OPEN RISK ENGINE

Peter Caspers, Niall O'Sullivan, Roland Lichters

Quaternion Risk Management

QuantLib User Meeting, London, 12 July 2016

AGENDA

Open Risk Engine Summary

QuantLib Extension Library

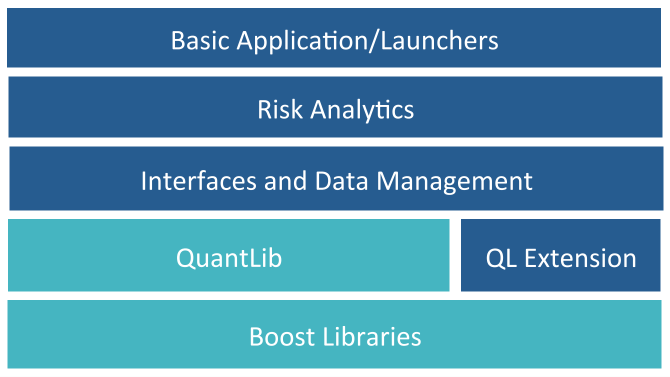
Data Library

Analytics Library

OPEN RISK ENGINE SUMMARY

OPEN RISK ENGINE ...

1. is a transparent platform for pricing and risk analysis, serves as
   - benchmarking, validation, training, teaching reference
   - extensible foundation for tailored risk solutions
2. extends QuantLib (simulation models, instruments, engines)
3. adds contemporary risk analytics and value adjustments
4. adds simple interfaces for trade/market data and system config
5. adds simple launchers in Excel, LibreOffice, Python, Jupyter
6. is free/open software, provided under the Modified BSD License
7. is sponsored by Quaternion Risk Management (www.quaternion.com).


Basic Application/Launchers
Risk Analytics
Interfaces and Data Management
QuantLib
QL Extension
Boost Libraries

TIMELINE

Milestones

- Beta Release: 11 July 2016
- Release: September 2016

ORE provides

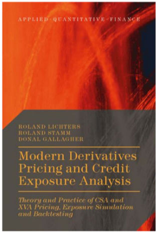
1. Portfolio pricing, cash flows, sensitivity analysis, stress testing
2. Derivative portfolio analytics based on Monte Carlo simulation
   - Credit exposure evolution taking netting and collateral into account (EE, EPE, EEPE, PFE) supporting regulatory capital charge calculation under internal model methods
   - Market risk measures (VaR, ES)
   - Derivative value adjustments (CVA, DVA, FVA, COLVA)
3. Parametric (non-simulation) analytics for risk and capital
   - Initial Margin methods to benchmark ISDA’s SIMM
   - SA-CCR, the new standard method for derivatives capital

PRODUCTS

ORE's initial product scope comprises Interest Rate and FX products

- Interest Rate Swaps
- Caps/Floors
- Swaptions
- Cross Currency Swaps
- FX Forwards
- FX Options

The simulation models applied in ORE are based on:
Modern Derivatives Pricing and Credit Exposure Analysis
Palgrave MacMillan 2015



SCOPE

ORE comes with extensive tests, examples and documentation

- Test suites with good coverage from the start
- Various examples which demonstrate typical use cases
- Several ways to launch ORE and visualise results
- A detailed user guide covering examples and parametrisation
- Comprehensive source code documentation

ROADMAP

A series of further releases is scheduled, covering:

- Sensitivity analysis, stress testing capability
- Credit simulation, Credit Derivatives and Loan products
- Default risk modeling and credit portfolio analysis
- Inflation simulation and Inflation Derivatives
- Equity simulation, Equity Derivatives
- Commodity simulation, Commodity Derivatives
- Open Risk Engine Book

Connect with ORE

- Follow us on Twitter @OpenRiskEngine
- Watch announcements on www.openriskengine.org

Once released:

- Fetch ORE from github.com/openriskengine
- Use ORE
- Share feedback
- Send pull requests

QUANTLIB EXTENSION LIBRARY

QUANTEXT - QUANTLIB EXTENSION LIBRARY

QuantExt adds supplementary building blocks to QuantLib

- a cross asset model and associated pricing engines
- rate helpers for bootstrapping cross currency and tenor basis curves
- a few instruments like currency swaps, basis swaps and average OIS swaps
- additional currencies and indexes

STRUCTURE

The directory structure is like in QuantLib

```
QuantExt / qle  / cashflows
                  currencies
                  indexes
                  instruments
                  math
                  methods
                  models
                  pricingengines
                  processes
                  quotes
                  termstructures
QuantExt / test /
```

Library	Files	Lines of Code	Unit Test Cases
QuantLib	~ 2400	360k	646
QuantExt	~ 200	20k	36
OpenRiskEngineData	~ 160	20k	20
OpenRiskEngineAnalytics	~ 60	7k	21
Sum	~ 420	47k	77

QuantExt provides an implementation of a cross asset model

- multi-Gaussian IR-FX (-INF-CR-EQ-COM)[1]
- exact discretization of the underlying stochastic process for large step simulations
- utilizing Joshi's Sobol Brownian bridge generator provided in QuantLib's market model implementation
- analytic vanilla option engines for fast calibration
- extensible - other models can be plugged in (Heston, multifactor LGM, stochastic basis models, ...)

[1] INF, CR, EQ, COM will be part of later releases

TEST SUITE

Extensive test suite, e.g. for the model part

- consistency with finite difference and Gaussian1D pricing engines in QuantLib
- recovery of analytical moments by Euler Monte Carlo
- martingale property of deflated payoffs
- repricing of calibration baskets with Monte Carlo

QuantLib 1.8 can be used for efficient XVA simulations

- no modifications in QuantLib necessary - this is fantastic
- but we use workarounds at some places, which are efficient in practice, but not clean
- in the following we derive proposals for future QuantLib development from this

PROPOSAL #1 FLOATING TERMSTRUCTURES

We make extensive use of evaluation date shifts during simulation

- provide floating and fixed reference date term structures consistently throughout the library
- expose `TermStructure::moving_` to make fixed and floating term structures distinguishable during run time[2]
- add floating lags for NPV and settlement date parameters in pricing engines, for example and notably in the `DiscountingSwapEngine`
- provide fixed and floating bootstrap helpers

---

[2] note that in addition there are the term structures that manage their reference date themselves

PROPOSAL #2 QUOTES

Quotes are the central tool to apply scenarios to term structures during simulation

- support quotes in `ExchangeRateManager`
- provide quote based constructors in term structures consistently

PROPOSAL #3 OBSERVABILITY

Observability is used to propagate quote updates to term structures and instruments during simulation

- a naive use yields correct results, but may be slow
- deferral of notifications[3] does not seem to speed up our simulation or even slows it down in cases
- our workarounds are
  - disable Notifications and manually update term structures and instruments
  - unregister coupons from evaluation date observation
- goal: can we tape the notification graph on a small subset of simulation paths (or one path) and derive a minimal set of objects that needs to be updated from that?

[3] introduced in QuantLib 1.8

PROPOSAL #4 SIMULATED FIXINGS

During simulation, future fixings have to be generated and published

- required fixings are implicitly known from pricing on the original evaluation date.
- no global notification of all observers[4] necessary when adding a simulated fixing
- pathwise generation of future fixings and publishing them can be automated by an extension of `Index`
- no need for changes in pricing engines
- (almost) zero overhead when simulated fixings mode is disabled

[4]typically floating rate coupons

DATA LIBRARY

DATA LIBRARY

- OpenRiskEngineData is a C++11 library that manages market and trade data
- Configured via API or XML (using RapidXML)
- Flexible curve bootstrap can be configured for Libor, OIS, XOIS, etc leaving the choice to users
- Curve configuration defined for all market curves (option surfaces/cubes) which maps to QL TermStructures
- Lightweight portfolio data model
- Again trade XML maps to QL Instruments

XML EXAMPLE

```xml
<Trade id=”123456”>
 <TradeType>Swap</TradeType>
 <Envelope>
  <CounterParty>CP_A</CounterParty>
  <NettingSetId>CP_A_NS_1</NettingSetId>
 </Envelope>
 <SwapData>
  <LegData>
   <LegType>Fixed</LegType>
   <Payer>true</Payer>
   <Currency>EUR</Currency>
   <Notionals>
    <Notional>70000000</Notional>
   </Notionals>
   <DayCounter>30/360</DayCounter>
   <PaymentConvention>
    Following
   </PaymentConvention>
   <FixedLegData>
    <Rates>
     <Rate>0.035000</Rate>
    </Rates>
   </FixedLegData>
   <ScheduleData>
    <Rules>
     <StartDate>20120530</StartDate>
     <EndDate>20160704</EndDate>
     <Tenor>1Y</Tenor>
     <Calendar>TARGET</Calendar>
     <Convention>
      Following
     </Convention>
     <TermConvention>
      Following
     </TermConvention>
     <Rule>Forward</Rule>
     <EndOfMonth/>
     <FirstDate>20120704</FirstDate>
     <LastDate/>
    </Rules>
   </ScheduleData>
  </LegData>
 </SwapData>
</Trade>
```

- Interface `openriskengine::data::Market` defines a complete set of all the market instruments and curves (as Handles to QL objects) needed for pricing
-
```
class Market {
  //...
  virtual Handle<YieldTermStructure> discountCurve(const string& ccy) = 0;
  virtual Handle<IborIndex> iborIndex(const string& indexName) = 0
  virtual Handle<Quote> fxSpot(const string& ccypair) = 0;
}
```
- `TodaysMarket` implements this interface using curves bootstrapped as on previous slide
- `openriskengine::data::EngineFactory` takes a Market and generates QuantLib::PricingEngines for the portfolio (Actual engine choice and parameters are configurable via API/XML)
- TodaysMarket + EngineFactory + Portfolio = T0 pricing

ANALYTICS LIBRARY

- OpenRiskEngineAnalytics is a smaller library built on top of QuantLib, QuantExt and OpenRiskEngineData.
- Provides a framework for Monte-Carlo simulation of future NPVs, aggregation and (in the future) market risk sensitivities.
- We use the following common definitions:
  - **DateGrid** a set of future dates we wish to calculate exposure on
  - **Cube** the 3-D matrix of trade NPVs for the portfolio on each path and each date in the date grid
  - **Scenario** A set of simulated market data points represented as a set of `QuantLib::Real` values
  - **ScenarioGenerator** A class that combines a model, date grid and PRNG to generate Scenarios.
- Scenarios can be generated by a CrossAssetModel, a Real world model or a set of defined sensitivities.

