C++ Vanilla Fixed Income Library – *KIRA*

Lun Li [a],
[a] *Stevens Institute of Technology, Business School*

Abstract In this project, we implemented a simplified version of fixed income library – KIRA[1] – for vanilla desk. The analytics implemented allows pricing/risk linear product, e.g., FRA, Swap, as well as nonlinear products, Cap/Floorlet and Swaption. Although a limited range of models and products are supported, the framework is reasonably general for further developments. In addition, we discuss delivery of analytics to the desk and show a demo in a static setting.
Keywords: Library Design, Fixed Income, Yield Curve, SABR Model

Introduction

The emergence of the subject quantitative finance was in 1900 when *Louis Bachelier's*[1] thesis provided a model to price options under a Normal Distribution. Later on, the development of continuous-time model and stochastic calculus leads to the seminal *Balck-Scholes-Merton* model [2,3]. Those advanced mathematical models are developed to price complex financial derivatives traded by investment bank, insurance company, hedge funds, e.t.c.. To implement those, people has strong quantitative / software engineering backgrounds are needed, this is origin of the career as a Quant. Emanuel Derman's book[4] helped to both make the role of a quantitative analyst better known outside of finance, and to popularize the abbreviation "Quant" for a quantitative analyst.

Nowadays, almost all investment bank that are involved in trading derivatives business needs a quantitative library for pricing and risk analytics. As such, quantitative analyst, in particular, front office quant takes ownership of in-house library for maintenance and development. In this project, we developed a simplified fixed income library focus on a set of vanilla instruments. The framework, however, is general to be expanded to a sizable library. In the following sections, we will go over basic derivative pricing concepts, library structure as well as some demos to demonstrate the interactions between end user, traders, and library.

1. Generalities; Model & Products

In this section, we give a simple introduction of basic interest rate derivatives[5]: zero-coupon bond, money market deposit, forward rate agreement, swap, caplet/floorlet and swaption. Some of them are used later on to build simple interest rate yield curve, while

[1] The name of the library is my cutie "daughter", as I am a dog-lover.

others will be priced with volatility models. We will also briefly go over yield curve construction, as well as volatility models, in particular, Bachelier Normal volatility model and Stochastic Alpha Beta Rho model.

1.1. Zero Coupon Bond & Discounting Factor

Let us first define our building block – discounting factor, or, equivalently, zero coupon bond. A zero-coupon bond is a debt security that does not pay coupon, instead it trades at discount, the profit is redeemed until maturity. If we have continuously-compounded spot interest rate $R(t,T)$ for the period $[t,T]$. To understand, we divided $[t,T]$ into N periods, using single compounding,

$$P(t,T)\left(1+\frac{R(t,T)}{N}\right)^{N\Delta(t,T)}=1 \tag{1}$$

As it is continuously compounded, we let $N\to\infty$,

$$P(t,T)=\exp\left\{-R(t,T)\cdot\Delta(t,T)\right\} \quad \Rightarrow \quad R(t,T)=\frac{\ln P(t,T)}{\Delta(t,T)} \tag{2}$$

A more general quantity to model zero coupon bond is *instantaneous forward rate* $f(t,u)$, which is a deterministic function of running time u, in this setting, we can write

$$P(t,T)=\exp\left\{-\int_t^T f(t,u)\,du\right\} \tag{3}$$

Comparing (2) and (3), we found the coincide if assuming $f(t,\cdot)\equiv f$. So in this project, we will use $f(t,\cdot)$ as fundamental quantity although it is really $R(t,\cdot)$ since we make it constant.

1.2. Money Market Deposit

Cash deposits are usually referring to contracts in which banks borrow in the inter-bank market for a fixed term of at most one year. And the interest rate is usually associated to LIBOR index. To be general, let us assume it is a lending between arbitrary counterparties with cash rate C. The mechanism is fairly simple. At t_0, the borrower enters the trade and will receive \$1 at settlement date t_s. At the end of the deposit term T, the lender is repaid the borrowed cash amount plus interest rate accrued, $1+\Delta(t_s,T)\cdot C$, where C is cash rate associated with accrual period $\Delta(t_s,T)$.
The current value of the contract can be calculated as follows:

$$V(0)=-P(0,t_s)+\big(1+\Delta(t_s,T)C\big)\cdot P(0,T) \tag{4}$$

where $P(t,s)$ is LIBOR discounting factor from t to s. As the trade costs \$0 to enter at t_0, we have

$$C=\frac{1}{\Delta(t_s,T)}\left(\frac{P(t_0,t_s)}{P(t_0,T)}-1\right) \tag{5}$$

Figure 1. Money-Market Deposit

1.3. Forward Rate Agreement

Forward rate agreement(FRA) is for borrowing money for a forward starting period at a rate determined today. This rate is known as the FRA rate which we denote it as $F_0 \equiv F(0;T,S)$(for simplicity, we always assume $T-S=3M$). To describe the trade, we need to understand dates below:

