MTH 9878 Interest Rate Models, Spring 2015

Programming Assignment 1

Constructed OIS and LIBOR curves with B-Splines implied by IR Swap, EuroDollar and Basis Swap

Instructor: Andrew Lesniewski

Group Members: Haotian Wu, Ray(Peiqin) He & Dov Sturm
Financial Engineering Program at Baruch College

Outline:

The purpose of this assignment is to build the LIBOR and OIS curves based on the actual closing market data as of December 13, 2011 (please see the enclosed Excel spreadsheet DataSheetCurve.xls), and use it for pricing swaps. I suggest that you:

- (A) Use Excel as your front end (but by no means you have to do it if you prefer other formats). Excel comes with graphing capabilities, and a number of date functions that will allow you generate the relevant dates and day count fractions for both 30/360 and act/360 conventions (you should not try do it yourself!). You can easily find these functions in the category “Date & Time” of Excel functions.
- (B) Ignore the holidays and end of the month issues (I chose the date so that that latter should not be an issue). This will lead to small inaccuracies but is going to be good enough.
- (C) All your functions should preferably be programmed in C++ or C#. In order to establish an interface in Excel (if you use it) you may want to use the open source package XLW (http://xlw.sourceforge.net (http://xlw.sourceforge.net)). You can also use VBA for Excel or Matlab but your code will run slower.
- (D) Don’t delete your files, you will need them later.

Problems:

- (1) Develop a small library implementing B-splines. This library should be capable of calculating the derivatives and integrals of the basis functions. Make sure that you take a full advantage of the recursive properties of the basis functions. Pay attention to computational performance issues: (i) cashe data whenever appropriate, (ii) take advantage of the support properties of the basis functions
- (2) Build a function that computes the discount factor between any two dates.
- (3) Build a function that computes the forward LIBOR rate for any settlement and underlying tenor.
- (4) Build a function that computes the (spot or forward) swap rate for any settlement and underlying tenor.
- (5) Use the enclosed market data sheet and the method described in class to build the instantaneous OIS and LIBOR curves. Make sure that three of the knot points of the B-spline lie to the left of t 0, choose t 0 = 0, choose t N to be close to 30 (say, t N = 31), and make sure that there are four auxiliary knot points to the right of t N. Experiment with the positions of the knot points using the following guidelines: (a) N should be around 10 - 12, (b) they should be close to each other in the short end of the curve and get sparser as we move further out. In order to complete the task you will need an optimizer: you can write your own Newton-Raphson style search algorithm or (even better) use Excel’s Solver, Matlab’s optimizer, etc. Plot the resulting curves.
- (6) Build a function that calculates the PV of any spot or forward starting swap based on your curves.

Code Running Prerequisite:

In [1]:

```
#Python Running Requirements
import xlrd
from xlwings import Workbook, Sheet, Range, Chart
import numpy as np
import sys
import os
sys.path.append(os.path.abspath("C://Users//a//Dropbox//interest_rate//Interest_Rate_HW//Big_Assignment//"))
from B_Spline import Rate_Functions, B_Spline
import Market_Data as mr
import ExcelDataReader as Excel
import Market_Data_Calibrator as opimizer
import matplotlib.pyplot as plt
%matplotlib inline
print "Python Running Requirements Done"
```

```
Python Running Requirements Done
```

Basic Idea:

To achieve this homework, we choose python instead of C++, because scipy has better optimizer to achieve the goal. And python also have many plugin to interact with EXCEL. Python also has nicer libarary for plotting curves. So, we think python is perfect to estabish this question. Ipython notebook is also wonderful for displaying ideas.

Our project has 5 parts:

- Implement class for B-Spline interpolation with function to calculate Basis Functions and its derivative and integral
- Implement class for calculating different rates like discount factor and libor forward rates
- Reading the real data from Excel
- Using optimizer in scipy to fit the curve according to the target function
- Construct the curves based on the optimization

Answer:

(1) B-Spline

B-Spline interpolation with function to calculate Basis Functions and its derivative and integral was implement in class B_Spline in python file B_Spline.py.

The code import is:

In [5]:

```
from B_Spline import B_Spline
```

(2,3,4) Function for discount factor, forward LIBOR and swap rate

All the rate functions are calculating in class Rate_Functions in in python file B_Spline.py.

- Function for discount factor is calculated by disc_factor(self, S, T, f):
- Function for forward LIBOR rate is calculated by libor_fwd_rate(self, S, T, l):
- Function for swap rate is calculated by swap_rate(self, T_0, tenor, f, l):

The code import is:

In [6]:

```
from B_Spline import Rate_Functions
```

(5) Optimization:

Modules for Reading Data from Excel:

In [5]:

```
import ExcelDataReader as Excel #read data from excel
import Market_Data as mr #model various kinds of market rates and perform fitting calculation
import Market_Data_Calibrator as opimizer #Class that performs optimization using scipy.optimize
```

Main Function and Optimization:

In [2]:

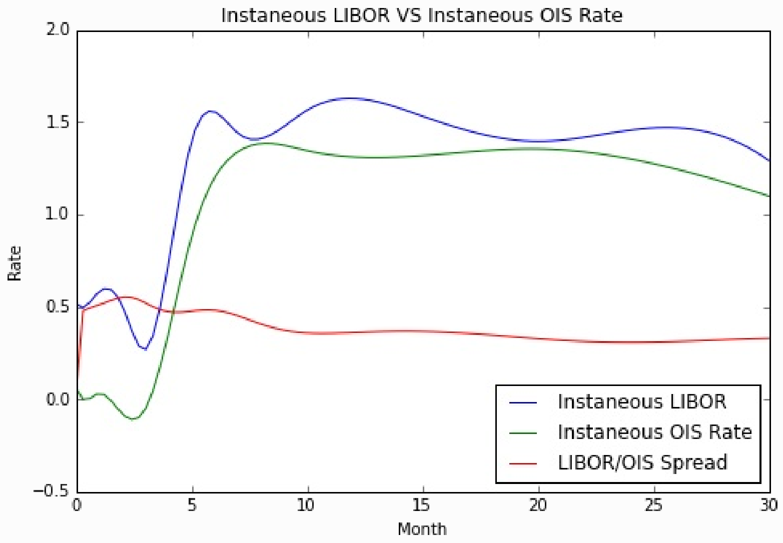
```
# ------------------ read data from spreadsheet ----------------
#Please change the path according to your own computer
wb = Workbook(r"C:\Users\a\Dropbox\interest_rate\Interest_Rate_HW\Big_Assignment\DataSheetCurve.xls")
reader = Excel.ExcelDataReader(r"C:\Users\a\Dropbox\interest_rate\Interest_Rate_HW\Big_Assignment\DataSheetCurve.xls")
reader.read_data_into_dictionary()

# --------------------------------------------------------------
#The years in our knot points
time_vector = range(-3,0) + [0, 1, 2, 3, 5, 7, 10, 15, 20, 25, 31] + range(32, 36)

#Convert into numpy array
time_vector = np.array(time_vector)
rc = Rate_Functions(time_vector)

#Start generating Market rate object using the input market rate
swapRateObj = mr.SwapRate(rc, **reader.swap_rate)
bSwapRateObj = mr.BasisSwapRate(rc, **reader.basis_swap_rate)
edObj = mr.EuroDollar(rc, **reader.ed_future)
liborObj = mr.LiborFixing(rc, **reader.libor)
ffrObj = mr.FedFundRate(rc, **reader.fed_fund)

# --------------Set up optimization parameters ------------------
fitobjs = [swapRateObj, bSwapRateObj, edObj, liborObj]

# Might need to re-visit the lambda
lambdas = np.array([0, 0, 0, 1e-5, 1e-5, 1e-3, 1e-3, 1e-2, 1e-1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

# create optimizer object
op = opimizer.MarketRateCalibrator(lambdas, time_vector, rc)

#Invoke the optimizar to find optimal f_k and l_k
xopt = op.optimize(*fitobjs)
n = op._n
fopt, lopt = xopt[:n], xopt[n:]

print "Optimization Done!"
```

```
Optimization Done!
```

Plot 1: Instaneous 3M LIBOR VS Instaneous OIS rate

In [3]:

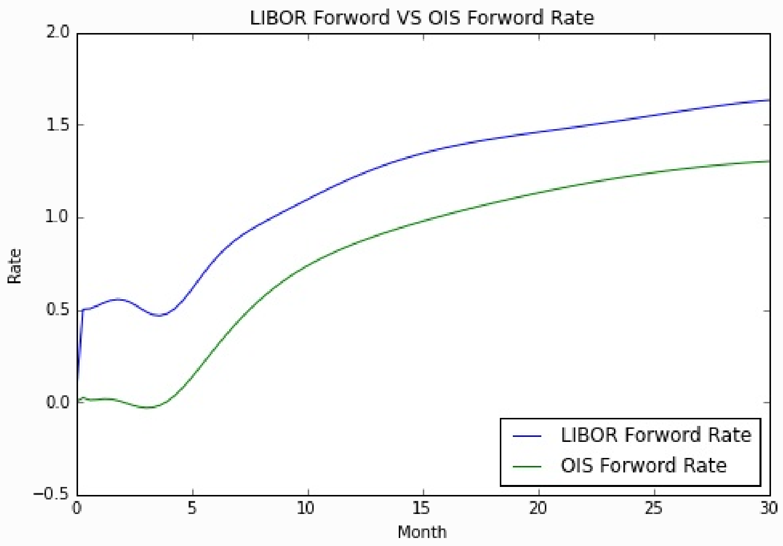
```
x1 = np.linspace(0, 30, 100)
y1 = map(lambda t: rc.inst_libor_fwd_rate(t, lopt), x1)
y2 = map(lambda t: rc.inst_ois_fwd_rate(t, fopt), x1)
y3 = map(lambda t: rc.libor_ois_spread(0, t, fopt, lopt), x1)
fig=plt.figure()
pt1 = fig.add_axes([1, 1, 1, 1])
pt1.plot(x1, 100*np.asarray(y1),label="Instaneous LIBOR")
pt1.plot(x1, 100*np.asarray(y2),label="Instaneous OIS Rate")
pt1.plot(x1, 100*np.asarray(y3),label="LIBOR/OIS Spread")
pt1.set_xlabel("Month")
pt1.set_ylabel("Rate");
pt1.set_title("Instaneous LIBOR VS Instaneous OIS Rate")
pt1.legend(loc=4)
plt.show()
```



Plot 2: LIBOR Forword VS OIS Forword Rate

In [7]:

```
x2 = np.linspace(0, 30, 100)
y4 = map(lambda t: rc.libor_fwd_rate(0,t, lopt), x2)
y5 = map(lambda t: rc.ois_fwd_rate(0,t, fopt), x2)
fig=plt.figure()
pt1 = fig.add_axes([1, 1, 1, 1])
pt1.plot(x2, 100*np.asarray(y4),label="LIBOR Forword Rate")
pt1.plot(x2, 100*np.asarray(y5),label="OIS Forword Rate")
pt1.set_xlabel("Month")
pt1.set_ylabel("Rate");
pt1.set_title("LIBOR Forword VS OIS Forword Rate")
pt1.legend(loc=4)
plt.show()
```