SCENARIOSIMMARKET

- `analytics::ScenarioSimMarket` is a concrete implementation of the `data::Market` interface that is `Quote` based.
- The method `ScenarioSimMarket::update()` retrieves a Scenario from a ScenarioGenerator and updates the underlying Quotes.
- When a portfolio's `EngineFactory` uses this Market, then all Instruments will be directly linked to the Market's `TermStrutures` and `Quotes`
- Therefore, to price under a scenario we simply call `update()` and then loop over the portfolio calling `Instrument::NPV()`
- This relies heavily on QuantLib's Lazy Object and Observer patterns.

LOOP ORDER

- To compute a Cube, we essentially have three nested loops
- Innermost loop is over portfolio ⇒ two options remain

- Option 1 - Outer Loop over Dates, then Paths

  Pro Minimise date changes - Rebuild static TermStructures at each date and so can use both floating or fixed reference dates.

  Con Need to cache scenarios, creates a memory constraint on the number of paths we can run

  Con Fixings are difficult to do properly

  Con Need to maintain state for path dependant trades

- Option 2 - Outer Loop over Path

  Pro Can price on a path and maintain fixings easily

  Pro Can stream scenarios, no memory constraints

  Con All TermStructures must have a floating reference date

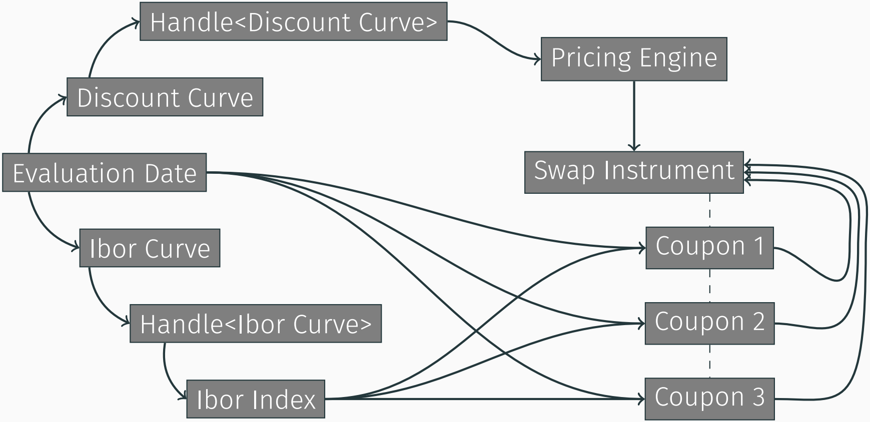
  Con Need to do multiple asof date changes, not cheap

OBSERVATIONS

- `Settings::evaluationDate` is observed a lot.
- Analysis from an early version of OpenRiskEngine:
  - 100 Fixed vs. Floating Swaps, Average maturity = 16.2 years
  - Total of 3,778 Floating Rate Coupons
  - Single call to `Settings::instance().evaluationDate() = d;` takes 1,500 microseconds.
  - 1,000 samples and 80 dates $\Rightarrow$ 120 seconds.
  - Update time is all notification, does not change even at later grid dates when trades are expired.
  - `evaluationDate` is observed by 4,915 observers.
  - Total number of notifications is 34,848
  - Each notification is fast (we are doing 24 per microsecond).
  - However total number is massive (over 2.7 Billion)
  - Deepest chain is of depth 6

OBSERVATION - SINGLE SWAP

- There is a lot of overlap in the notification chain.
- Consider a simple 3 coupon swap.



- Swap is notified 7 times $(2n + 1)$ of a change in the eval date

OBSERVATION - SOLUTIONS

All of the following solutions are available in OpenRiskEngine

1. Do nothing.
    - everything is working as designed and all values are correct
    - it can be slow
2. Minimise notification chain
    - Reduce the notification chain by careful selection or implementation of market data objects
    - remove duplication by unregistering connected observers with common observables (e.g. floating rate coupon and index)
3. Disable all notifications
    - Use `ObservableSettings::disableUpdates(false);`
    - Disable notifications and maintain a separate list of observers that require explicit notification
    - Notification still preformed, but the large chains do not kick in.
4. Defer all notifications
    - Use `ObservableSettings::disableUpdates(true);`
    - Defer notifications until all market quotes and fixings have been updated. This is generally slower than (1)!

- Swap exposure benchmark test runs a portfolio of vanilla swaps over 1,000 samples and 80 dates.
- Cube generation time in seconds

Mode	1 Swap	100 Swaps
None	12.64	320.99
Minimise	12.46	227.91
Disable	11.4	229.45
Defer	13.32	349.07

Questions?