1. t_0 is the trade date, t_s is the settlement date (usually $2b$ after t_0);
2. T is the forward starting date from when interest rate starts to accrue;
3. S is the end date, on which the effective payment is equal to: $\Delta(T_1,T_2)\cdot\big(L(T,S)-F_0\big)$

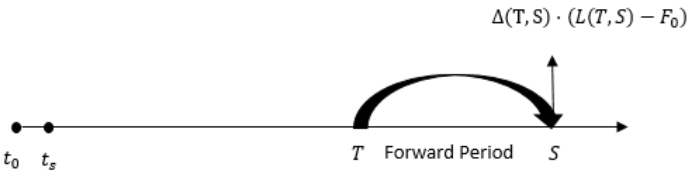

Figure 2. Forward Rate Agreement

To price FRA at time t_0, we use a *replication strategy*:

1. Pay $P(t_s,T)$ to purchase \$1 face value of a T maturity zero coupon bond which settles at time t_s;
2. Sell face value $(1+\Delta(T,S)F_0)$ of a S maturity zero coupon bond which settles at time t_s;
3. When the T maturity bonds mature at time T, invest the \$1 received in a S maturity money-market deposit. This pays $(1+\Delta(T,S)L(T,S))$ at a later time S

The results of the strategy means that:

1. At time t_s we pay $P(t_s,T)-(1+\Delta(T,S)F_0)P(t_s,S)$;
2. At time S we receive $\Delta(T,S)(L(T,S)-F_0)$.

Obviously, the payoff at time S is the same as an FRA, then we require at time t_s it costs nothing to enter. Therefore,

$$F_0 = \frac{1}{\Delta(T,S)}\left(\frac{P(t_s,T)}{P(t_s,S)} - 1\right). \tag{6}$$

And the value of the contract is:

$$V(t_0) = P(t_0,S)\Delta(T,S)\big(L(T,S) - K\big) \tag{7}$$

where K is pre-determined strike.

1.4. Interest Rate Swap

Interest rate swap(IRS) can be thought as a generalization of FRAs, as it is multi-period. There are two parties in an IRS transaction, one party pays fixed rate periodically, the other party pays floating rate (LIBOR rate) periodically. Conventionally, we call them fixed leg, floating leg, respectively. Notice, the payment frequency for different legs can be different, but for simplicity, let us assume they are the same. Notice here t_s can be right after t_0, we call it spot starting IRS, while if $t_s >> t_0$, it is termed as forward starting IRS.

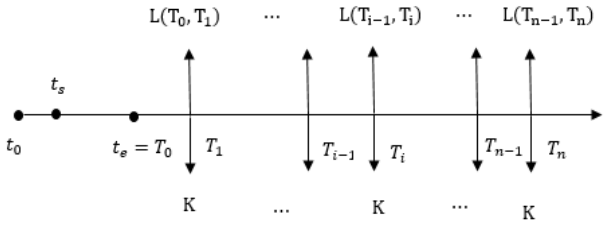

Figure 3. Interest Rate Swap

The fixed leg value can be calculated as

$$V^{fixed}(t_s) = \sum_{i=1}^{n} \Delta(T_{i-1},T_i)P(t_s,T_i)K \tag{8}$$

and, floating leg:

$$V^{float}(t_s) = \sum_{i=1}^{n} \Delta(T_{i-1},T_i)P(t_s,T_i)F(t_s;T_{i-1},T_i) \tag{9}$$

As the contract costs \$0 to enter, $V^{fixed}(t_0) = V^{float}(t_0)$, which enables us to determine par swap rate,

$$S(t_s;T_0,T_n) = \frac{P(t_s,T_0) - P(t_s,T_n)}{\sum_{i=1}^{n}\Delta(T_{i-1},T_i)P(t_s,T_i)} \tag{10}$$

On the other hand, the valuation of swap at t_0 is

$$V(t_0) = \sum_{i=1}^{n} \Delta(T_{i-1},T_i)P(t_0,T_i)\Big(F(t_s;T_{i-1},T_i) - K\Big) \tag{11}$$

1.5. Caplet/Floorlet

Caplet/Floorlet can be thought as an call/put option on FRA. It provides the option holder an interest rate ceiling/floor on floating rate. The payoff of cap/floor at settlement date t_s is:

$$V(t_s) = \big(L(T,S) - K\big)^+ \; or \; \big(K - L(T,S)\big)^+ \tag{12}$$

where K is pre-determined strike. At t_0, the valuation is going to be (for call)

$$V(t_0) = P(t_0,t_s)\mathbb{E}\left[\left(L(T,S) - K\right)^+\right] \tag{13}$$

To calculate expectation, a volatility model needs to be given. As it is European style option, the only thing matters is implied volatility σ, then we can use Black-Scholes formula

$$V(t_0) = P(t_0,t_s)BS\big(F_0,t_s,K,\sigma,optType\big) \tag{14}$$

1.6. Swaption

Swaption is an option on IR swap. It(payer swaption, similar logic holds for receiver swaption) gives the owner right to enter a payer swap at given swaption maturity where the owner becomes the fixed rate payer aat pre-determind level K. The instrument is typically used to mange interest rate risk arising from their core business, for example, a corporate wants proetection from rising interest rate might buy a payer swaption. Then in the future, the corp can be financed at the fixed rate if the floating rate at that time has increasing trend.

The payoff of the Swaption at expiry time is basically the positive side of spot starting swap:

$$\left(\sum_{i=1}^{n} \Delta(T_{i-1},T_i)P(T_0,T_i)\Big(F(T_0;T_{i-1},T_i) - K\Big)\right)^+ \tag{15}$$

With change of numeraire technique, we can derive the time t_0 value of the option is:

$$V(t_0) = A(t_0;T_0,T_n)\mathbb{E}\left[\left(S(T_0;T_0,T_n) - K\right)^+\right] \tag{16}$$

$$= A(t_0;T_0,T_n)BS\big(S_0,t_s,K,\sigma,optType\big) \tag{17}$$

where $A(t_0;T_0,T_n)$ is called annuity defined as:

$$A(t_0;T_0,T_N) = \sum_{i=1}^{N} \Delta(T_{i-1},T_i)P(t_0,T_i) \tag{18}$$

1.7. Yield Curve Model

A yield curve can be think of as a function $T \mapsto P(t_0, T)$, it provides discounting factor for arbitrary maturities. As we can observe above, all products above requires such information, especially, for linear products, actually, the set of discounting factor uniquely determine their valuations. Why we need a yield curve model? On market, traders can trade arbitrary products but not all of them are liquid, we want to extrpolate/interpolate those information from liquid points. To be more specific, to build a yield curve, we usually gather:

- Cash Deposite: 1BD, 1M and 3M cash rates;
- FRA: 3m x 6m, 6m x 9m, 9m x 1Y, e.t.c, those are forward rate levels;
- Swap: 1Y, 2Y, 3Y, e.t.c, these are spot swap rate.

Two immediate observations are, different products are responsible for different regions of the curve(usually not overlapped), i.e., cash is for the short end of the curve, FRA is for the middle range, while swap rate is for the long end. Secondly, all of them are standard tenors(not specific to a date), we call them pillar points and they're relatively liquid on the market. Constructing a yield curve is essentially synthesizing the current market level so that our pricing will be up to market as well. For example, if a client comes in, wants to trade $2019-11-18x1Y$ forward starting swap, we can price use yield curve although it is not a standard point on the curve.

Construction of yield curve is essentially solving an optimization problem:

$$\min_{x^i : i \in I} \sum_{i=1}^{N} \left(x^M - F(x^i) \right)^2 \tag{19}$$

where x^M are market quotes and $x^i (i \in I)$ are model internal parameters, in this case, the instantaneous forward rate. The function F here is basically pricing function mentioned above, they takes x^i to get discounting factors and then reassmeble discounting factors to specific product valuations. Notice, we choose all instruments that don't overlap, thus we can assign each termination date an internal parameter x^i, which basically make instantaneous forward rate as a piecewise constant function on time doamin.

1.8. Volatility Model

One of the simplest model for vanilla option pricing in fixed income domain is probably Bachelier[1], which assumes normal volatility instead of log-normal(this is because the prevailing negative interest rate in Europe invalidates logrnomal volatility model). The Bachelier formula assumes the underlying dynamics S_t as:

$$dS_t = \sigma dW_t, \quad S_0 = S \tag{20}$$

states, given forwad rate, normal volatility, strike, time to expiration and option type, we have(for call option)

$$C(t_0) = (S-K)\Phi\left(\frac{F-K}{\sigma\sqrt{T}}\right) + \sigma\sqrt{T}\phi\left(\frac{S-K}{\sigma\sqrt{T}}\right) \tag{21}$$

However, Bachelier model fails to model volatility smiles in the market as it uses a single parameters to model volatility. Local volatility model as well stochastic volatility model is then proposed to better fit the market smile. In fixed income domain, the prevailing model is Stochastic-Alpha-Beta-Rho model by [6], it assumes the underlying as

$$\begin{cases} dS_t = \alpha_t (S_t)^{\beta} dW_t^1, \quad S_t = S, \\ d\alpha_t = \nu \alpha_t dW_t^2, \quad \alpha_0 = \alpha, \\ \langle dW_t^1, dW_t^2 \rangle = \rho dt. \end{cases} \tag{22}$$

which is captured by four parameters $(\alpha, \beta, \rho, \nu)$. There eixsts a closed formula to price an option with payoff(Section 2.3 in [6]), $\mathbb{E}[(S-K)^+]$, which makes it very appealing. In addition, it can be re-parameterized by $(\sigma, \beta, \rho, \nu)$, where normal volatility is a function of the rest of parameters, i.e.,

$$\sigma = G(\alpha, \beta, \rho, \nu) \tag{23}$$

Notice here G consists of mapping from α to log-normal volatility σ^{LN} and mapping from σ^{LN} to σ(via one price principle). The four parameters have separated role, σ is used to match ATM option, β is usually set as constant (with some historical regression), ν and ρ controls smile and skewness.

2. Library Design

In this section, we illustrate the library design. We start by giving a high-level overview and then dril into each components.

2.1. Library Overview

On a high-level, the library consists of three main objects:

- **Model** Model family includes yield curve model, volatility model that will be used to price / risk various products;
- **Product** Product provides a description, from which all trade information can be extracted;
- **Valuation Engine** Combine information from model and product to execute pricing analytics.

Their interactions are best described by the *Figure 4*. Basically, we can summarize as follows:

- Whenever a request is submitted, it locates the pricing model and product, which then get maps to corresponding valuation engine;
- Valuation engine analyze information of the product, request from model financial quantities required to value the products and execute relevant calculations;
- Based on request, valuation engine decides to send the results back.

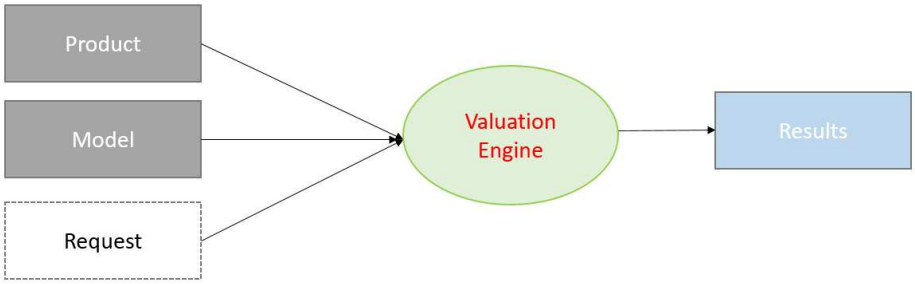

Figure 4. Library Design Overview

2.2. Model

We implemented a model abstract class, which requires model takes in the following four inputs:

- ValueDate - the as-of-date of the model;
- ModelType - the type of the model, YieldCurve, Volatility Model, e.t.c.;
- ModelBuildMethod - instructions how the model should be constructed;
- DataCollection - All the market data the model is built from.

Model is enforced to answer basic questions, such as get the value date, model type, number of internal parameters, as well as execution of calibration, these are set as virtual function in model class:

```
virtual string getModelType() const = 0;
virtual Date getModelValueDate() const = 0;
virtual size_t getNumOfParameters() const = 0;
virtual void calibration() = 0;
```

When a model is required to be constructed, it will scan the build methods to fish out required data objects and configurations if any. These are set up for calibration. Let's take a yield curve model *ModelYieldCurve* to demonstrate(see *Figure* **??**). The constructor looks into build method, the target tells it is building a LIBOR-3M curve, then it scans key-value-pair one by one, which is basically calibration instruments. The data type and data convention form a unique identifier to fetch it from data collection which is basically a map. Each item in data collection is a data object that can be further expanded. For example, if it is a Data 1D data object, it will have axis1, the tenor, and axis2, the corresponding level. By gathering all these products, the calibration can be executed, we will come back to calibration later on.

Another point worth mentioning is we also put hierarchy in model objects, for example, all volatility models requires yield curve to provide discounting factor, forward rate, thus volatility model sits on top of yield curve model. On one hand, it allows volatility model to price linear products as it is a higher order model, on the other hand, it allow efficient construction of model. When volatility level changes, we don't need to start from scratch to construct model, we can take an existing yield curve and only rebuild the volatility part.

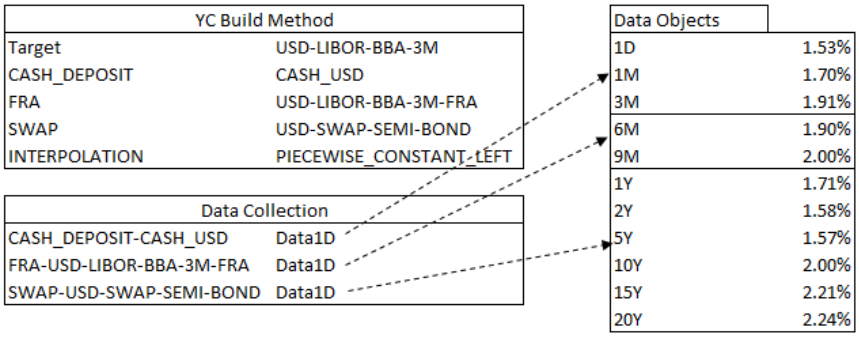

Figure 5. Build Method and Data Collection

2.3. Product

Similarly, we implemented an abstract class, Product, as all products have a lot overlapping questions need to be answered and thus shared members.

```
class Product
{
public:
  Product(string const& uniqueName, string const& valueDate, double const& ntl, string const& buyOrSell, string const& productType) :
    m_uniqueName(uniqueName), m_valueDate(valueDate), m_notional(ntl), m_buyOrSell(buyOrSell),  m_productType(productType) {};

  string uniqueName() const { return m_uniqueName; }
  string valueDate() const { return m_valueDate; }
  double notional() const { return m_notional; }
  string buyOrSell() const { return m_buyOrSell; }
  string productType() const { return m_productType; }
  Date spotDate() const { return m_spotDate; }
  Date termDate() const { return m_termDate; }
  void setSpotDate(Date const& d) { m_spotDate = d; };
  void setTermDate(Date const& d) { m_termDate = d; }

  virtual string tenorOrDate() const = 0;
  virtual double rate() const = 0;

private:
  string m_uniqueName;
  string m_valueDate;
  Date m_spotDate;
  Date m_termDate;
  double m_notional;
  string m_buyOrSell;
  string m_productType;
};
```

Figure 6. Product Abstract Class

As we can see from *Figure 6*, a set of getter functions retrieve product information, such as notional, buy or sell, termination date and so forth. We also have setter function to compute spot date, effective date, e.t.c.. The derived class, when called, will first invoke base class constructor to assign basic information, then it derives its special information based on own natural.

2.4. *Valuation Engine*

The abstraction of valuation engine is *valuationEngine* class, which has the following pure virtual member functions:

```
class valuationEngine
{
public:
  virtual void calculateValue() = 0;
  virtual double parRateOrSpread() = 0;
  virtual double pv01() const = 0;
  virtual double value() const = 0;
};
```

Figure 7. Valuation Engine Abstraction

The *calculateValue()* function calculates value of the contract and other relevant quantities, the *parRateOrSpread()* function calculates break-even rate and *pv01()* calculates the duration of the products, finally *value()* returns back the NPV of the trade.

Valuation engine proceeds as follows:

- In constructor/initialize phase, it pulls out information from products;
- In calculation phase, based on product information, it asks model to provide financial quantities, such as discounting factor, normal volatility, e.t.c..;
- In post-process phase, based on request, with little more effort(as heavy duty is already accomplished in calculation phase), it delivers the corresponding results.

Besides these basic operations, specific valuation engine could render more information that can be useful. For example, for volatility valuationEngine, it can also renders implied normal volatility, as it is anyway being extracted and utilized in the calculation phase.

```
⊞class valuationEngine { ... };
⊞class valuationEngineCashDeposit { ... };
⊞class valuationEngineFRA { ... };
⊞class valuationEngineSwap { ... };
⊞class valuationEngineCapFloorLet { ... };
⊞class valuationEngineSwaption { ... };
⊞class valuationEngineSABRCapFloorLet { ... };
⊞class valuationEngineSABRSwaption { ... };
```

Figure 8. Valuation Engine List

2.5. *Miscellaneous*

Both build method and static data conventions are of the format name-value pairs. We collect all basic functionalities in base *NameValuePair class*(*Figure 9*) to avoid duplications. Namely, it can display the whole dictionary, retrieve items by key, e.t.c.. Build method inherits from it, then it gets specialized into different model's build methods (see hierrachy in *Figure 10*).

```
class NameValuePair
{
public:

  NameValuePair(string const& uniqueName, vector<vector<string>> const& nvp)
    : m_uniqueName(uniqueName), m_nvp(nvp), m_nvpType("") {
    initialise();
  };
  NameValuePair(string const& fileName);
  void initialise();
  size_t numItems() const { return m_nvp.size(); };
  void displayNVP() const;
  void setType(string const& type) {
    m_nvpType = type;
  }
  string getValueByKey(string const& key) const { return m_map.at(key); };

private:
  string m_uniqueName;
  string m_nvpType;
  vector<vector<string>> m_nvp;
  map<string, string> m_map;
};
```

Figure 9. Name Value Pair Base Class

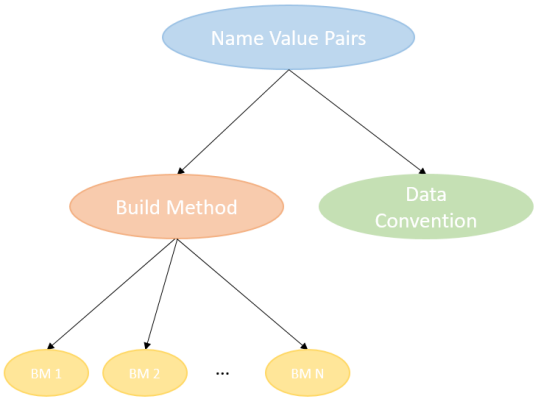

Figure 10. Name Value Pair Hierrachy

We separate all lower level utilities to *"utilities.h"*. Black-Scholes(1973) formulae and Bachelier(1900) analytics implemented, *newton Raphson solver* is implemented with

analytical derivative function to speed up convergence, they are used to invert price from volatility, convert back and forth between log normal volatility and normal volatility. Besides, a suite of SABR analytics are avilable to calculate equivalent lognormal volatility as well as normal volatility, equivalent alpha calculation is also included there to allow different parameterizations.

2.6. Calibration

Valuation engines are not only used as part of calculating user requests but also in model calibration. As mentioned in *Section 2.4*, for instance, in calibrating yield curve model to swap quotes, it is obvious that swaps need to be price (to calibrate prices) and the same valuation engine needs to be used. For calibration purpose, valuation engines are wrapped into a setup that can be called in the context of solvers. Ultimately, we are matching market quotes by calling valuation engine that takes an evolving internal parameters x^I. Specifically, if we abstract valuation engine as a functional, it really takes in model internal parameters and render results, i.e.,

$$V = F(x^I, x^M) \tag{24}$$

Caibration amount to finding a zero of the functional F by choosing suitable internal parameters. For this purpose, the valuation engine is wrapped into an objective function that

- Applis a perturbation to the internal model parameter;
- Calculates the value of the functional under the perturbation.

Using this as the target in standard solve (such as *Newton Raphson*) will then form the backbone of any model calibration. Below is pseudo code of yield curve calibration:

Algorithm 1 Yield Curve Calibration

```
// loop through calib instruments
for EACH productType in CalibInstruments do
    dataObject = getDataObject(productType)
    Axis1 = getAxis1(dataObject), Values = getValues(dataObject)
    for Each (tenor, quote) in Axis1, Values do
        calibration(productType, tenor, quote)
    end for
end for
// calibration sub-routine
fun calibration(producType, tenor, quote)
    prod = createProduct(productType, tenor, quote)
    x^I = getHoldOfInternalParams(prod)
    ve = map[prod + model]
    x̃^I = solver(x^I, ve, prod)
    ResetInternalParameters(x̃^I)
```

Notice the main loop walk through all calibration instruments type and extract corresponding axis and levels. Those are sent to corresponding calibration facility, where calibration products are created according to axis and levels. Then, the product locates the internal parameters as well as appropriate valuation engine, they are taken by a solver to finally solve for the optimal internal parameters.

	Curve Marking	
Cash	1D	1.91%
FRA	2019-12-18	1.91%
	2020-01-15	1.83%
	2020-02-19	1.76%
	2020-03-18	1.72%
	2020-06-17	1.62%
	2020-09-16	1.54%
	2020-12-16	1.53%
	2021-03-17	1.46%
	2021-06-16	1.45%
	2021-09-15	1.44%
	2021-12-15	1.47%
	2022-03-16	1.47%
	2022-06-15	1.49%
	2022-09-21	1.51%
SWAP	4Y	1.58%
	5Y	1.59%
	6Y	1.61%
	7Y	1.63%
	8Y	1.65%
	9Y	1.68%
	10Y	1.71%
	12Y	1.75%
	15Y	1.80%
	20Y	1.86%

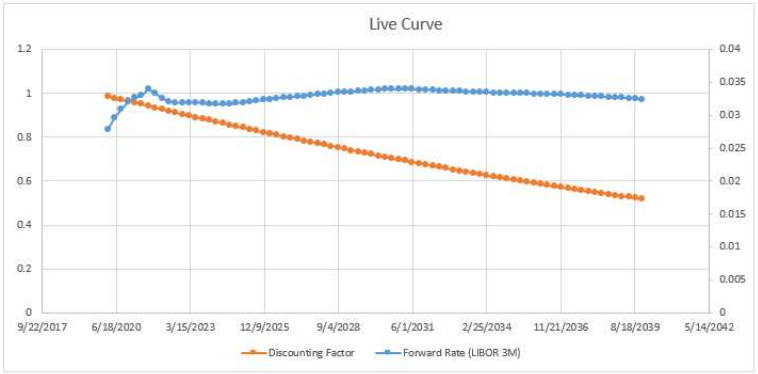

Figure 11. Keep Track Of Market Rates

3. Library Demo

We provided a set of APIs can be exposed to other scripting languages, e.g., Excel(VBA), Python, e.t.c., they are:

- *kiraCreateModel:* A generic API to create model given unique name, value date, data collection, build method and model type;
- *kiraCreateModelYieldCurve:* API to create yield curve model specifically;
- *kiraCreateVolModel:* Instead of starting from scratch, it takes in a cached yield curve and build volatility model on top of it;
- *kiraGetDiscountFactor:* Given a yield curve and termination date of interests, it returns the discounting factor associated;
- *kiraGetValueFwd:* Given a yield curve and termination date, it returns LIBOR-3M forward rate;
- *kiraCreateProduct:* A generic API, given a series of properties, including product type, it creates a product as requested;
- *kiraCreateValueReport:* Given a model, product and request, it delivers the valuation results.

We start with a yield curve marker, which is used to keep up with rates market, as soon as traders observe liquid points are updating, they will mark it down locally. Then, they execute *kiraCreateModel*, which takes in the current data 1d table to re-generate yield curve model, discounting curve(*kiraGetDiscountingFactor*) and forward rate curve(*kiraGetFwdRate*) can be visualized simultaneously as well(see *Figure 11*).

Similarly, volatility desk needs to re-mark their volatilities and skew parameters if needed. If using SABR model, traders monitor BBG for quotes on standard points. Usually, ATM options are quite liquid, so they need to update ATM normal volatility matrix often, while skew trades are less frequent, they usually change skew every week or more. See *Figure 12* the marking schema for SABR parameters.

Normal Volatility(bps)	3M	1Y	2Y	5Y	10Y	15Y	20Y
1W	23.02	23.59	33.37	43.82	44.94	44.43	43.92
1M	23.47	26.29	37.19	48.83	50.08	49.51	48.94
3M	26.18	30.19	42.30	52.77	54.40	53.45	52.49
9M	39.03	45.94	55.97	61.38	61.77	60.16	58.76
1Y	45.48	52.58	60.89	64.40	63.92	62.00	60.57
18Y	56.14	60.84	66.68	67.61	66.17	63.87	62.39
2Y	63.86	67.72	72.16	70.69	68.24	65.52	64.01
5Y	78.64	78.25	77.55	74.20	70.50	66.58	64.39
10Y	73.83	73.10	72.11	69.82	65.56	61.20	58.88
15Y	66.87	66.20	64.58	62.23	58.71	55.24	53.38
20Y	60.44	59.84	58.38	56.12	53.07	50.24	48.73

Beta	3M	1Y	2Y	5Y	10Y	15Y	20Y
1W	40%	40%	40%	40%	40%	40%	40%
1M	40%	40%	40%	40%	40%	40%	40%
3M	40%	40%	40%	40%	40%	40%	40%
9M	40%	40%	40%	40%	40%	40%	40%
1Y	40%	40%	40%	40%	40%	40%	40%
18Y	40%	40%	40%	40%	40%	40%	40%
2Y	40%	40%	40%	40%	40%	40%	40%
5Y	40%	40%	40%	40%	40%	40%	40%
10Y	40%	40%	40%	40%	40%	40%	40%
15Y	40%	40%	40%	40%	40%	40%	40%
20Y	40%	40%	40%	40%	40%	40%	40%

NU	3M	1Y	2Y	5Y	10Y	15Y	20Y
1W	1.65	1.65	1.60	1.55	1.55	1.56	1.56
1M	1.65	1.65	1.60	1.55	1.55	1.56	1.56
3M	1.25	1.25	1.20	1.20	1.20	1.21	1.21
9M	0.69	0.69	0.71	0.69	0.69	0.69	0.70
1Y	0.59	0.59	0.59	0.59	0.59	0.59	0.59
18Y	0.49	0.49	0.49	0.49	0.49	0.49	0.49
2Y	0.44	0.44	0.44	0.43	0.43	0.43	0.43
5Y	0.34	0.34	0.34	0.32	0.32	0.32	0.32
10Y	0.30	0.30	0.29	0.29	0.29	0.29	0.29
15Y	0.29	0.29	0.29	0.29	0.29	0.29	0.29
20Y	0.29	0.29	0.29	0.29	0.29	0.29	0.29

Beta	3M	1Y	2Y	5Y	10Y	15Y	20Y
1W	-0.55	-0.52	-0.40	-0.20	-0.08	-0.08	-0.08
1M	-0.55	-0.52	-0.40	-0.20	-0.08	-0.08	-0.08
3M	-0.55	-0.52	-0.40	-0.20	-0.08	-0.08	-0.08
9M	-0.55	-0.52	-0.40	-0.21	-0.09	-0.09	-0.09
1Y	-0.55	-0.52	-0.40	-0.23	-0.10	-0.10	-0.10
18Y	-0.53	-0.50	-0.39	-0.24	-0.12	-0.12	-0.13
2Y	-0.50	-0.48	-0.37	-0.25	-0.14	-0.15	-0.15
5Y	-0.31	-0.28	-0.26	-0.22	-0.20	-0.21	-0.21
10Y	-0.21	-0.21	-0.21	-0.22	-0.23	-0.24	-0.25
15Y	-0.20	-0.20	-0.20	-0.21	-0.21	-0.23	-0.24
20Y	-0.20	-0.20	-0.20	-0.21	-0.21	-0.23	-0.24

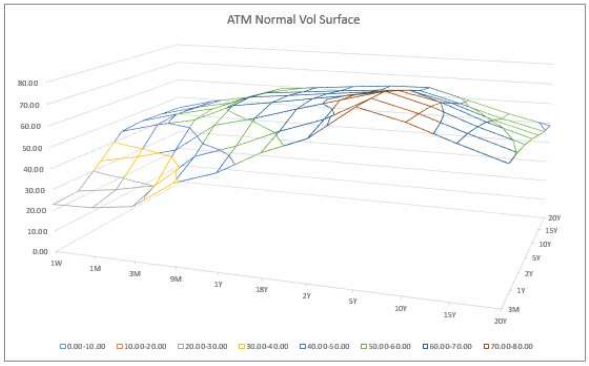

Figure 12. Volatility Marking Matrices

By continuously synthesizing market information, i.e., remarking - rebuilding model, or equivalently, executing *kiraCreateModel* and *kiraCreateVolModel*, traders are able to price for external clients via *kiraCreateValueReport*. Below is a screenshot of line-pricer[2]:

User Inputs							Output		
Product Type	Buy	Ntl	Expiry	Tenor	PayOrRec	Strike	ParRate	PV	PV01
SWAP	BUY	10,000.00	3/3/2020	5Y	REC	3.30%	3.17%	5703.35	4.53
SWAP	SELL	20,000.00	2D	2Y	PAY	2.56%	3.37%	-308.96	1.915
FRA	BUY	10,000.00	4M	3M	PAY	2.35%	2.43%	203.059	0.24
FRA	BUY	30,000.00	6M	3M	REC	2.40%	2.97%	4175.44	0.26
SWAPTION	SELL	50,000.00	6/30/2020	10Y	PAY	2.40%	3.25%	-350189	8.03
CAPFLOOR	BUY	40,000.00	5/2/2021	3M	CAP	2.65%	3.25%	5948.07	2.1
SWAPTION	SELL	1,000,000.00	10Y	10Y	PAY	3.43%	3.34%	-43106	6.1

Figure 13. Line Pricer

[2]Line pricer really means one line one trade one valuation.

It is surprising that traders actually not trying to calibration to option on the market in the traditional sense of model calibration, that's why volatility model building does not take market option quotes as input but directly taking parameters (unlike yield curve, which takes market quotes). There are two reasons:

- There are not that many liquid points along strike directions, and one would hardly trust these quotes even if it show rarely;
- Traders want to have completely control of which set of points to hit when executing a trade as those are perhaps the only hedging instruments to them.

We show below a graph by manually adjusting SABR parameters to hit market price.

	Strikes	Mkt Vol	SABR Vol
REC	3.0320%	0.00731715	0.00718215
REC	3.1420%	0.006966171	0.00695718
REC	3.1920%	0.006732758	0.00672692
REC	3.2420%	0.006496899	0.0064911
REC	3.2920%	0.006251315	0.00624941
PAY	3.3420%	0.006038365	0.00600092
PAY	3.3920%	0.006012451	0.00599423
PAY	3.4420%	0.00599578	0.00598663
PAY	3.4920%	0.006009308	0.00598241
PAY	3.5420%	0.00606366	0.00597729

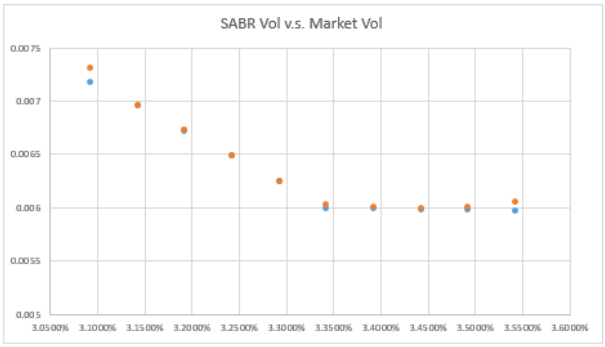

Figure 14. Manually Calibrate to the Market

Acknowledgement: *This project cannot be accomplished without discussions with my friend who is a desk quant in major investment bank. He gives me a lot valuable suggestions in terms of library design as well as fixed income derivatives knowledge.*

References

[1] Bachelier, L. *Théorie de la spéculation* Annales Scientifiques de l'École Normale Supérieure, 3 (17), pp. 21-86, 1900a.

[2] Black, F.; Scholes,M. *The Pricing of Options and Corporate Liabilities*. Journal of Political Economy, 81 (3): 637654, 1973.

[3] Merton, R. *Theory of Rational Option Pricing*. Bell Journal of Economics and Management Science. 4 (1): 141183, 1973.

[4] Derman, E. *My Life as a Quant.* John Wiley and Sons, 2004.

[5] Damiano, B., Fabio M. *Interest Rate Models - Theory and Practice: With Smile, Inflation and Credit*. Springer; 2nd edition, 2006

[6] Hagan, P., Kumar, D., Lesniewski, A. *Manage Smile Risk*, Wilmott Magazine, Page 84, 2002

[7] Hagan, P., West, Graeme. *Methods for Constructing a Yield Curve* Wilmott Magazine, p.29, 2006