.Annual
spotCurve = ql.ZeroCurve(spotDates, spotRates, dayCount, calendar, interpolation,
                         compounding, compoundingFrequency)
spotCurveHandle = ql.YieldTermStructureHandle(spotCurve)
```

In [4]:
```python
issueDate = ql.Date(15, 1, 2015)
maturityDate = ql.Date(15, 1, 2016)
tenor = ql.Period(ql.Semiannual)
calendar = ql.UnitedStates()
bussinessConvention = ql.Unadjusted
dateGeneration = ql.DateGeneration.Backward
monthEnd = False
schedule = ql.Schedule (issueDate, maturityDate, tenor, calendar, bussinessConvention
                            bussinessConvention , dateGeneration, monthEnd)
list(schedule)
```

Out[4]:
```
[Date(15,1,2015), Date(15,7,2015), Date(15,1,2016)]
```

In [5]:
```python
# Now lets build the coupon
dayCount = ql.Thirty360()
couponRate = .06
coupons = [couponRate]

# Now lets construct the FixedRateBond
settlementDays = 0
faceValue = 100
fixedRateBond = ql.FixedRateBond(settlementDays, faceValue, schedule, coupons, dayCou

# create a bond engine with the term structure as input;
# set the bond to use this bond engine
bondEngine = ql.DiscountingBondEngine(spotCurveHandle)
fixedRateBond.setPricingEngine(bondEngine)

# Finally the price
fixedRateBond.NPV()
```

Out[5]:
```
105.27653992490683
```

2 NS/NSS fitting using clean price, face value, etc.

In [27]:
```python
import numpy as np
import QuantLib as ql

today = ql.Date(8, 2, 2018)
ql.Settings.instance().evaluationDate = today

terminationDates = [ql.Date(4, 7, 2044), ql.Date(15, 2, 2028), ql.Date(14, 4, 2023)]
tenors = np.repeat(ql.Period(ql.Annual), 3) #allusion on R function rep()
calenders = np.repeat(ql.Germany(), 3)
termDateConvs = np.repeat(ql.Following, 3)
genRules = np.repeat(ql.DateGeneration.Backward, 3)
endOfMonths = np.repeat(False, 3)
firstDates = [ql.Date(27, 4, 2012), ql.Date(10, 1, 2018), ql.Date(2, 2, 2018)]

settlementDays = np.repeat(2, 3)
coupons = [0.025, 0.005, 0.0]
cleanPrices = [126.18, 98.18, 99.73]
faceValues = np.repeat(100.0, 3)
dayCounts = np.repeat(ql.ActualActual(), 3)

schedules = []
bonds = []
bondHelpers = []
for j in range(0, 3):
    # without int() and bool() conversion it will not work due to int vs. int32_ anc
    schedules.append(ql.Schedule(firstDates[j], terminationDates[j], tenors[j], cale
                                 int(termDateConvs[j]), int(termDateConvs[j]), int(g
                                 bool(endOfMonths[j])))
    bonds.append(ql.FixedRateBond(int(settlementDays[j]), float(faceValues[j]), sche
                                  [float(coupons[j])], dayCounts[j]))
    bondHelpers.append(ql.BondHelper(ql.QuoteHandle(ql.SimpleQuote(float(cleanPrices

list(schedules[0])
list(schedules[1])
list(schedules[2])

print(bonds[0].bondYield(float(cleanPrices[0]), dayCounts[0], ql.Compounded, ql.Annu
print(bonds[1].bondYield(float(cleanPrices[1]), dayCounts[1], ql.Compounded, ql.Annu
print(bonds[2].bondYield(float(cleanPrices[2]), dayCounts[2], ql.Compounded, ql.Annu

curveSettlementDays = 2
curveCalendar = ql.Germany()
curveDaycounter = ql.ActualActual()

#piecewise log cubic discount curve. Surprisingly there is no log-linear...
yieldCurve = ql.PiecewiseLogCubicDiscount(today, bondHelpers, curveDaycounter)
print(yieldCurve.discount(ql.Date(1, 3, 2019)))
print(yieldCurve.discount(ql.Date(1, 3, 2020)))
print(yieldCurve.discount(ql.Date(1, 3, 2035)))

##and Nelson-Siegel
#curveFittingMethod = ql.NelsonSiegelFitting()
curveFittingMethod = ql.SvenssonFitting()
tolerance = 1.0e-5
iterations = 1000
yieldCurveNS = ql.FittedBondDiscountCurve(curveSettlementDays, curveCalendar, bondHe
                                          curveDaycounter, curveFittingMethod, toler
res = yieldCurveNS.fitResults()
print(yieldCurveNS.discount(ql.Date(1, 3, 2019)))
print(yieldCurveNS.discount(ql.Date(1, 3, 2020)))
print(yieldCurveNS.discount(ql.Date(1, 3, 2035)))
```

```
0.013187208287715916
0.006888236205577852
0.0005233827483989923
0.9999769684998407
0.9998298241913117
0.8217469489308218
1.0089671485666836
1.0122250811058053
0.8178739222331985
```

In [28]:

```
#paramters returned
np.array(res.solution())
```

Out[28]:

```
array([-0.0001615 , -0.01141152,  0.04413113,  0.03121141,  0.04677576,
        0.11417798])
```

In [29]:

```
np.array(res.weights())
```

Out[29]:

```
array([0.22253298, 0.45770962, 0.86080252])
```

In [26]:

```
?yieldCurveNS.fitResults()
```

In [ ]:

Type *Markdown* and LaTeX: α^2

In [ ]: