Interest Rate Modeling for Counterparty Risk

CQF Final Project

Kai Liu

1. Introduction

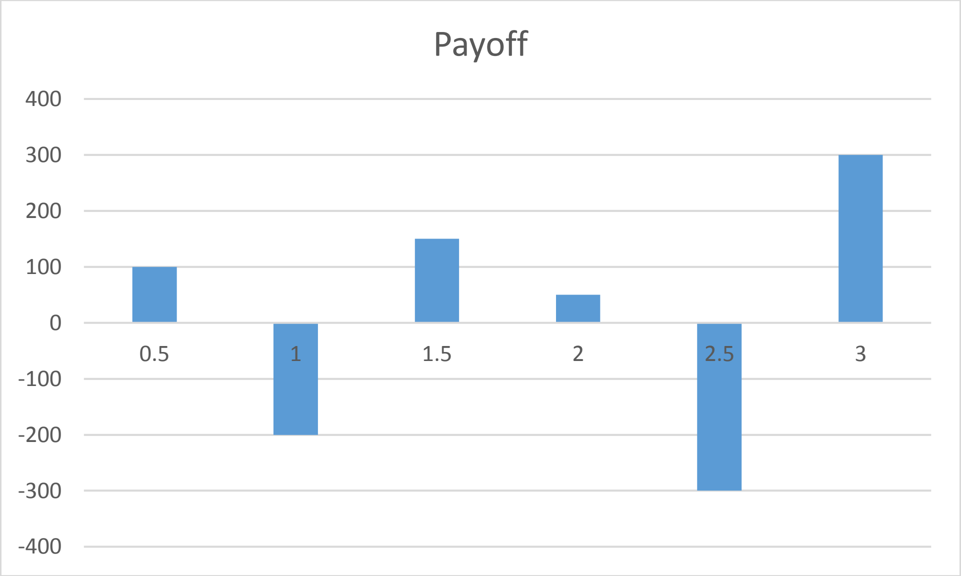
This paper is to give you an overview of the final project – Interest Rate Modeling for Counterparty Risk. I breakdown the project into five components, Heath-Jarrow-Morton Model, Principle Component Analysis, Interest Rate Swap Pricing Model, Libor-OIS Spread, and Counterparty Credit Valuation Adjustment(CVA). Each topic covers a brief of the term and the detail of the implementation in Python code.

2. Interest Rate Swap Pricing Model

Interest rate swap is a series of exchanges of a fixed interest rate for a floating rate over the contract time period. It has two legs, 'fixed' and 'floating'. The present value of an IRS is the difference of two legs. For each leg, we have to predict the future cash flow and discount the cash flows with an appropriate interest rate curve. The detail of discount factor will be disclosed in the Libor-OIS spread part. The payments of floating leg have no certainty until 6 months before the payment date in this project, while the future cash flows for fixed leg are known even before entering the contract. For each payment date, the present value of payoff happening on time t is

$$\text{PV of Payoff }(0, t) = (F_t - C) * \text{discount\_factor}(0, t)$$

And the chart below shows a scenario of future payoffs.

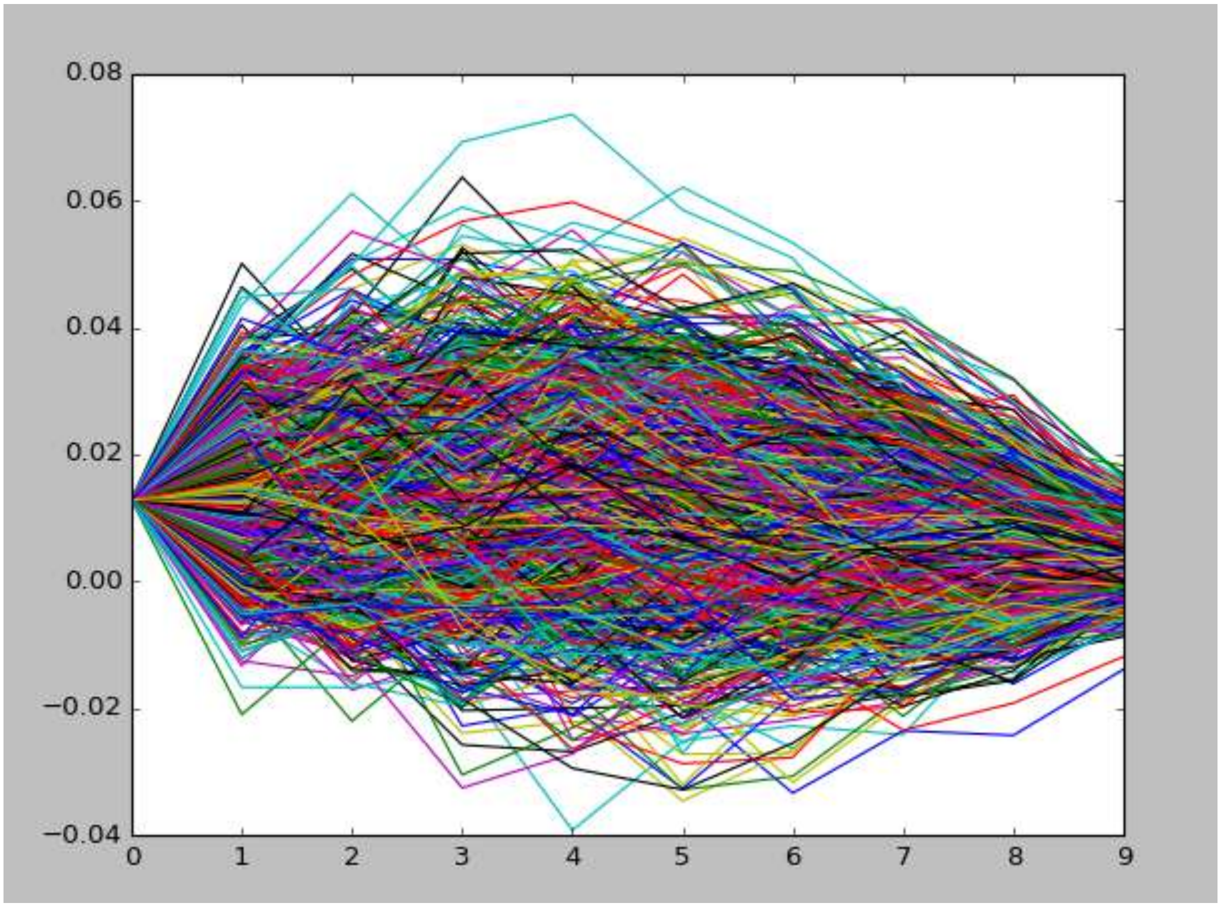


In IRS_Pricer.py, I created a function to calculate the payoff based on the role, payer or receiver, in the contract. The payoffs can be discounted by using the simulated Libor rate directly or a separate rate which can represent the funding cost. And the output of pricing functions, swap_payoffs() and swap_payoffsOIS(), include three contents, payoffs, Mark-

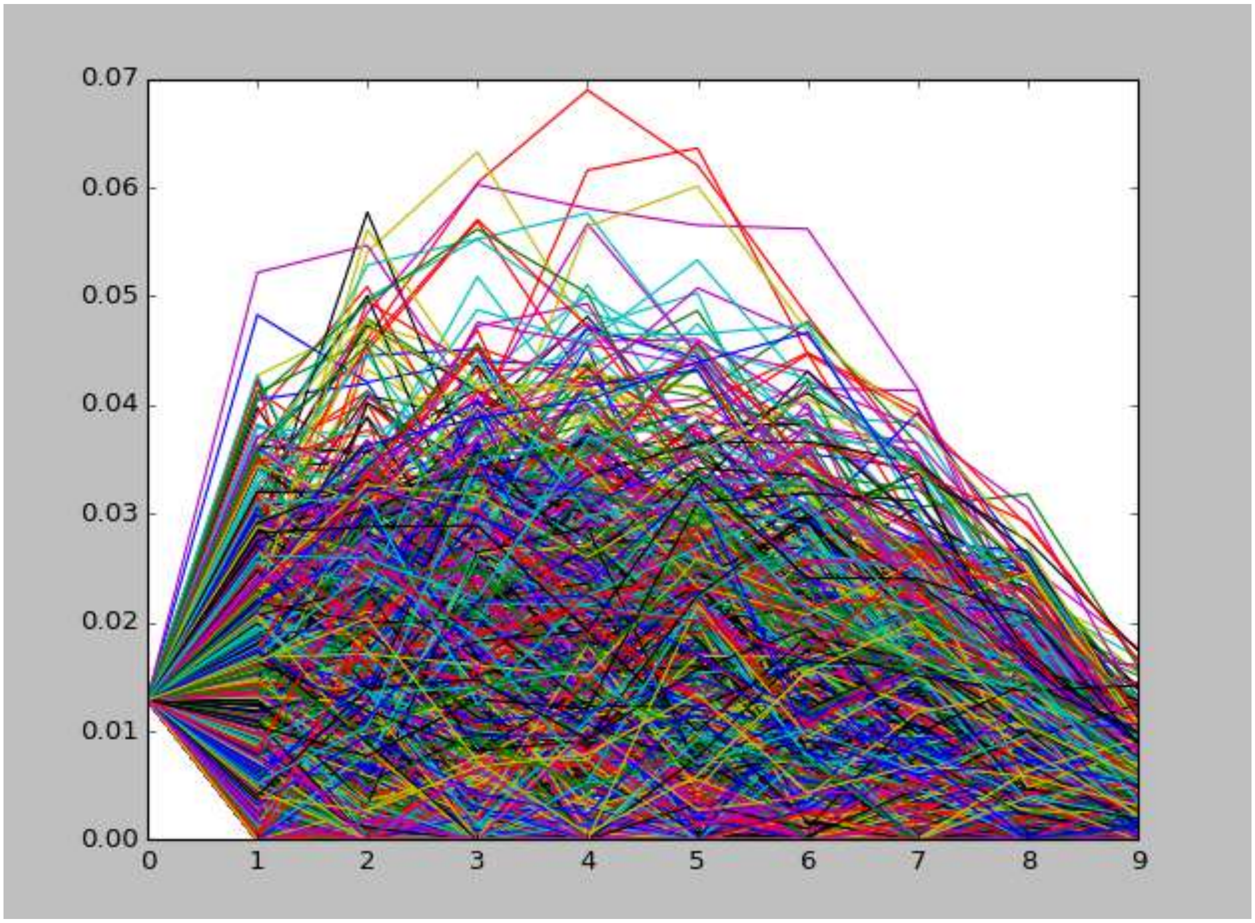
To-Markets over time, and expected exposure. I use 1% as the fixed leg rate so that most of the time we can see the positive expected exposure in the data.

Since it involves a sequence of cash flows happening in the future, the Mark-To-Market over the time period is a hard task to perform if we can only reply on a spot rate model. To capture the full curve, either Heath-Jarrow-Morton Model or Libor Market Model fits the purpose better. In this project, I chosen HJM model since it a very good framework to simulate the evolving of full forward rate curve and can incorporate the term structure of interest rates. And I got a chance to explore the implementation of PCA which used to find the factors of HJM Model.

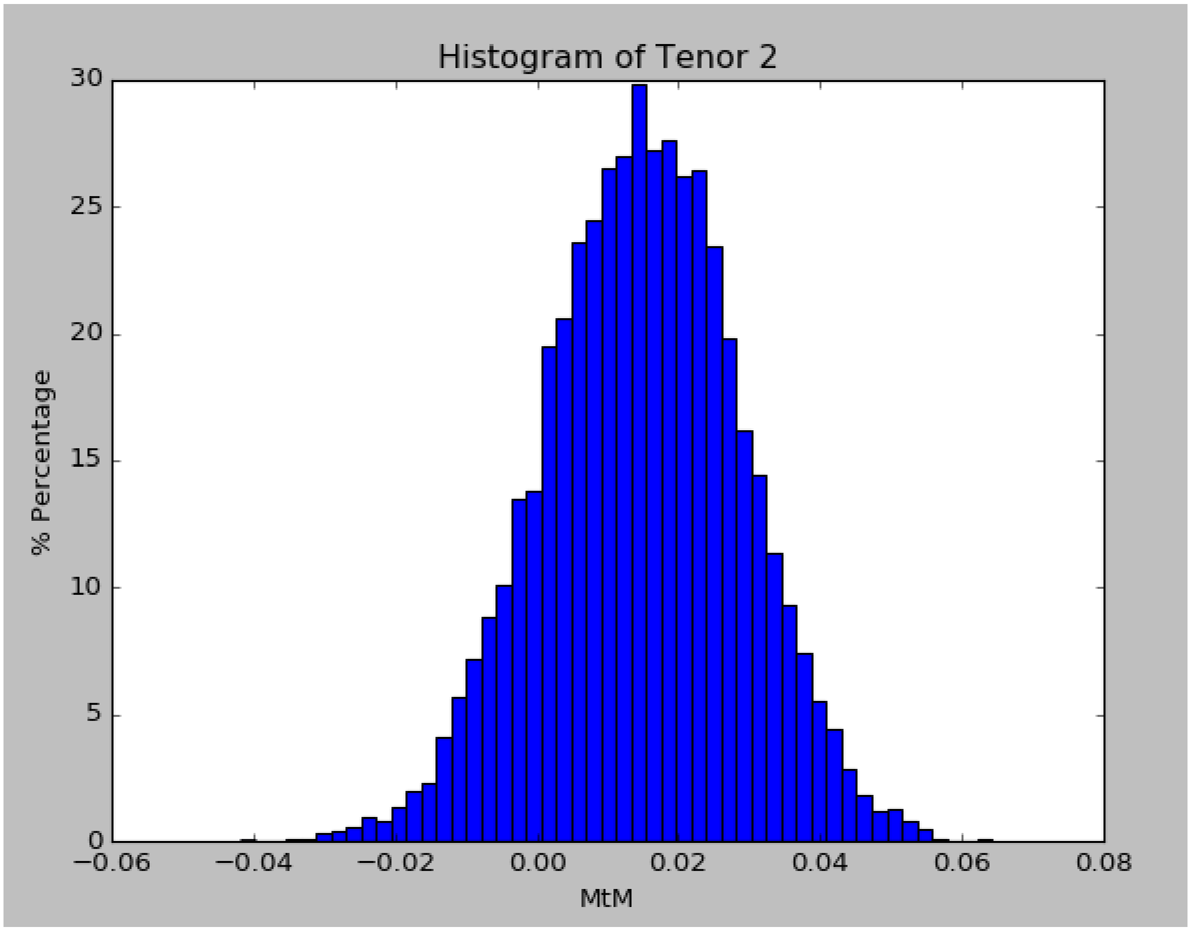
Mark to Markets at each future payment point



Expected Exposure at each future payment point



The histogram of MtM at 1Y tenor



3. Heath-Jarrow-Morton Model SDE

Under the Heath-Jarrow-Morton framework, we can plug any spot rate models, like Ho-Lee model, Hull-While model, and Vasicek model, to make a simple one-factor model presentation. But in this project, I want to use multi-factor approach so the yield curve can show a richness term structure.

As a result, the dynamics of forward rate is:

$$df(t,T) = (\sum_{i=1}^{n} \theta(t,T) \int_{t}^{T} \theta(t,s)\, ds\ )dt + \sum_{i=1}^{n} \theta(t,T)\, dXi$$

The drift term of the SDE depends on the history of the stochastic increments. It is a non-Markov process.

Mathematical formulation:

The relation between zero coupon bond prices and the instantaneous forward rate is

$$f(t,T) = \frac{dLogZ(t,T)}{dT}$$

And we assume the bond price follow a log normal random walk:

$$dZ(t,T) = \ \mu Z(t,T)dt + \ \sigma Z(t,T)dX$$

Let Y = LogZ(t, T), then PDE of instantaneous forward rate becomes

$$df(t,T) = \ \frac{\partial}{\partial T}(dY)$$

Since Y is a function of Z(t, T) then based on Ito's lemma, the differential equation becomes

$$df(t,T) = \ \frac{\partial}{\partial T}\ (\frac{\partial Y}{\partial Z}dZ + \frac{1}{2}\frac{\partial^2 Y}{\partial Z^2}dZ^2\ )$$

After plug in $\frac{\partial Y}{\partial Z} = \frac{1}{Z}$ & $\frac{\partial^2 Y}{\partial Z^2} = \ -\frac{1}{Z^2}$

$$df(t,T) = \ -\frac{\partial}{\partial T}(\left(\mu - \frac{1}{2}\sigma^2\right)\ dt + \ \sigma dX)$$

Let $v = -\frac{\partial}{\partial T}\sigma$

Under the pricing measure Q, u should equal to risk free rate which is irrelevant to time T, then the mean part of the equation turns to

$$\frac{\partial}{\partial T}\left(\frac{1}{2}\sigma^2\right) = \frac{1}{2}(\sigma * v + v * \sigma) = \sigma * v$$

After plug this into the differential equation of forward rate before, we get the initial SDE.

Since the points of yield curve are fixed tenors, (t, T-t), rather than maturity dates, T, we have to introduce Musiela Parametrisation of the HJM Model so the model can be implemented more convenient. Then the dynamics of forward rate in terms of fixed tenors can be obtained by applying the chain rule of differentiation,

$$d\bar{f}(t,\tau) = (\sum_{i=1}^{n} \theta(t,\tau) \int_t^T \theta(t,s)\, ds \; + \; \frac{\partial \overline{f(t,\tau)}}{\partial \tau})dt + \sum_{i=1}^{n} \theta(t,\tau)\, dXi$$

In HJM_Calibrator.py, I implemented this formula in the function, fcurve_HJM_MusielaParams(). For the drift term, it has an integration involved. So I made two numerical integration functions, drift_HJM_Generator_TR() and drift_HJM_Generator_SR(). The function, drift_HJM_Generator_TR(), is to approximate the integration by using Trapezoidal rule. The approximation formula is

$$I(TR) = \frac{h}{2}\,(f(a0) + 2\sum_{i=1}^{n-1} f(ai) + f(an))$$

And drift_HJM_Generator_SR() adopt Simpson’s rule to implement this approximation. And the expression is

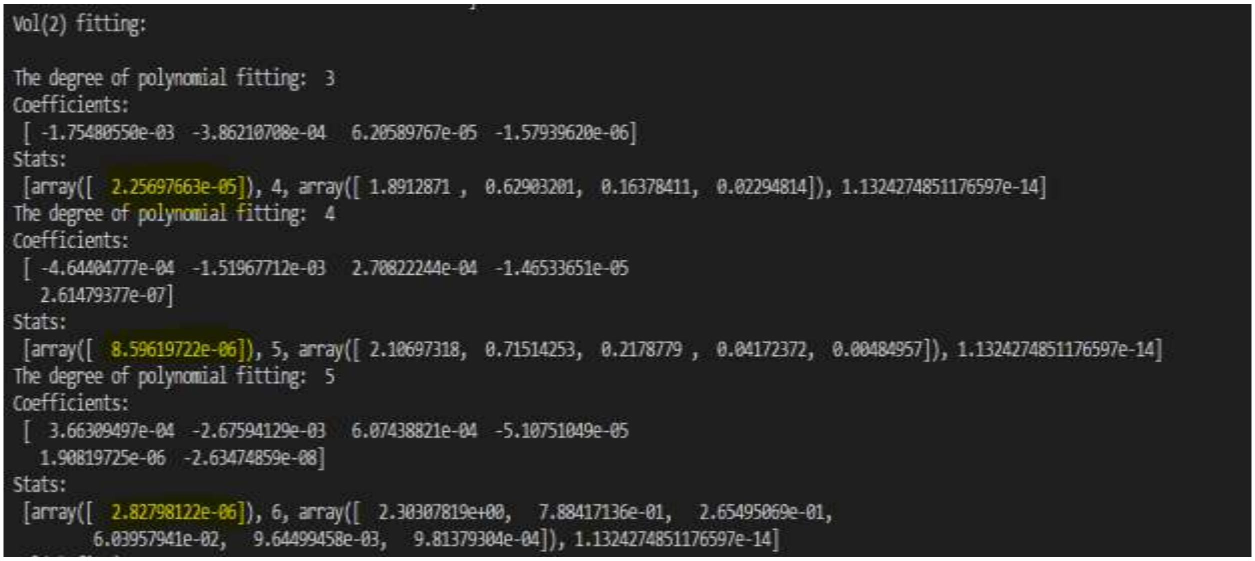
$$I(SR) = \frac{h}{6}\left(f(a(0)) + 2\sum_{i=1}^{n-1} f(a(i)) + f(a(n)) + 4\sum_{i=1}^{n} f(xi)\right)$$

Where x_i = (a(i-1) + a(i))/2

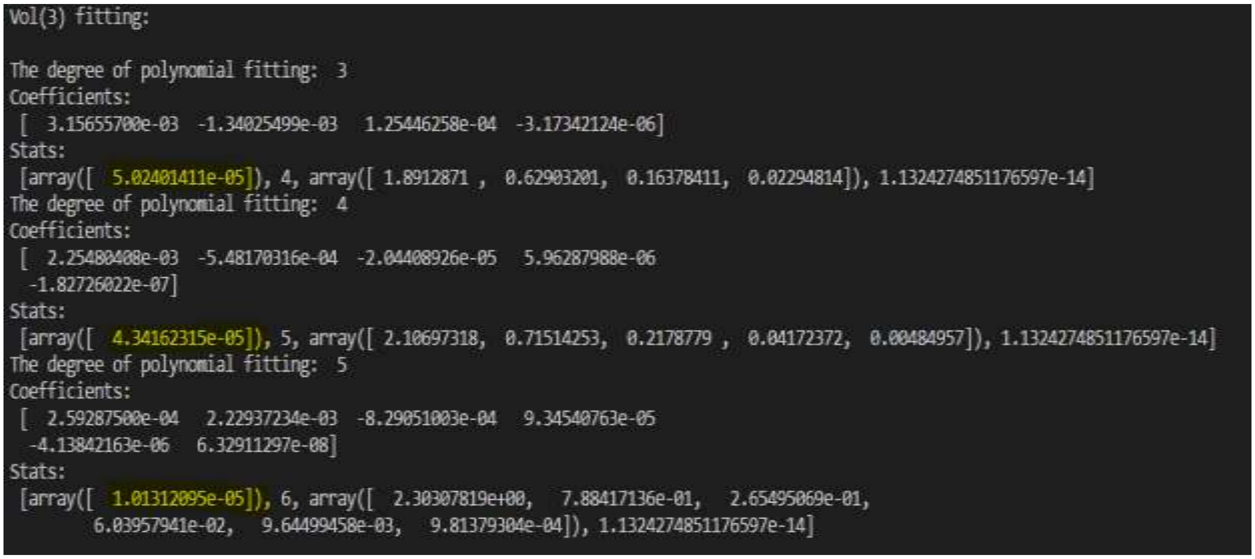
And vol_nHJM_Solver() and vol_Generator() functions are for Musiela parameterization HJM SDE volatility functions. The function takes the eigenvalues and the eigenvectors picked by PCA and make the volatility function by using a constant value for volatility function one and fitting a cubic spline for volatility functions two and three.

The function, get_eigenValues() from the module HJM_Calibrator.py, sorts the eigenvalues extracted by PCA solver in a descending order and return the top three factors and the corresponding eigenvectors. I tried to export the fourth and fifth factor, but the statistic measure, R-squared, didn't improve much. And it also increase the time to do Monte Carlo simulation as more random terms are involved for each run.

Please see the statistics of polynomial fitting with degree 3, 4, 5 for both Vol2 function and Vol3 function. A same observation from the tests of Vol2 and Vol3 is that the sum of squared residuals of the least squares fits performed very well with just three coefficients.



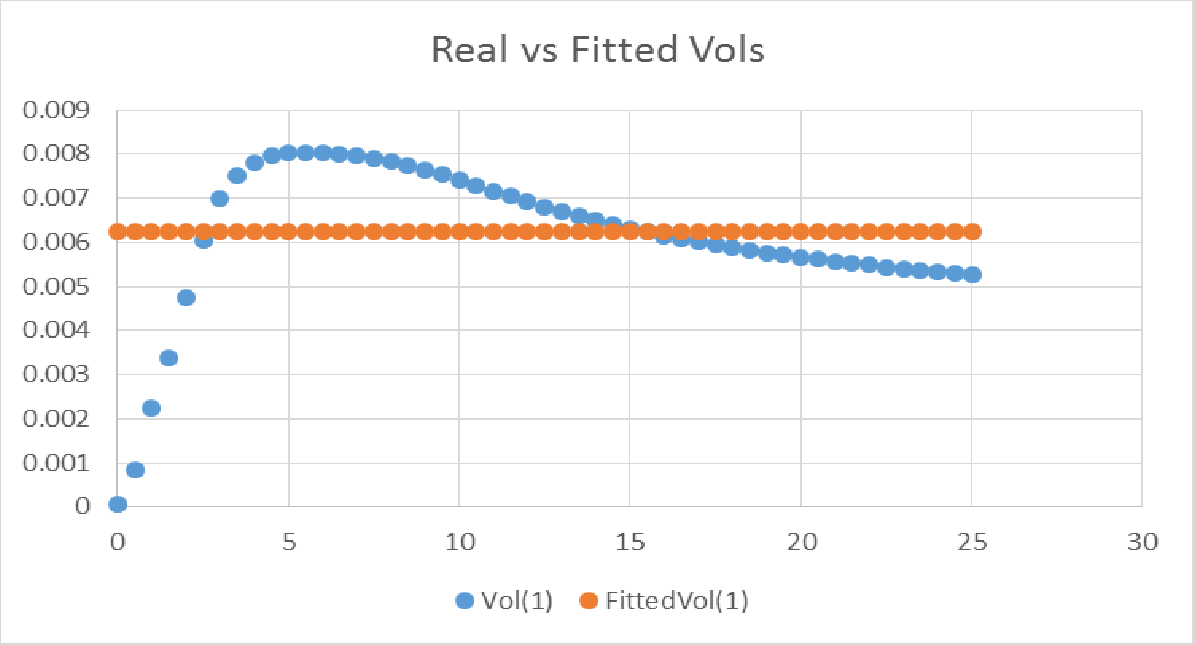
```
Vol(2) fitting:

The degree of polynomial fitting:  3
Coefficients:
 [ -1.75480550e-03  -3.86210708e-04   6.20589767e-05  -1.57939620e-06]
Stats:
 [array([  2.25697663e-05]), 4, array([ 1.8912871 ,  0.62903201,  0.16378411,  0.02294814]), 1.1324274851176597e-14]
The degree of polynomial fitting:  4
Coefficients:
 [ -4.64404777e-04  -1.51967712e-03   2.70822244e-04  -1.46533651e-05
   2.61479377e-07]
Stats:
 [array([  8.59619722e-06]), 5, array([ 2.10697318,  0.71514253,  0.2178779 ,  0.04172372,  0.00484957]), 1.1324274851176597e-14]
The degree of polynomial fitting:  5
Coefficients:
 [  3.66309497e-04  -2.67594129e-03   6.07438821e-04  -5.10751049e-05
   1.90819725e-06  -2.63474859e-08]
Stats:
 [array([  2.82798122e-06]), 6, array([  2.30307819e+00,   7.88417136e-01,   2.65495069e-01,
         6.03957941e-02,   9.64499458e-03,   9.81379304e-04]), 1.1324274851176597e-14]
```

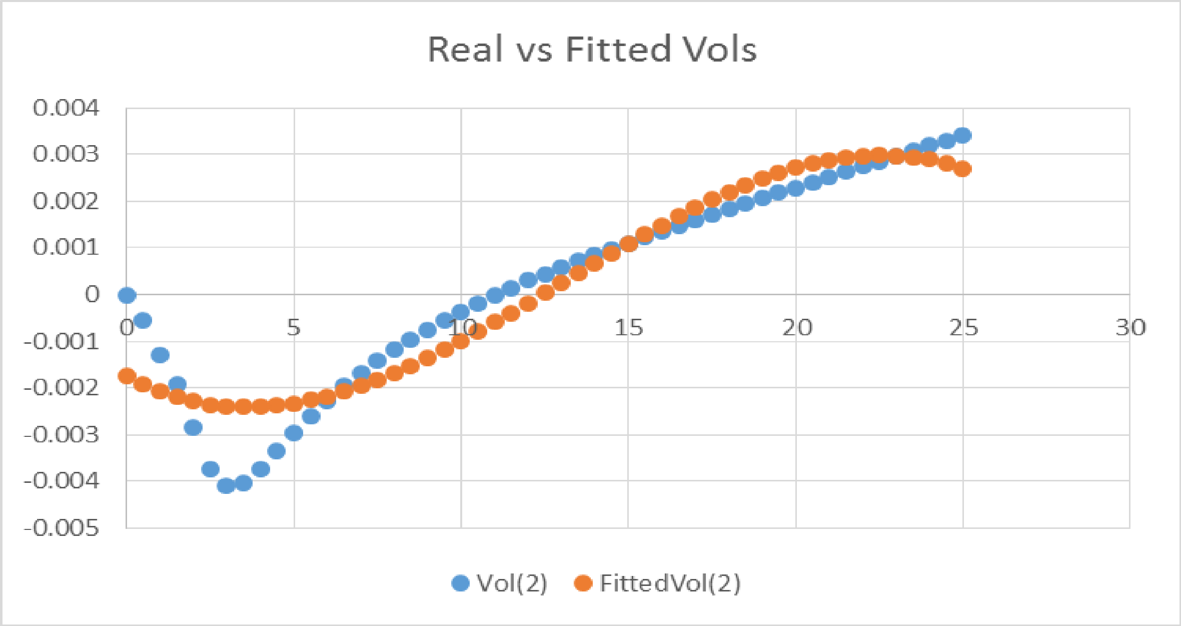


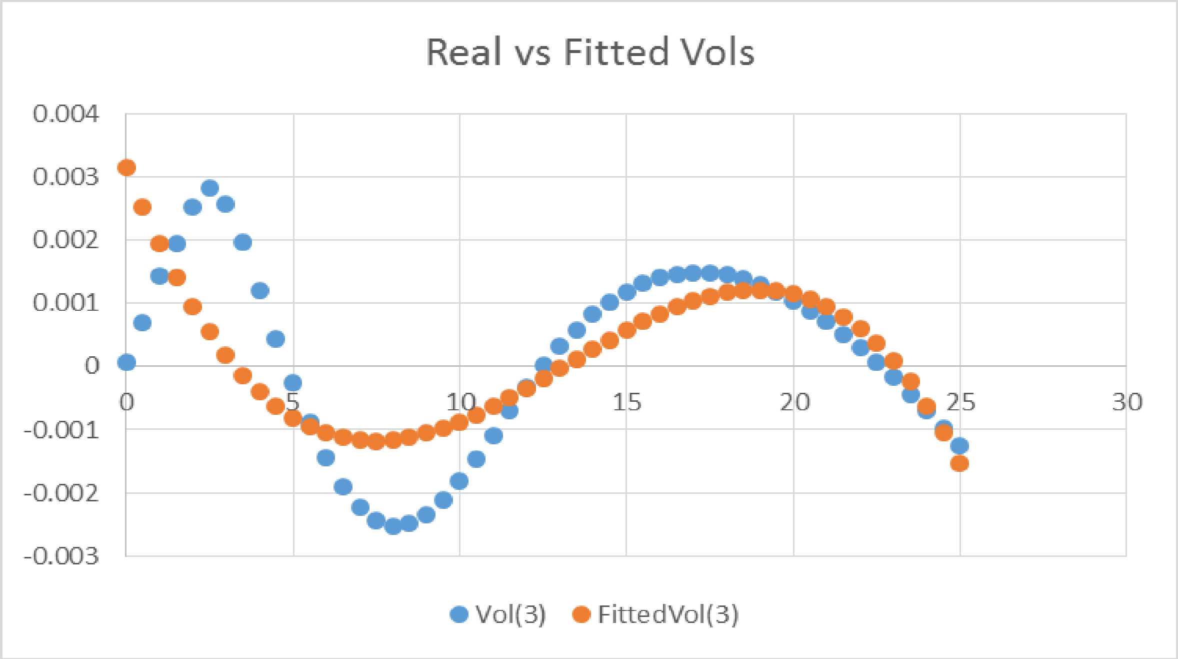
```
Vol(3) fitting:

The degree of polynomial fitting:  3
Coefficients:
 [  3.15655700e-03  -1.34025499e-03   1.25446258e-04  -3.17342124e-06]
Stats:
 [array([  5.02401411e-05]), 4, array([ 1.8912871 ,  0.62903201,  0.16378411,  0.02294814]), 1.1324274851176597e-14]
The degree of polynomial fitting:  4
Coefficients:
 [  2.25480408e-03  -5.48170316e-04  -2.04408926e-05   5.96287988e-06
  -1.82726022e-07]
Stats:
 [array([  4.34162315e-05]), 5, array([ 2.10697318,  0.71514253,  0.2178779 ,  0.04172372,  0.00484957]), 1.1324274851176597e-14]
The degree of polynomial fitting:  5
Coefficients:
 [  2.59287500e-04   2.22937234e-03  -8.29051003e-04   9.34540763e-05
  -4.13842163e-06   6.32911297e-08]
Stats:
 [array([  1.01312095e-05]), 6, array([  2.30307819e+00,   7.88417136e-01,   2.65495069e-01,
         6.03957941e-02,   9.64499458e-03,   9.81379304e-04]), 1.1324274851176597e-14]
```

After extracting principal components, the next step is to get Musiela parameterization HJM SDE volatility functions.

Charts of three eigenvectors vs fitted curve. Comparison of Real Vols and Fitted Vols for Vol1, Vol2, Vol3

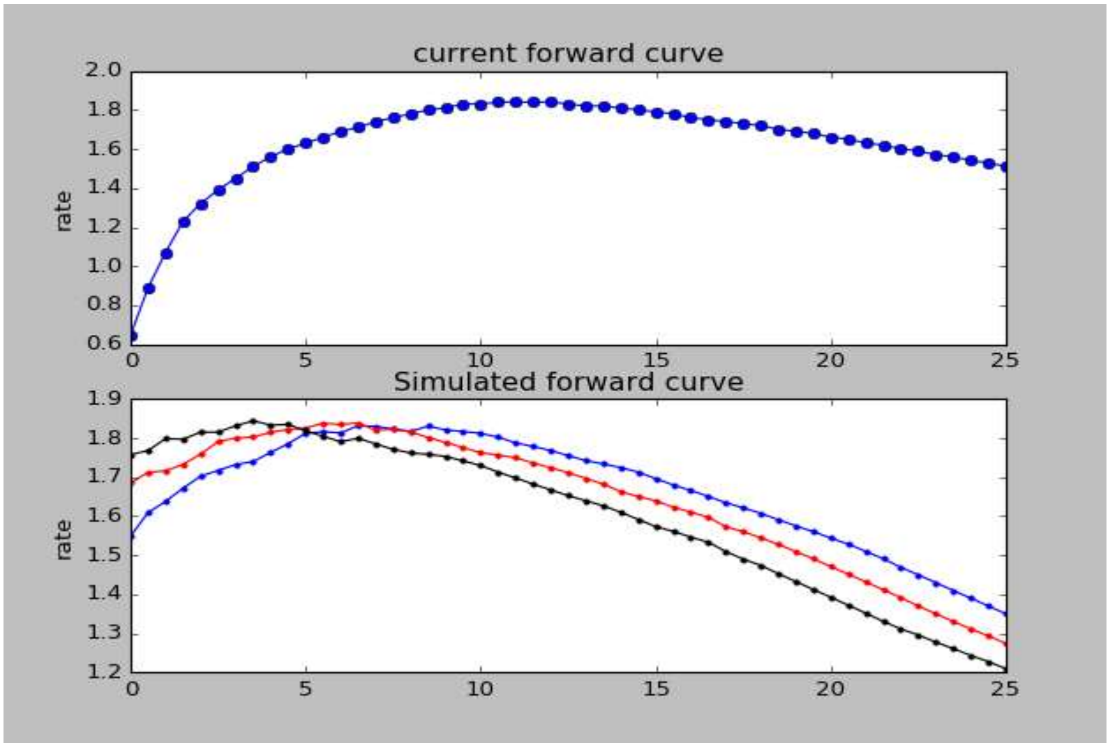




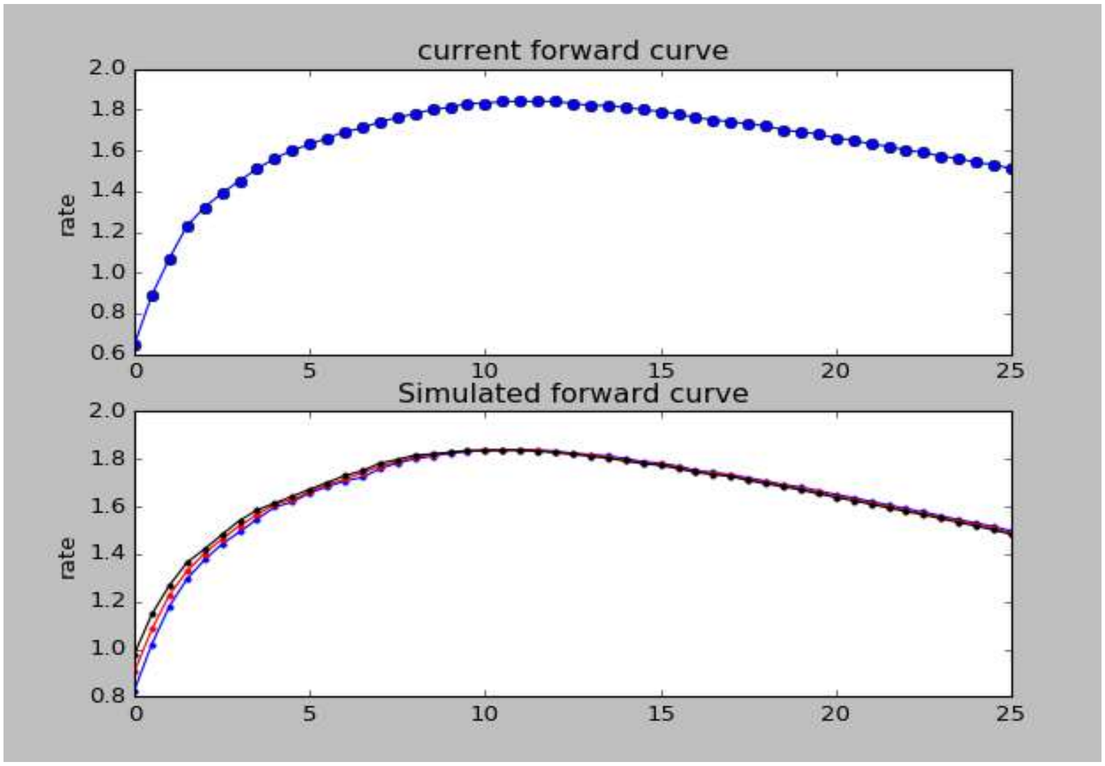


After getting the drift functions and volatility functions of all tenors, I have to find the initial stage to start simulating the evolving of forward rate curve. The initial forward rate curve I picked up is the latest one, as of July 12th, of UK instantaneous commercial bank liability forward curve available on July 13th. The graphs are with different time step settings. Each plot charts below include the current forward rate curve and several simulated forward curves at different time points.

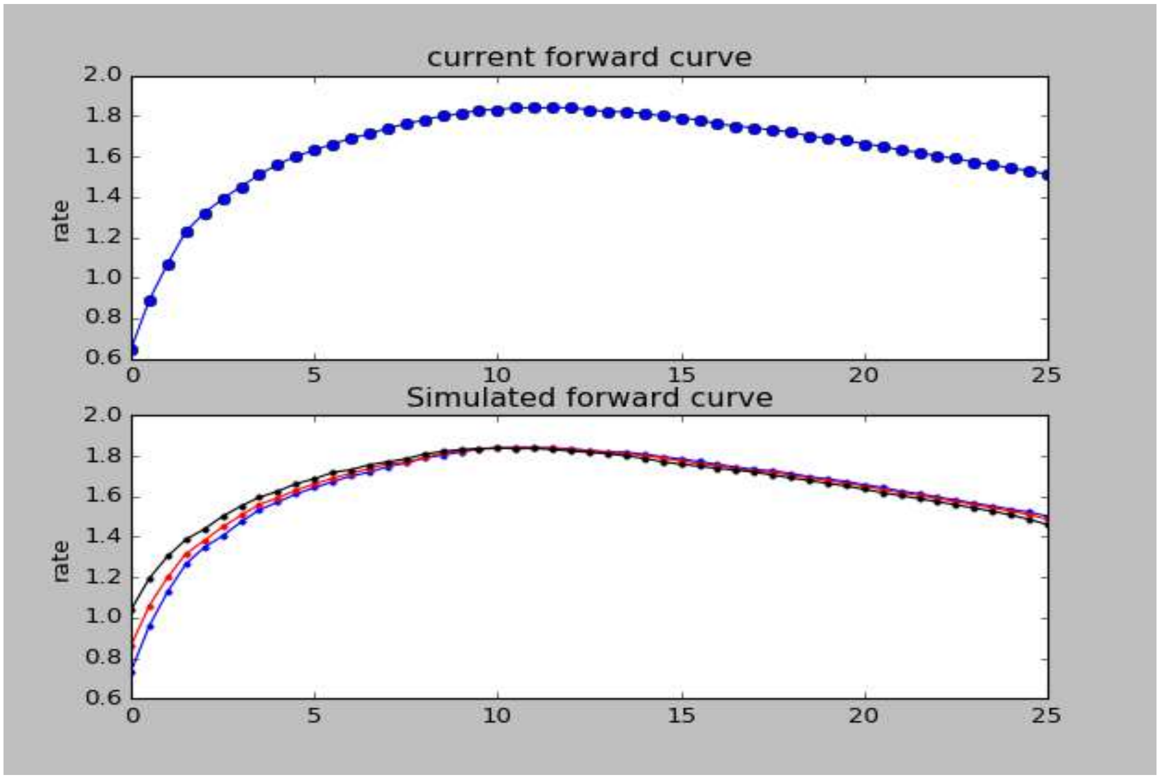
Time step: 5 increments between two tenors



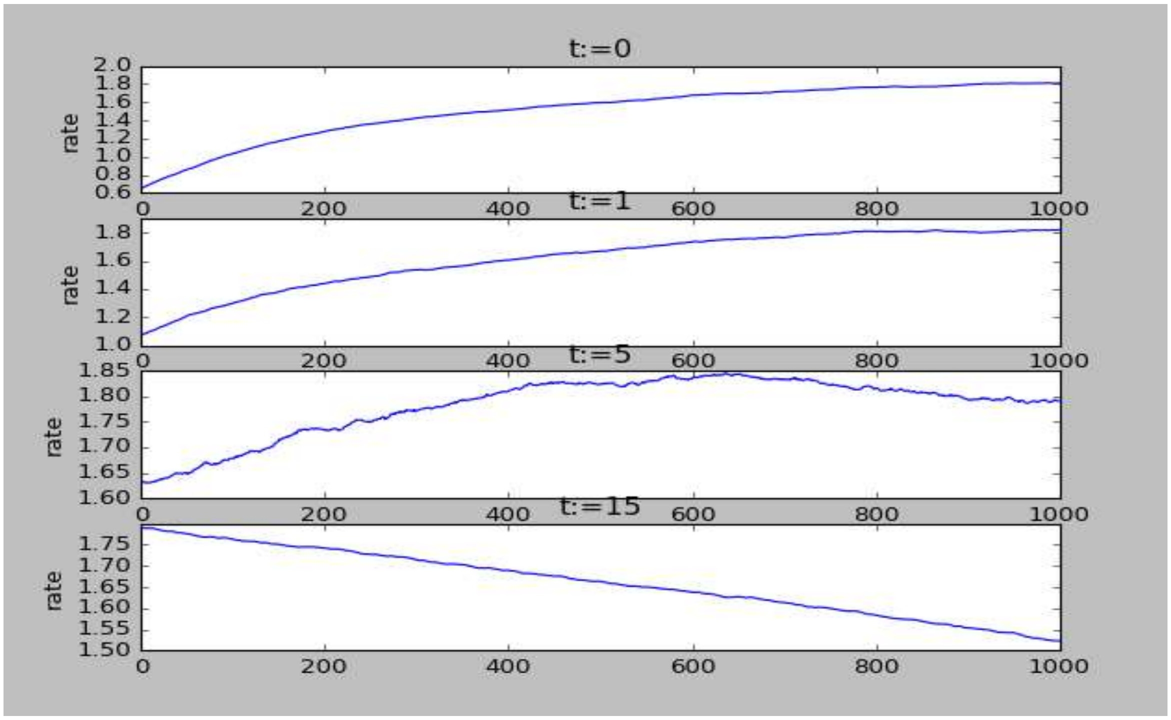
Time step: 50 increments between two tenors



Time steps: 100 increments between two tenors



Projection of forward rate (at selected tenors)



4. Principle Component Analysis

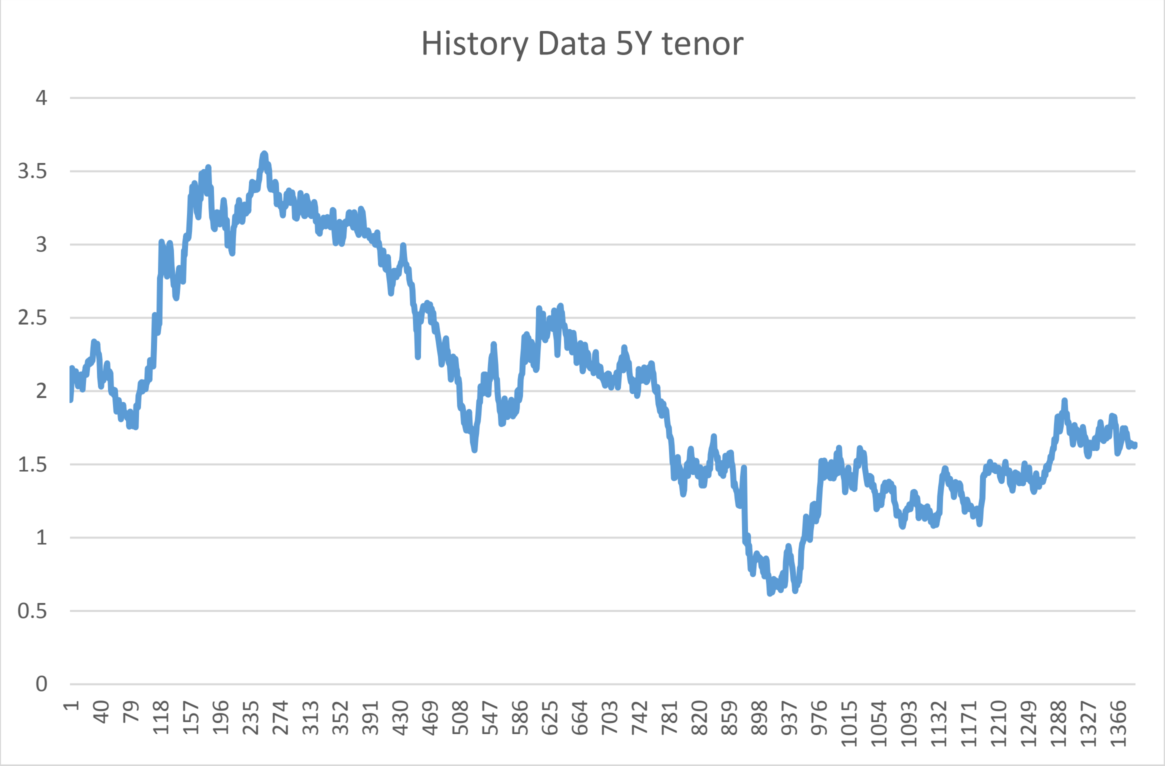
There are so many tenors on a yield curve and market can trade with enormous maturity dates. The critical part of multi-factor model is to figure out the dominant factors that drive the changes of the curve. Principal Component Analysis is a great approach to tackle down this task. Normally a few factors describe well above 90% of variance of the yield curve.

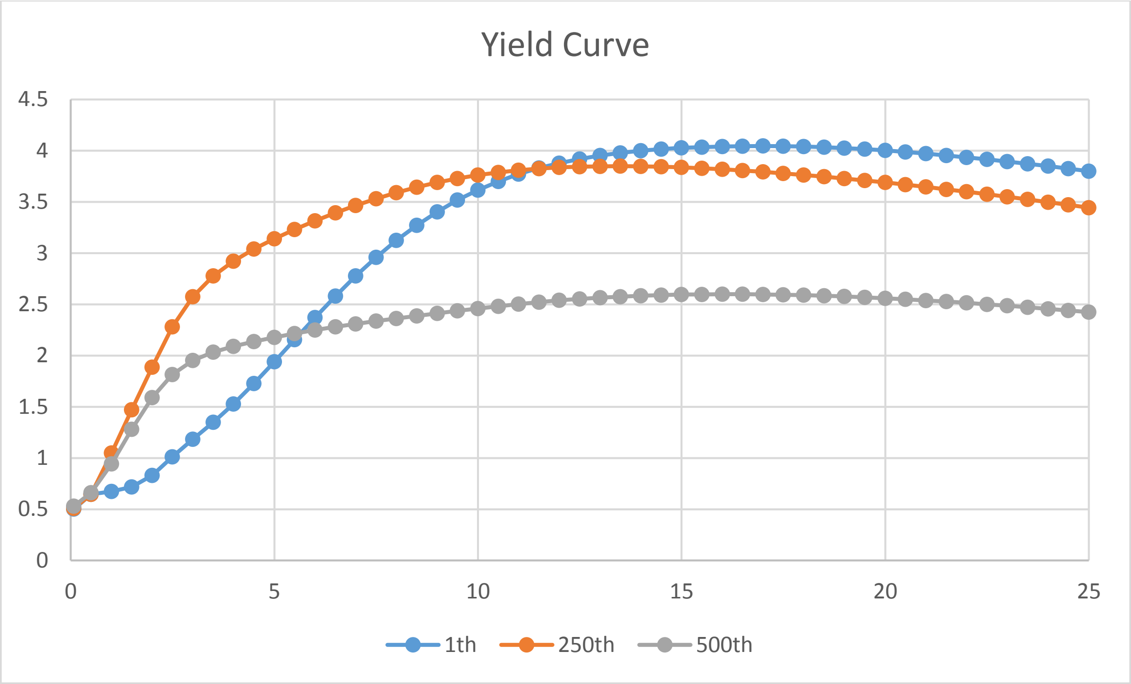
The numerical method I chose is Jacobi Transformation. The module PCA_Solver.py is the implementation. Eigenvalue is variance of the movements in each eigendirection. And sum of eigenvalues is equal to the sum of the raw data variances. Each eigenvector is a projection of curve movement in one direction. To decompose the covariance matrix to an eigenvalues matrix and an eigenvectors matrix, the solver follows the steps below:

1. Check the values of off-diagonal whether the sum is close to the tolerance
2. If not, find the largest entry of off-diagonal elements
3. Based on the position of the value returned from step 2, calculate values for orthogonal rotation matrix which can make the new matrix with value 0 on that entry
4. Multiply the matrix with the rotation matrix got from step 3 to get a new matrix
5. Repeat steps 1 to 4 until the values of off diagonal elements are below the accepted tolerance
6. Calculate eigenvectors through the history of rotations

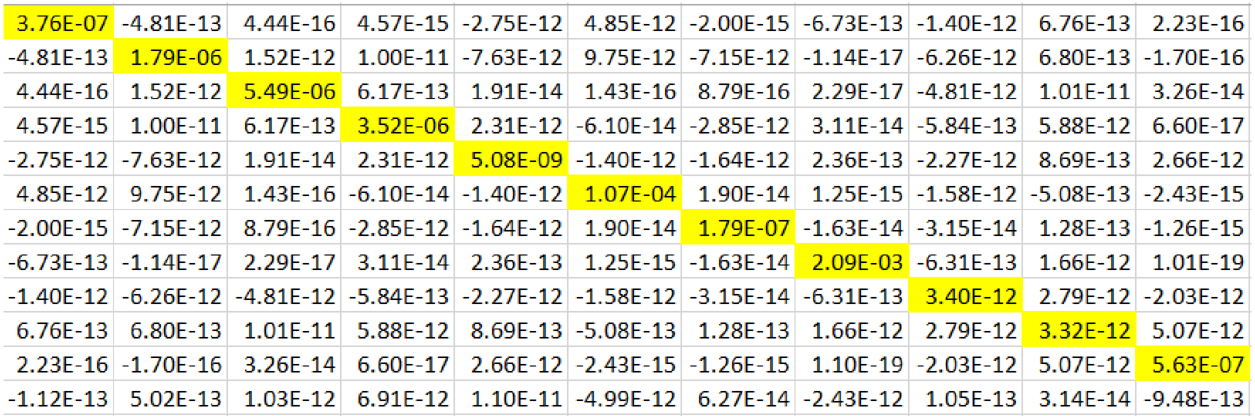
And I tested the functions by using the same dataset given in the spreadsheet, CQF_June_2018_M4L4_HJM Model – PCA, and got the same results on returns, covariance matrix, eigenvalues, and eigenvectors. I use the daily movements to calculate covariance matrix directly since the daily-scaled mean is very small. After doing the function validation, the system was used to calibrate the parameters of HJM SDE. The history data, Pound Sterling Bank Liability Curve (nominal), was downloaded from the Bank of England website and I picked the period January 1st, 2013 to June 29th, 2018, for calibrating the parameters.

This time period has low interest rate level and is in a low volatile stage. Please see the charts below for this feature. It doesn't involve a 'regime-switching' period so that I don't expect the principal components have unclear attributions.





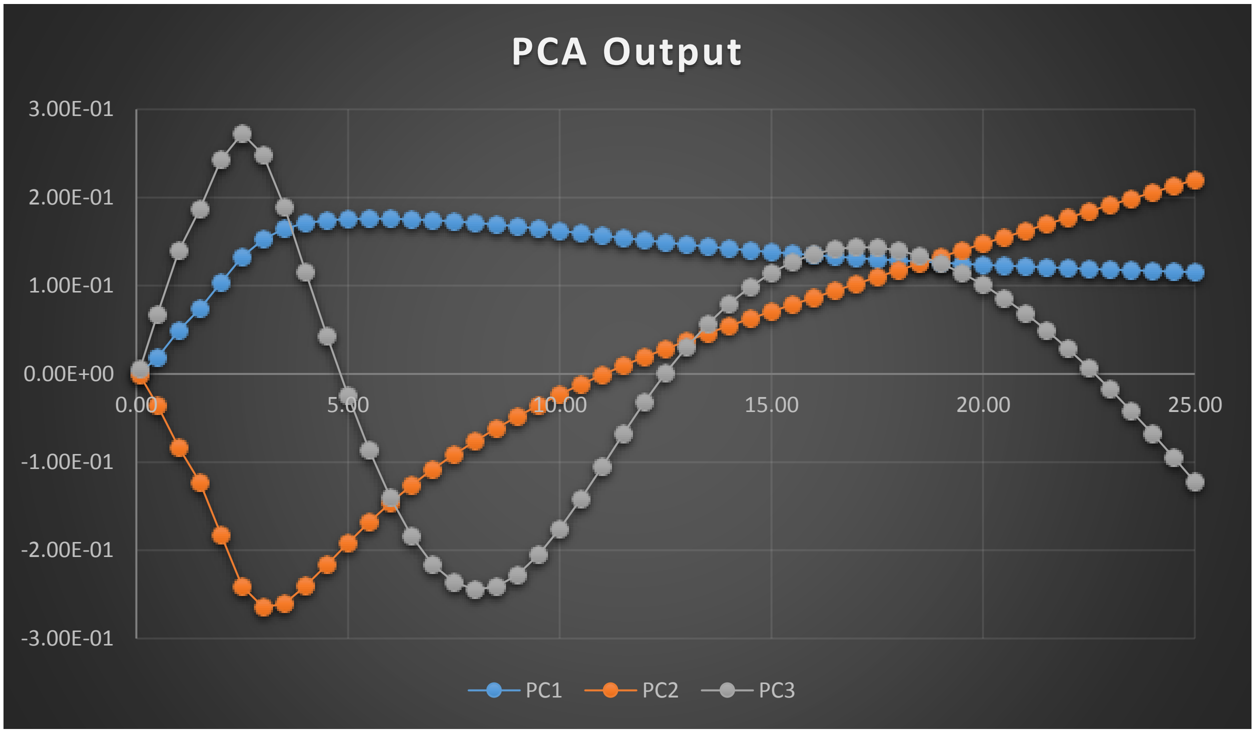
A snap shot for a part of returned eigenvalues.



3.76E-07	-4.81E-13	4.44E-16	4.57E-15	-2.75E-12	4.85E-12	-2.00E-15	-6.73E-13	-1.40E-12	6.76E-13	2.23E-16
-4.81E-13	1.79E-06	1.52E-12	1.00E-11	-7.63E-12	9.75E-12	-7.15E-12	-1.14E-17	-6.26E-12	6.80E-13	-1.70E-16
4.44E-16	1.52E-12	5.49E-06	6.17E-13	1.91E-14	1.43E-16	8.79E-16	2.29E-17	-4.81E-12	1.01E-11	3.26E-14
4.57E-15	1.00E-11	6.17E-13	3.52E-06	2.31E-12	-6.10E-14	-2.85E-12	3.11E-14	-5.84E-13	5.88E-12	6.60E-17
-2.75E-12	-7.63E-12	1.91E-14	2.31E-12	5.08E-09	-1.40E-12	-1.64E-12	2.36E-13	-2.27E-12	8.69E-13	2.66E-12
4.85E-12	9.75E-12	1.43E-16	-6.10E-14	-1.40E-12	1.07E-04	1.90E-14	1.25E-15	-1.58E-12	-5.08E-13	-2.43E-15
-2.00E-15	-7.15E-12	8.79E-16	-2.85E-12	-1.64E-12	1.90E-14	1.79E-07	-1.63E-14	-3.15E-14	1.28E-13	-1.26E-15
-6.73E-13	-1.14E-17	2.29E-17	3.11E-14	2.36E-13	1.25E-15	-1.63E-14	2.09E-03	-6.31E-13	1.66E-12	1.01E-19
-1.40E-12	-6.26E-12	-4.81E-12	-5.84E-13	-2.27E-12	-1.58E-12	-3.15E-14	-6.31E-13	3.40E-12	2.79E-12	-2.03E-12
6.76E-13	6.80E-13	1.01E-11	5.88E-12	8.69E-13	-5.08E-13	1.28E-13	1.66E-12	2.79E-12	3.32E-12	5.07E-12
2.23E-16	-1.70E-16	3.26E-14	6.60E-17	2.66E-12	-2.43E-15	-1.26E-15	1.10E-19	-2.03E-12	5.07E-12	5.63E-07
-1.12E-13	5.02E-13	1.03E-12	6.91E-12	1.10E-11	-4.99E-12	6.27E-14	-2.43E-12	1.05E-13	3.14E-14	-9.48E-13

And the top three factors picked by the principal component analysis system can explain 94.72% of variance of the curve movement. The chart below shows the eigenvectors related to the top three eigenvalues. The largest principal component of curve movement is parallel shit in overall level of rates. The second principal component of curve movement is steepening/flattening of the curve.

The third principal component of curve movement is curvature about specific maturity points.


PCA Output
3.00E-01
2.00E-01
1.00E-01
0.00E+00
-1.00E-01
-2.00E-01
-3.00E-01
0.00
5.00
10.00
15.00
20.00
25.00
PC1
PC2
PC3

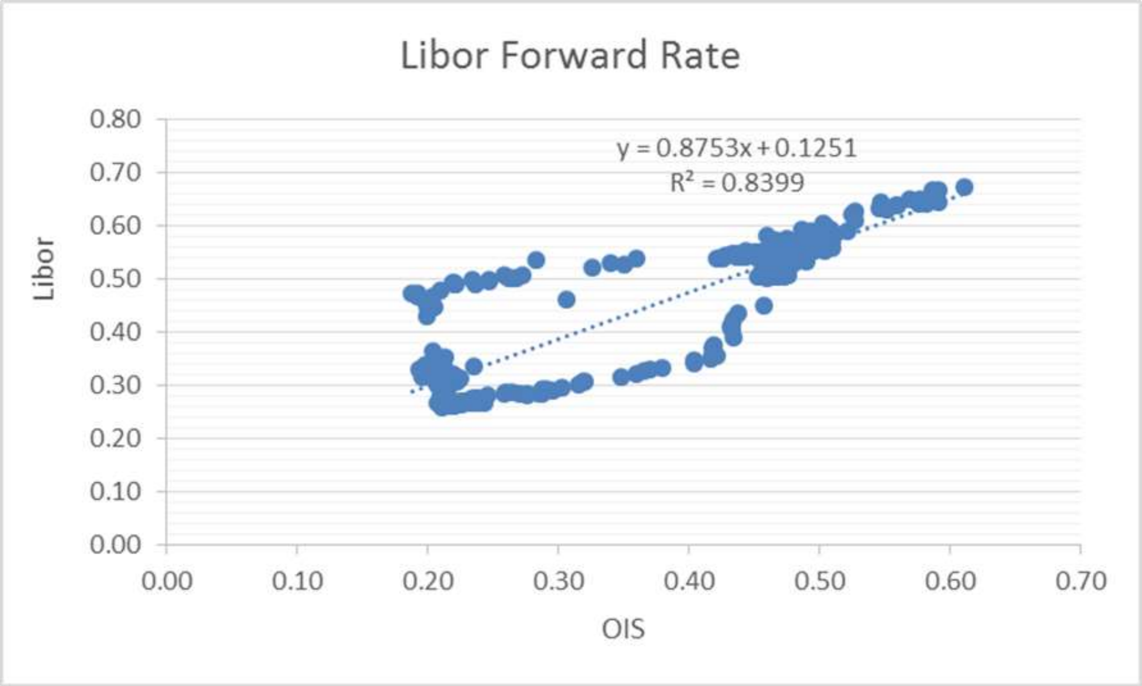
5. Libor-OIS Spread

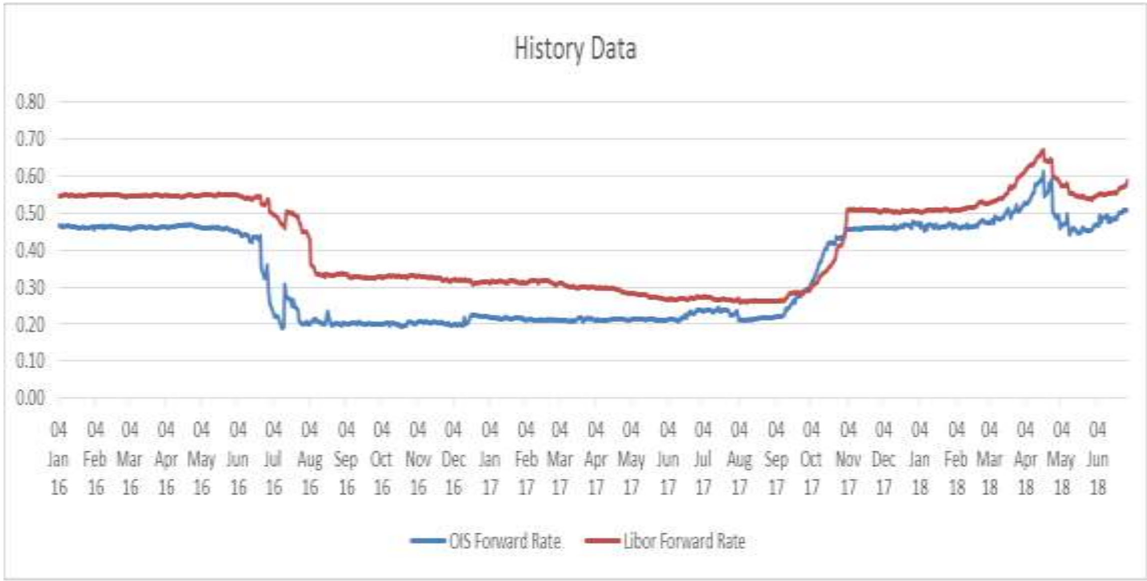
Traditionally the discount factor is derived from the Libor forward rate. But it isn't an option any more. For instance, in Credit Default Swap world, there is an ISDA yield curve for each currency. It is a Libor rate curve and used to calculate the cash settlement. But after holding a position, valuation control team in each bank will use an OIS-like curve instead of this standard curve. Thus, except simulating a Libor forward curve, finding a way to derive the discount curve is critical too.

I picked up three points, 0.08Y, 1Y, and 5Y, of Libor curve and OIS curve and compare past three years data. From the charts below, a high correlation is existed in the Libor and OIS rates cross multiple tenor points. The Libor-OIS spread is relatively stable cross the time period.

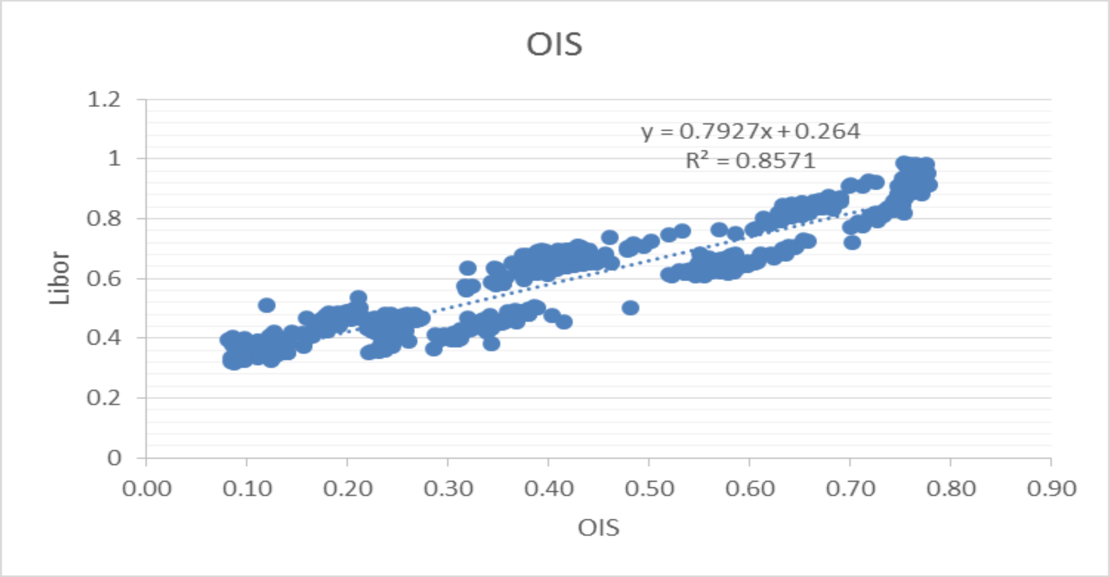
As a result, I applied a constant Libor-OIS spread to get OIS curve from the simulated Libor forward rate.

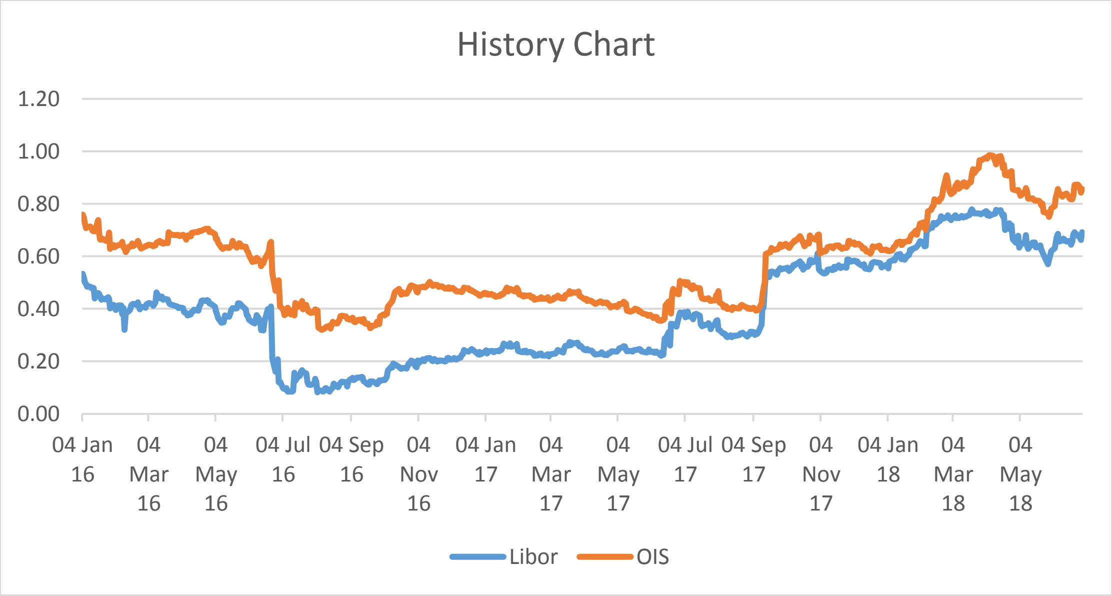
0.08Y Comparison:



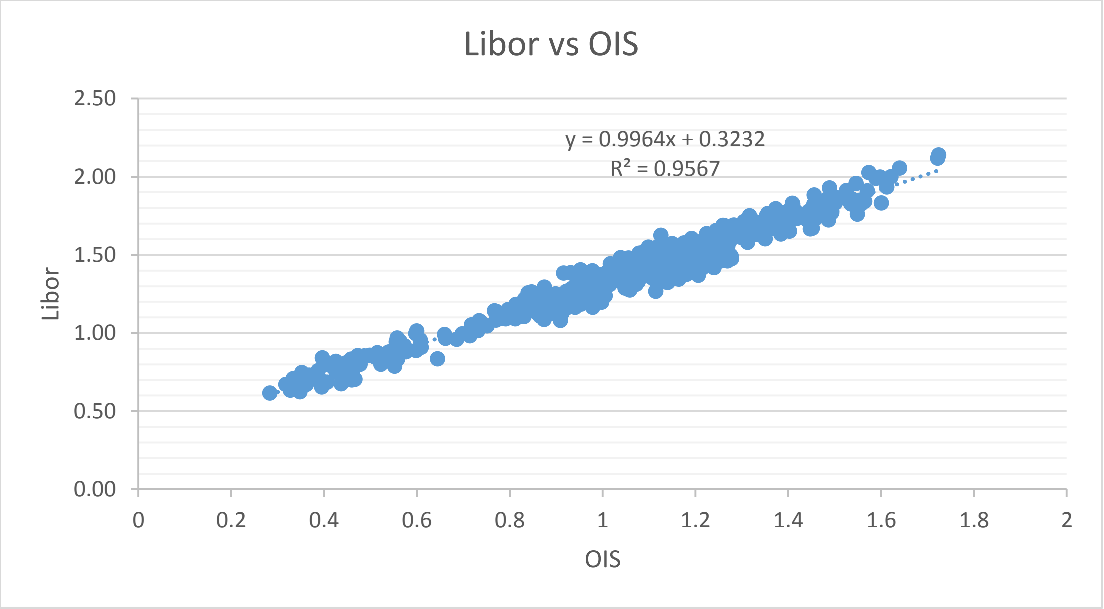

History Data
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
04 Jan 16
04 Feb 16
04 Mar 16
04 Apr 16
04 May 16
04 Jun 16
04 Jul 16
04 Aug 16
04 Sep 16
04 Oct 16
04 Nov 16
04 Dec 16
04 Jan 17
04 Feb 17
04 Mar 17
04 Apr 17
04 May 17
04 Jun 17
04 Jul 17
04 Aug 17
04 Sep 17
04 Oct 17
04 Nov 17
04 Dec 17
04 Jan 18
04 Feb 18
04 Mar 18
04 Apr 18
04 May 18
04 Jun 18
OIS Forward Rate
Libor Forward Rate

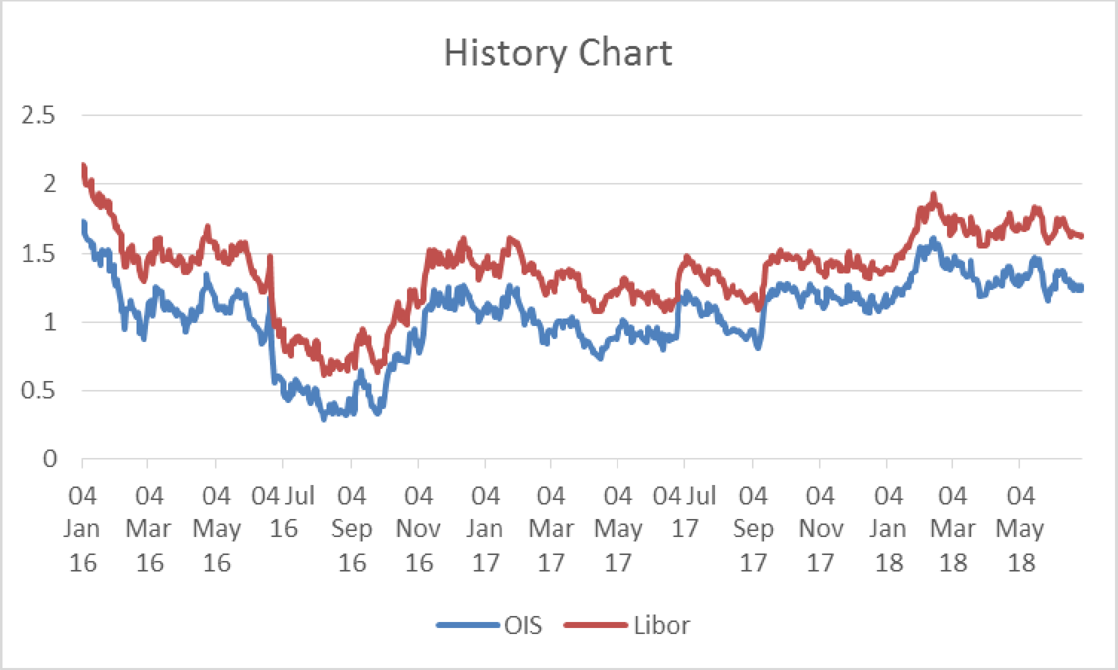
1Y Comparison:





5Y Comparison:





6. CVA

Counterparty risk valuation adjustment can be decompose to three parts, Credit, Expected Exposure, and Discount factor. Article 383 from the EU Capital Requirements Regulation (CRR) illustrates the regulatory advanced approach for the calculation of the capital requirement for the credit valuation adjustment.

$$\mathrm{CVA} = \mathrm{LGD}_{\mathrm{MKT}} \sum_{i=1}^{T} PD\left(t_{i-1,} t_i\right)\left(\frac{EE_{i-1}}{2} D_{i-1} + \frac{EE_i}{2} D_i\right)$$

Where

EE = max[Mark to Market, 0]

D_i the default risk free discount factor and for discounting at time i

$PD\left(t_{i-1,} t_i\right)$ = Exp(-$S_{i-1}T_{i-1}/\mathrm{LGD}_{\mathrm{MKT}}$) – Exp(-$S_i\ T_i/\mathrm{LGD}_{\mathrm{MKT}}$)

= Survival Probability$_{i-1}$– Survival Probability$_i$ =dPD$_i$.

The formula above is for default probability. The floor value is 0. The difference from the survival probability at the beginning of the period to the survival probability at the end of the period. Si/LGDMKT is an approximation for the hazard rate.

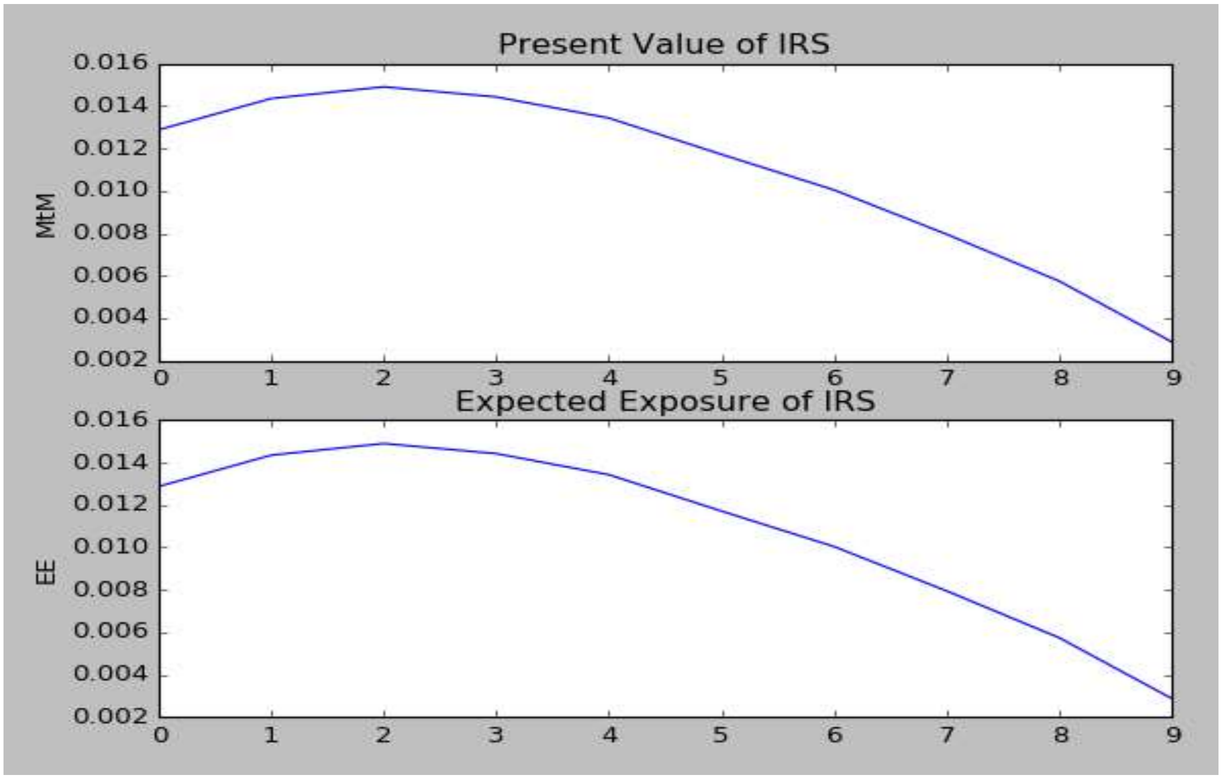
$\mathrm{LGD}_{\mathrm{MKT}}$ is the loss percentage given default. $\mathrm{LGD}_{\mathrm{MKT}}$ = 1 – Recovery rate

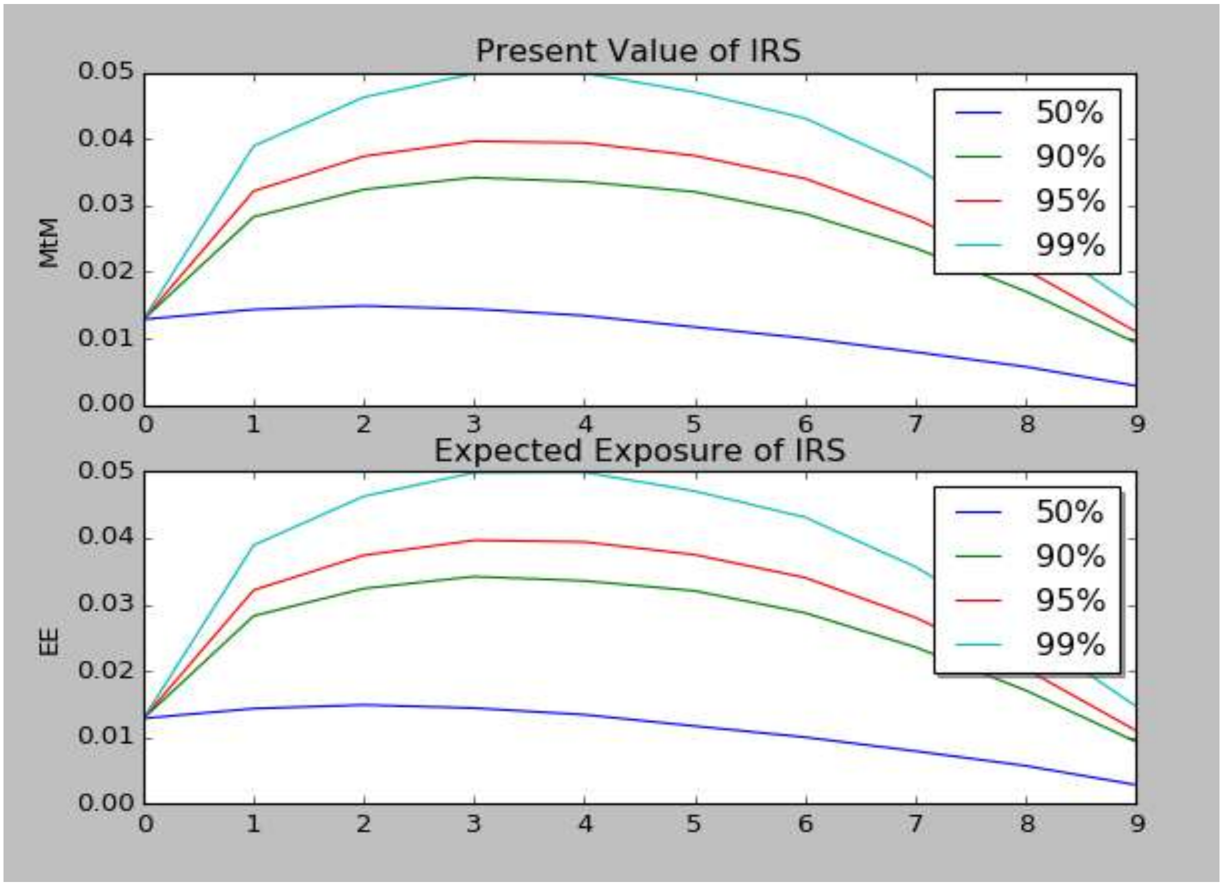
In my implementation, I used the simulated Expected Exposure values saved at the local folder with other parameters to calculate CVA. Please see the charts for the levels of MtM and Expected Exposure.

And I have several observations on it:

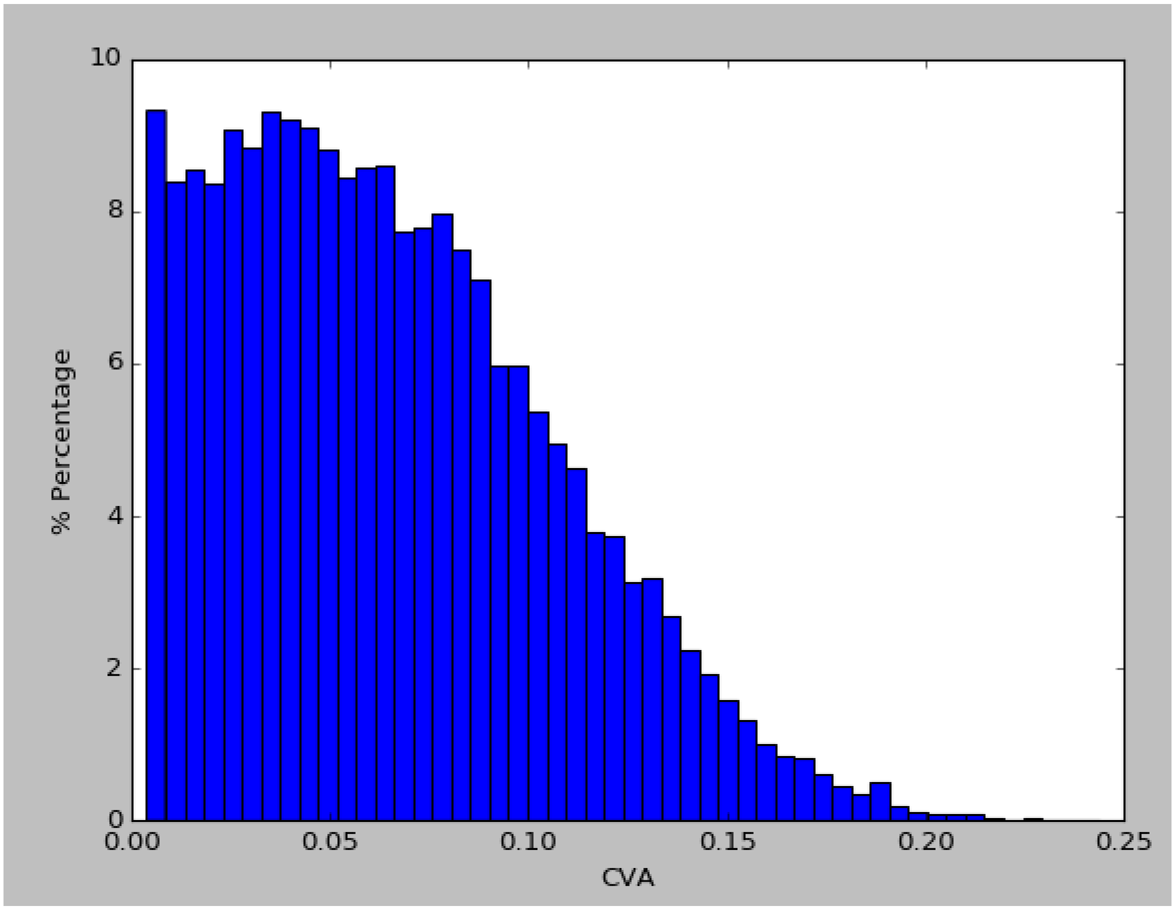
a) CVA curve isn’t flat but a kind parabola
b) The risk is higher at the beginning time period
c) Higher percentile, the skew of the curve is bigger

For default probability, I use the credit triangle formula to calculate the hazard rate then get the survival probability level.


Present Value of IRS
MtM
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0 1 2 3 4 5 6 7 8 9
Expected Exposure of IRS
EE
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0 1 2 3 4 5 6 7 8 9


Present Value of IRS
MtM
0.05
0.04
0.03
0.02
0.01
0.00
0 1 2 3 4 5 6 7 8 9
50%
90%
95%
99%
Expected Exposure of IRS
EE
0.05
0.04
0.03
0.02
0.01
0.00
0 1 2 3 4 5 6 7 8 9
50%
90%
95%
99%

The graph below is the distribution of CVA for simulation = 10000



Percentile	CVA
50%	0.605297472791
90%	0.124584152866
95%	0.142041141515
99%	0.175339445745

7. Summary

In this practice, I explored the formulas, HJM SDE, CVA calculation, and Interest Rate Swap valuation, and numerical implementation, Trapezodial Rule and Simpson's Rule, and Jacobi Transformation for Principal Component Analysis. I programmed the functions in Python and used some external libraries, numpy, pandas, and matplotpyplot for different perspectives to speed up the development.

8. References

- Principal Component Analysis Uncorrelated Factors for Yield Curve

  by Richard Diamond
- The XVA Challenge – Counterparty Credit Risk, Funding, Collateral and Capital

  by Jon Gregory
- Paul Wilmott on Quantitative Finance

  by Paul Wilmott
- A primer for Mathematics of Financial Engineering

  by Dan Stefanica