QLIB for SwapClear

dibyendu.majumdar@lchclearnet.com

Revised June 6, 2013

Abstract

This document describes the design and implementation of the risk analytics library QLIB, as well as the SMART API that provides a programming API for SwapClear members and clients.

1 History

QLIB originated from SMART C++/Excel product which was a margin estimation tool created by the SwapClear Risk Team. Unlike the original risk analytics engine which was written in C++, QLIB is a pure Java library. The move to Java was completed after it was realised that the Java version performed significantly better than the original C++ version.

QLIB is now being extended to cover Fixed Income and ForexClear. This document however focuses on the SwapClear bits of QLIB.

2 High Level Design

QLIB design follows some basic principles.

QLIB is designed as a set of modules that are organised in a hierarchy. The higher level modules can call any API in the lower level modules, but not vice versa. This ensures that dependencies are only one way.

Orthogonal to the module structure is the DataProvider concept. Each module supplies a set of DataProvider interfaces that define the external data inputs that the module needs in order to function. For instance the dates module provides a DataProvider interface for Holiday Calendar data. Implementations can provide their own DataProviders to fulfil the input data requirements.

The core QLIB functionality is risk analytics. In order to keep this core pure and reusable, we avoid creating dependencies that are not fundamental to the analytics part. For example, the core QLIB components do not manage concurrency via threads, or dependency injection via an IoC container. QLIB code is thread-safe and designed for concurrency, but concurrency is not managed within QLIB. The QLIB components used constructor based injection, but do

not directly rely upon any particular IoC container. Another example is the use of CSV type data files for inputs rather than providing access to Relational Databases. An application using QLIB can use a Relational Database to retrieve data, but QLIB itself must not have such a dependency as it would make QLIB less reusable. Similarly QLIB is not aware of caching technologies such as Coherence, although applications may deploy QLIB within such frameworks.

An important goal of QLIB is to make it easy for anyone to start using it. The only major dependency is the reference and market data that is required by QLIB - and as mentioned above, interfaces are provided for applications to provide their own DataProvider implementations.

Intellectual property is an important concern for QLIB. QLIB is distributed externally to members, clients and trade sources. As such the intellectual property in it must belong to LCH Clearnet, especially when it comes to core risk analytic functions. We sparingly use OpenSource libraries to provide generic capabilities such as logging (Log4J), platform independent serialisation (Google Protocol Buffers) and Matrix Algebra (COLT). The core risk analytic functions are built from first principles using knowhow within the LCH Clearnet's Risk and Quantitative teams.

QLIB is designed to perform in high performance multi-threaded applications. We try to use immutable objects wherever possible to avoid the cost of thread synchronisation and to make it easier to reason about concurrency. Returning array objects and lists from interfaces is inherently unsafe in a multi-threaded environment as these objects are not immutable. Hence we try to avoid such interfaces.

Finally it is worth mentioning that the design avoids overusing inheritance, and relies more upon composition. We need to guard against creating complex abstractions and inheritance hierarchies as these make the system tightly coupled and hard to maintain and understand. We favour loose coupling where components interact with each other via interfaces.

2.1 SMART API

The SMART API is the public interface to QLIB that is available to SwapClear members, clients and trade sources. This API offers a limited set of functions that are guaranteed to work across different versions of QLIB.

There is an internal API that is less controlled, and that is available for use within LCH Clearnet.

The SMART API uses Google Protobuf generated types in its interfaces. This was requested by a Clearing Member who wanted to serialise data so that it could be read by a .Net application. We are however careful to avoid using any features of protobuf within the API as we would like the API to be rebuilt without protobuf if necessary. There is in fact a utility that generates replacement types using the same protobuf definition file.

3 Modules

QLIB is organised as a set of modules. Each module is meant to be reusable to an extent - to ensure reusability, the dependencies between modules is strictly controlled.

3.1 util

This is the lowest level module. It provides basic things such as the Exception hierarchy, utilities for reading CSV files, etc. Most of the code in this module is agnostic to the business functionality.

3.2 dates

Sitting above the util module is the dates module. This module contains implementations of various date and calendar functions such as Day Count Fractions, Business Day Conventions, Holiday Calendars, Date Roll and Schedule generation utilities. The module defines an interface for Holiday Calendar data via the Calendar interface.

A date implementation is provided - this implementation is designed to be compatible with Excel date representation, and is also designed for high performance. For instance the operation of adding or subtracting days is just a simple numeric addition or subtraction.

Note that we deliberately use a custom date implementation rather than a third-party library such as Joda Time as QLIB is very sensitive to the performance of the date library, and we do not require much of functionality offered by third party libraries.

The Schedule Generator is based upon conventions of FpML.

3.3 Multiple Date Interfaces

QLIB contains three different date interfaces. The reasons for this oddity are:

- The SMART API needs to use a protobuf type for date, but we do not want protobuf dependency introduced elsewhere.
- The QLIB library is delivered obfuscated to SwapClear members, clients and SEFs, hence the DataProviders need a date interface that is unobfuscated. We allow external parties to redefine the DataProvider implementations, therefore if we exposed the internal date interface in the unobfuscated API then we would also need to provide mechanisms for such date objects to be created, thereby leaking implementation details.

The internal date interface is `XDate`. The DataProvider uses `SimpleDate` whereas the protobuf definitions use `Date`.

3.4 marketdata

The marketdata module handles the provisioning of FX Rates, Zero Curves and Index Rate Fixings.

The Index Fixings component also handles Index Conventions ; these are market conventions associated with an Index. The SwapClear implementation loads Index Conventions from a CSV format configuration file named `indexConventions.csv`.

The Zero Curves component supports Linear Interpolated Zero Curves only. The interpolated curve is overlayed on top of the raw Curve; the raw Curve is obtained from a DataProvider interface. The SwapClear implementation reads the raw curves from SwapClear End of day reports.

Only Linear Interpolation method is supported as the sensitivity calculations in the valuation module cannot handle any other form of interpolation. If another form of interpolation is required, it will be necessary to ensure that the valuation module is capable of generating sensitivities; additionally the PnL and VaR calculations may also need to be changed.

3.5 trade

The trade module provides the object representation of various trade types. including the ability to parse FpML.

Both StaX and DOM parser implementations are available for FpML, although DOM parser is currently used in Production. We use DOM or StaX parsers as QLIB must be able to parse trades without reliance on a particular version of FpML. We do not insist on a particular message format, instead we just search for the trade element. In particular use of a generic XML to Java binding library must be avoided as we use a more optimised object model that is populated directly.

Any schema validation must be performed by the application using QLIB prior to passing an FpML to QLIB.

The trade module currently supports Swap trades and FRA trades.

3.6 valuation

The valuation module is responsible for pricing trades and generating delta and gamma sensitivities required for Taylor approximation of PnL.

The valuation module calculates delta and gamma sensitivities for Swaps and FRAs. The sensitivities are calculated analytically in an optimised manner. To sensitivity results are stored in arrays that mirror the curve structure.

Amongst swaps, we handle standard Vanilla, Basis, OIS and Variable Notional, Spread or Fixed Rate swaps. Compounding and Zero Coupons are supported. Stubs are handled.

The valuation routines do not amend the underlying trade objects, any data structure needed are constructed within the module, sometimes wrapping trade

level data structures, such as cashflow streams. This design ensures that the valuation module is decoupled from the trade module.

3.7 scenarios

The scenarios module is responsible for managing the VaR model historical scenario data. It also provides capabilities to generate the scenarios given historical market data. The scenario generator use configuration to decide how to scale the historical returns. It supports absolute or relative returns for interest rates, and relative returns for FX rates.

In case of SwapClear, the historical scenarios are by currency, index and tenor. The scenario data is obtained from a SwapClear end of day report; both relative and absolute returns are supported.

3.8 margin

The margin module, as its name implies, is responsible for calculating PnL and VaR at the portfolio level. Both TDG and Full Historical Simulation models are supported, the FHS model uses TDG to shortlist the top ten scenarios. This module also contains IMM calculators.

The VaR model supports both WCL and Expected Shortfall. It supports relative FX shifts, and absolute or relative returns.

In the case of TDG method, the profit and loss of a portfolio is computed as follows for each currency, index and scenario number:

$$PnL = \sum_{i=1}^{n} \left(\frac{\partial PV}{\partial z_i} h + \frac{1}{2} \frac{\partial^2 PV}{\partial z_i^2} h^2 \right) \tag{1}$$

where i ranges over the tenors of the curve, and h is the scenario shift for a particular tenor.

3.9 smart

This contains the SMART API.

3.10 bootstrap

The bootstrap module contains a bootstrapper for creating Zero Curves from market par rates.

4 Risk Analytics Information

4.1 Notation

- t represents a date
- t_s Adjusted accrual start date for a calculation period

- t_e Adjusted accrual end date for a calculation period
- t_v Value date for an interest rate, i.e., the date from which the rate is effective
- t_m Maturity date for an interest rate
- t_f The fixing date when the interest rate is published, typically 1 or 2 days prior to value date
- t_p Adjusted payment date
- Z represents a Zero Curve
- Z_d The Zero Curve used for discounting
- Z_f The Zero Curve used for forecasting forward rates
- N Notional amount
- τ_x Day count fraction as per Index
- τ Day count fraction as per trade definition
- PV Present Value, can be of a payment period, or an entire leg, or an entire trade
- CP Calculation Period, as used in FpML standard, this means the period over which interest is accrued; this period may be a regular one or a front or back stub. Each CP provides a Cash Flow
- PP Payment Period, as used in FpML standard, means the period for which a single payment is made; multiple CP may be contained in a single PP, typically when compounding is required
- z is a zero rate function that returns the zero rate for given Z and t
- s is the floating rate spread
- p is the discount factor, defined as $exp(-z(Z,t)t)$, where Z and t are parameters to the p function.
- z is the forward zero rate between two dates t_1 and t_2 using curve Z

4.2 Pricing

The valuation module contains pricing routines for Interest Rate Swaps and FRAs.

4.3 Swaps

The PV of the swap is defined as the difference between the PV of the two legs of the swap. The direction in which the difference is taken depends upon which party is paying and which party is receiving.

The PV of a leg is the sum of the PV of each PP contained in the leg.

$$PV^{leg} = \sum_{k=1}^{n} PV_k \tag{2}$$

where PV_k is the PV of the k_{th} PP.

4.4 Fixed Leg

For a fixed leg, there are three possibilities.

- A simple cashflow involving a fixed amount; for instance if there is a fee payment, or if the swap is Zero Coupon and the total payable amount on the fixed leg has been precomputed
- A fixed rate cashflow over a single CP that is non-compounding
- A fixed rate cashflow compounded over multiple CP within one PP

4.4.1 Fixed Amount Cashflow

This occurs when there is a fee payment, or when a fixed amount has been agreed at the outset, for instance, for a Zero Coupon swap.

$$PV = a^{fixed} p(Z_d, t_p) \tag{3}$$

A fixed amount cashflow is sensitive to the t_p on the Z_d curve. The delta sensitivity is computed by taking the first derivative as shown below.

The PV expression can be expanded as below.

$$PV = a^{fixed} exp(-z(Z_d, t_p)\, t_p) \tag{4}$$

We treat the rate $r = z(Z_d, t_p)$ as the independent variable to obtain:

$$PV(r) = a^{fixed} exp(-r t_p) \tag{5}$$

Differentiating we obtain following as the delta, i.e. the first order derivative:

$$\frac{dPV(r)}{dr} = -t_p a^{fixed} exp(-r t_p) \tag{6}$$

4.4.2 Fixed Rate Cashflow

In the non-compounding case the fixed rate cashflow gives:

$$PV = N\, r^{fixed} \tau(t_s, t_e)\, p(Z_d, t_p) \tag{7}$$

We expand this to:

$$PV = N\, r^{fixed} \tau(t_s, t_e)\, exp(-z(Z_d, t_p) t_p) \tag{8}$$

We treat the rate $r = z(Z_d, t_p)$ as the independent variable to obtain:

$$PV(r) = N\, r^{fixed} \tau(t_s, t_e)\, exp(-r t_p) \tag{9}$$

Differentiating we obtain following as the delta, i.e. the first order derivative:

$$\frac{dPV(r)}{dr} = -t_p N\, r^{fixed} \tau(t_s, t_e)\, exp(-r t_p) \tag{10}$$

4.4.3 Compounding Fixed Rate Cashflow

In the compounding case, each cashflow except the first one within a PP accrues interest on the non-discounted amount from previous cashflows.

Let A_i represent the amount accrued from CP_i within the PP. Then,

$$A_{i+1} = N\, r^{fixed} \tau(t_{s_{i+1}}, t_{e_{i+1}}) + r^{fixed} \tau(t_{s_{i+1}}, t_{e_{i+1}}) \sum_{k=1}^{i} A_i \tag{11}$$

Let AI be the sum of all the A amounts. That is:

$$AI = \sum_{i=1}^{n} A_i \tag{12}$$

where n is the number of CP in a single PP.

We can now treat AI as a $fixedAmount$ and use the PV formula for a fixed amount. Delta is calculated in a similar way.

4.4.4 Stub periods

Stub periods on the fixed leg are treated in the same way as regular cashflows except that the fixed rate used is the one specified for the stub.

4.4.5 Variable notional or fixed rate

Swaps that have a variable notional or fixed rate are handled the same way as normal swaps except that each period may have its own notional value or fixed rate set. Note that SwapClear requires the notional or fixed rate to remain constant for a PP hence also constant for a compounding period if applicable.

4.5 Floating Leg

Floating leg has following possibilities.

- A non-compounding vanilla floating leg
- A non-OIS compounding floating leg
- An OIS floating leg
- Handling of stubs

4.5.1 Floating Forward Rates

The floating forward zero rate is defined as:

$$r^{float} = \frac{(\frac{p(Z_f, t_v)}{p(Z_f, t_m)} - 1)}{\tau_x(t_v, t_m)} \tag{13}$$

4.5.2 Noncompounding Floating Cashflow

The PV of the non-compounding floating cashflow for a regular CP is given by:

$$PV = N\,(r^{float} + s)\tau(t_s, t_e)\,p(Z_d, t_p) \tag{14}$$

This can be expanded to:

$$PV = N\,(\frac{(\frac{p(Z_f,t_v)}{p(Z_f,t_m)} - 1)}{\tau_x(t_v, t_m)} + s)\tau(t_s, t_e)p(Z_d, t_p) \tag{15}$$

The expanded equation shows that the PV is sensitive to the forward curve zero rate at t_v and t_m, and to the discount curve zero rate at t_p.

Therefore, to calculate the Delta Sensitivity of the floating cash flow, we take the partial derivative the floating rate at each of the curve points (Z_f, t_v), (Z_f, t_m) and (Z_d, t_p). To illustrate this, we let r_v be the rate at (Z_f, t_v), r_m the rate at (Z_f, t_m), and r_p the rate at (Z_d, t_p). Taking r_v as the independent variable, while holding r_m and r_p as constants, and remembering that:

$$p(Z, t) = exp(-r(Z, t)t) \tag{16}$$

we get the partial derivative:

$$\frac{\partial PV(r_v)}{\partial r_v} = \frac{-t_v\,N\,\tau(t_s, t_e)}{\tau_x(t_v, t_m)}\,exp(-r_v t_v)\,exp(r_m t_m)\,exp(-r_m t_p) \tag{17}$$

and similarly for r_v, we get:

$$\frac{\partial PV(r_m)}{\partial r_m} = \frac{t_m\,N\,\tau(t_s, t_e)}{\tau_x(t_v, t_m)}\,exp(-r_v t_v)\,exp(r_m t_m)\,exp(-r_p t_p) \tag{18}$$

And finally for r_p we get:

$$\frac{\partial PV(r_p)}{\partial r_p} = -t_p\,N\,\tau(t_s, t_e)exp(-r_p t_p)(r^{float} + s) \tag{19}$$

4.5.3 Compounding Floating Cashflow

In the compounding case, each cashflow except the first one within a PP accrues interest on the non-discounted amount from previous cashflows.

Flat compounding method omits the spread when compounding the interest.

Let A_i represent the amount accrued from CP_i within the PP. Then,

$$\begin{aligned} A_{i+1} = {} & (r^{float} + s)\tau(t_{s_{i+1}}, t_{e_{i+1}})N \\ & + r^{float}\tau(t_{s_{i+1}}, t_{e_{i+1}})\sum_{k=1}^{i} A_i \end{aligned} \tag{20}$$

In the Straight compounding case, the spread is included when compounding the interest.

$$A_{i+1} = (r^{float} + s)\tau(t_{s_{i+1}}, t_{e_{i+1}})N + (r^{float} + s)\tau(t_{s_{i+1}}, t_{e_{i+1}}) \sum_{k=1}^{i} A_i \tag{21}$$

Let AI be the sum of all the A amounts. That is:

$$AI = \sum_{i=1}^{n} A_i \tag{22}$$

where n is the number of CP in a single PP.

The compounding cashflows are much more computationally intensive when it comes to calculating first and second order derivatives, due to the recursive nature of the PV expression. To simplify the description of how the derivatives are obtained, we first consider the amount accrued prior to discounting. For the Straight Compounding case, this is given by:

$$(r^{float} + s)\tau(t_s, t_e)N + (r^{float} + s)\tau(t_s, t_e)AI_{i-1} \tag{23}$$

Above equation is for the i_{th} cashflow, where AI_{i-1} is the accrued amount up to the $i - 1_{th}$ cashflow. Remembering that the forward zero rate is defined as:

$$r^{float} = \frac{(\frac{p(Z_f, t_v)}{p(Z_f, t_m)} - 1)}{\tau_x(t_v, t_m)} \tag{24}$$

we see that the forward rate has sensitivity to the zero rate at value date and the zero rate at maturity date. We therefore differentiate the equation (23) with respect to the two zero rates. The first order derivative with respect to the zero rate r_v at value date is given by:

$$\begin{aligned}\frac{\partial}{\partial r_v}(AI) = N\frac{\partial}{\partial r_v}(r^{float} + s)\tau(t_s, t_e) \\ + \frac{\partial}{\partial r_v}(r^{float} + s)\tau(t_s, t_e)AI_{i-1} \\ + (r^{float} + s)\tau(t_s, t_e)\frac{\partial}{\partial r_v}(AI_{i-1})\end{aligned} \tag{25}$$

Similarly, the first order derivative with respect to the zero rate r_m at maturity date is given by:

$$\begin{aligned}\frac{\partial}{\partial r_m}(AI) = N\frac{\partial}{\partial r_m}(r^{float} + s)\tau(t_s, t_e) \\ + \frac{\partial}{\partial r_m}(r^{float} + s)\tau(t_s, t_e)AI_{i-1} \\ + (r^{float} + s)\tau(t_s, t_e)\frac{\partial}{\partial r_m}(AI_{i-1})\end{aligned} \tag{26}$$

Note the recursive nature of the derivatives. The second order derivative, gamma, is given by taking derivatives of the equations obtained above. To illustrate the first case, we obtain:

$$\begin{aligned}\frac{\partial^2}{\partial r_v^2}(AI) = N &\frac{\partial^2}{\partial r_v^2}(r^{float}+s)\tau(t_s,t_e) \\ &+ \frac{\partial^2}{\partial r_v^2}(r^{float}+s)\tau(t_s,t_e)AI_{i-1} \\ &+ \frac{\partial}{\partial r_v}(r^{float}+s)\tau(t_s,t_e)\frac{\partial}{\partial r_v}(AI_{i-1}) \\ &+ \frac{\partial}{\partial r_v}(r^{float}+s)\tau(t_s,t_e)\frac{\partial}{\partial r_v}(AI_{i-1}) \\ &+ (r^{float}+s)\tau(t_s,t_e)\frac{\partial^2}{\partial r_v^2}(AI_{i-1}) \end{aligned} \tag{27}$$

We have still to consider the sensitivity to the discount curve. We consider that the PV expression is:

$$PV = (AI)p(Z_d,t_p) \tag{28}$$

The PV is sensitive to the zero rate r_p at the payment date t_p. Therefore:

$$\frac{\partial}{\partial r_p}(PV) = \frac{\partial}{\partial r_p}(AI)p(Z_d,t_p) + AI\frac{\partial}{\partial r_p}p(Z_d,t_p) \tag{29}$$

And second order derivative, gamma, is given by:

$$\begin{aligned}\frac{\partial^2}{\partial r_p^2}(PV) = &\frac{\partial^2}{\partial r_p^2}(AI)p(Z_d,t_p) \\ &+ \frac{\partial}{\partial r_p}(AI)\frac{\partial}{\partial r_p}p(Z_d,t_p) \\ &+ \frac{\partial}{\partial r_p}(AI)\frac{\partial}{\partial r_p}p(Z_d,t_p) \\ &+ AI\frac{\partial^2}{\partial r_p^2}p(Z_d,t_p) \end{aligned} \tag{30}$$

4.6 Zero Curves

QLIB uses zero curves from SwapClear. SwapClear publishes reports 79 and 100 for IM and VM curves respectively.

Zero curves are linearly interpolated. Changing the interpolation method is not trivial as valuation models, margin methodology all rely upon the linear interpolation.

The ZeroCurveService interface defines the main access point for ZeroCurves. This service uses a mapping file to lookup curves based upon supplied parameters. The SwapClear implementation uses a curve mapping file that is based upon Murex configuration.

Of course curves can also be instantiated programmatically.

4.7 BootStrapping Zero Curves

QLIB implements a bootstrapper that generates zero curves from par rates. The par instruments supported are Deposit Rates, Short Term IR Futures, FRAs and Swaps. Please refer to the BootStrapper User Guide for operational instructions.

The bootstrapper algorithm works by converting all the par instrument pricing functions to linear form involving constant terms and exponential terms. An error term is introduced in the exponential, this term defines the difference between the ideal zero curve and the initial guess curve made up of the par rates. The bootstrapper uses linear algebra to solve the system of linear equations, where the unknowns are the error terms. The solver iterates after correcting the guess curve using the error terms and eventually settles down to the desired curve.

Multiple curves can be simultaneously constructed.

For the theoretical background please refer to the document Bootstrap by Christer Rydberg.

The linearisation of the pricing formulas is further described below.

4.7.1 Floating Cashflow Example

As discussed before, the floating cashflow for a non-compounding swap can be described by the equation:

$$PV = Nr^{float}\tau(t_s, t_e)p(Z_d, t_p) \tag{31}$$

This equation can be expanded to the following form that has constant terms and discount factor terms:

$$\begin{aligned} PV = & \frac{N\tau(t_s, t_e)}{\tau_x(t_v, t_m)}p(Z_f, t_v)p(Z_f, t_m)^{-1}p(Z_d, t_p) \\ & - \frac{N\tau(t_s, t_e)}{\tau_x(t_v, t_m)}p(Z_d, t_p) \end{aligned} \tag{32}$$

Replacing each discount factor term by:

$$p(Z, t) = exp(-r(Z, t)t) \tag{33}$$

the resulting equation looks like this:

$$\begin{aligned} PV = & \frac{N\tau(t_s, t_e)}{\tau_x(t_v, t_m)}exp(-r(Z_f, t_v)t_v) \\ & exp(r(Z_f, t_m)t_m)exp(-r(Z_d, t_p)t_p) \\ & - \frac{N\tau(t_s, t_e)}{\tau_x(t_v, t_m)}exp(-r(Z_d, t_p)t_p) \end{aligned} \tag{34}$$

Above equation is of the general form:

$$C_0 = \overbrace{\underbrace{C_1 e^{r_1 C_2}}_{\text{factor}}\underbrace{e^{r_2 C_3}}_{\text{factor}}\underbrace{e^{r_3 C_4}}_{\text{factor}}}^{\text{term}} + \overbrace{\underbrace{C_5 e^{r_4 C_6}}_{\text{factor}}}^{\text{term}} \tag{35}$$

4.7.2 Linearisation

We now show how above equation can be expressed in a linear form with the error term introduced, making it suitable for solving in a system of linear equations.

To keep the discussion general, let us assume we have two terms in the equation:

$$F_1 = ae^{r_1 s_1} e^{r_2 s_2} \tag{36}$$

$$F_2 = ce^{r_3 s_3} e^{r_4 s_4} \tag{37}$$

where a, b, c, and s_i are constants, and r_i are rates. Let

$$ER = F_1 + F_2 \tag{38}$$

where ER is the expected result, and is a constant. Therefore:

$$ER = ae^{r_1 s_1} e^{r_2 s_2} + ce^{r_3 s_3} e^{r_4 s_4} \tag{39}$$

Introducing an error term d_i as the difference between the ideal rate and the guess rate, we get:

$$ER = ae^{(r_1+d_1)s_1} e^{(r_2+d_2)s_2} + ce^{(r_3+d_3)s_3} e^{(r_4+d_4)s_4} \tag{40}$$

$$ER = ae^{(s_1 r_1 + s_1 d_1)} e^{(s_2 r_2 + s_2 d_2)} + ce^{(s_3 r_3 + s_3 d_3)} e^{(s_4 r_4 + s_4 d_4)} \tag{41}$$

$$ER = ae^{s_1 r_1} e^{s_1 d_1} e^{s_2 r_2} e^{s_2 d_2} + ce^{s_3 r_3} e^{s_3 d_3} e^{s_4 r_4} e^{s_4 d_4} \tag{42}$$

$$ER = F_1 e^{s_1 d_1} e^{s_2 d_2} + F_2 e^{s_3 d_3} e^{s_4 d_4} \tag{43}$$

Following step approximates exp() to linear expression (using Taylor expansion) by assuming that the error d_i is very close to zero:

$$\begin{aligned} e^{s_1 d_1} e^{s_2 d_2} &\approx (1 + s_1 d_1)(1 + s_2 d_2) \\ &= 1 + s_1 d_1 + s_2 d_2 + s_1 d_1 s_2 d_2 \\ &\approx 1 + s_1 d_1 + s_2 d_2 \end{aligned} \tag{44}$$

Substituting the approximation of exp we get:

$$ER = F_1(1 + s_1 d_1 + s_2 d_2) + F_2(1 + s_3 d_3 + s_4 d_4) \tag{45}$$

$$ER - (F_1 + F_2) = F_1 s_1 d_1 + F_1 s_2 d_2 + F_2 s_3 d_3 + F_2 s_4 d_4 \tag{46}$$

Above is a linear equation where the d_i terms are the unknowns, and the rest are constants. By converting the PV expression of each instrument in this way, we get a system of linear equations where the unknowns are the error terms that we can use to correct the rates. We solve the system using matrix operations, apply corrections to the rates using the error terms, and iterate the process until the error term becomes very small.