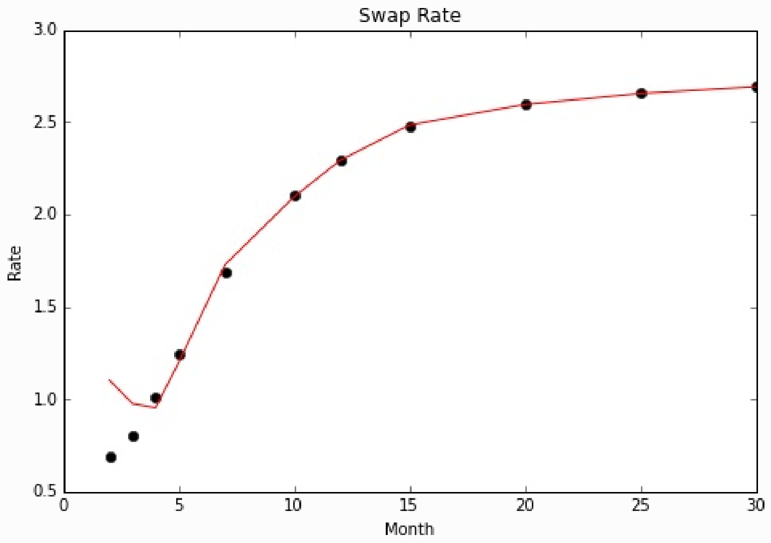


Calibration Lookback:

Plot 3: Swap Rate

In [8]:

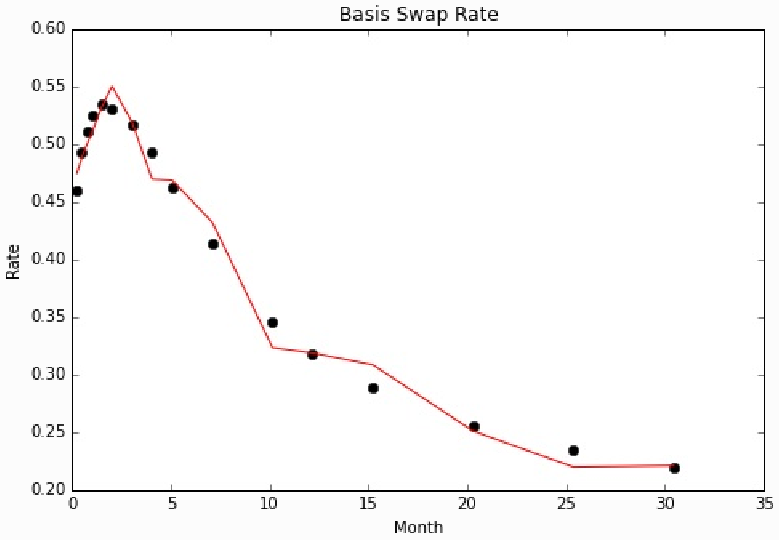
```
fig=plt.figure()
pt2 = fig.add_axes([1, 1, 1, 1])
pt2.plot(swapRateObj._tenor_, 100*swapRateObj._rvec_, 'ro',color="black")
pt2.plot(swapRateObj._tenor_, 100*swapRateObj.calibrate(fopt, lopt), 'y-',color="red")
pt2.set_xlabel("Month")
pt2.set_ylabel("Rate");
pt2.set_title("Swap Rate")
plt.show()
```



Plot 5: Basis Swap Rate

In [9]:

```
fig=plt.figure()
pt3 = fig.add_axes([1, 1, 1, 1])
pt3.plot(bSwapRateObj._tenor_, 100*bSwapRateObj._rvec_, 'ro',color="black")
pt3.plot(bSwapRateObj._tenor_, 100*bSwapRateObj.calibrate(fopt, lopt), 'y-',color="red")
pt3.set_xlabel("Month")
pt3.set_ylabel("Rate");
pt3.set_title("Basis Swap Rate")
plt.show()
```



(6) PV of Swap

The function for PV of a common Swap with fixed leg and float Libor leg is implemented in B_Spline.py, class Rate_Functions, function PV_Swap(self, tenor, fixed_rate, fopt, lopt, N=100, T_0=0):

If we Notional=100, T_0=0, tenor = 5, fixed_rate = 0.02, then

In [10]:

```
print "PV of Swap is ",rc.PV_Swap(5,0.02,fopt,lopt)
```

```
PV of Swap is  8.46919723129
```