QuantLib

A free/open-source library for quantitative finance

R/QuantLib Integration

Klaus Spanderen, R/Finance 2013

The QuantLib Project
Overview

- A free/open source library for quantitative finance under a very liberal (modified) BSD license.
- Project was started of in 2000 by Ferdinando Ametrano and Luigi Ballabio and is supported by the Italian consultancy StatPro.
- QuantLib has 150+ contributors and is still growing.
- SLOCCount: ~320T source lines of code.

The QuantLib Project
Overview

- QuantLib is written in C++ in a rigorous object oriented design.
- QuantLib comes with over 500 detailed units tests using the Boost unit test framework.
- Luigi Ballabio documentation *"Implementing QuantLib"* is available: https://sites.google.com/site/luigiballabio/qlbook
- Dimitri Reiswich provided the slides for his course "*QuantLib Introduction*": http://quantlib.org/docs.shtml

The QuantLib Project
Key Components

- Asset classes: Interest Rates, Equities, FX, Inflation, Credit Derivatives and Commodities
- Models: (snippet)
  - Interest Rates: Libor Market Model, Markov functional, SABR, bootstrapping a multi-curve set-up
  - Equity: Stochastic volatility with jump diffusion, local volatility
  - ...
- Methods: (snippet)
  - Monte-Carlo Framework, including Quasi-Monte-Carlo algorithms
  - Multi-dimensional finite difference framework
  - Trees
- Calendars, day counters, math tools, design patterns,...

The QuantLib Project
Interoperability

- Interoperability with other languages and spread sheets was an important aspect right from the beginning.
- QuantLibAddin exports a QuantLib interface to Excel and other end-user applications.
- A subset of QuantLib can also be called from C#, Java/Scala, Python, Perl, Ocaml, ... and R.

SWIG-Interface Layer
Overview

- The Simplified Wrapper and Interface Generator (SWIG) connects C/C++ libraries to many other languages.
- The developer has to provide an interface file for the classes and functions to be exported.
- The SWIG tool will generate the glue code between C++ and the target languages.

SWIG-Interface Layer

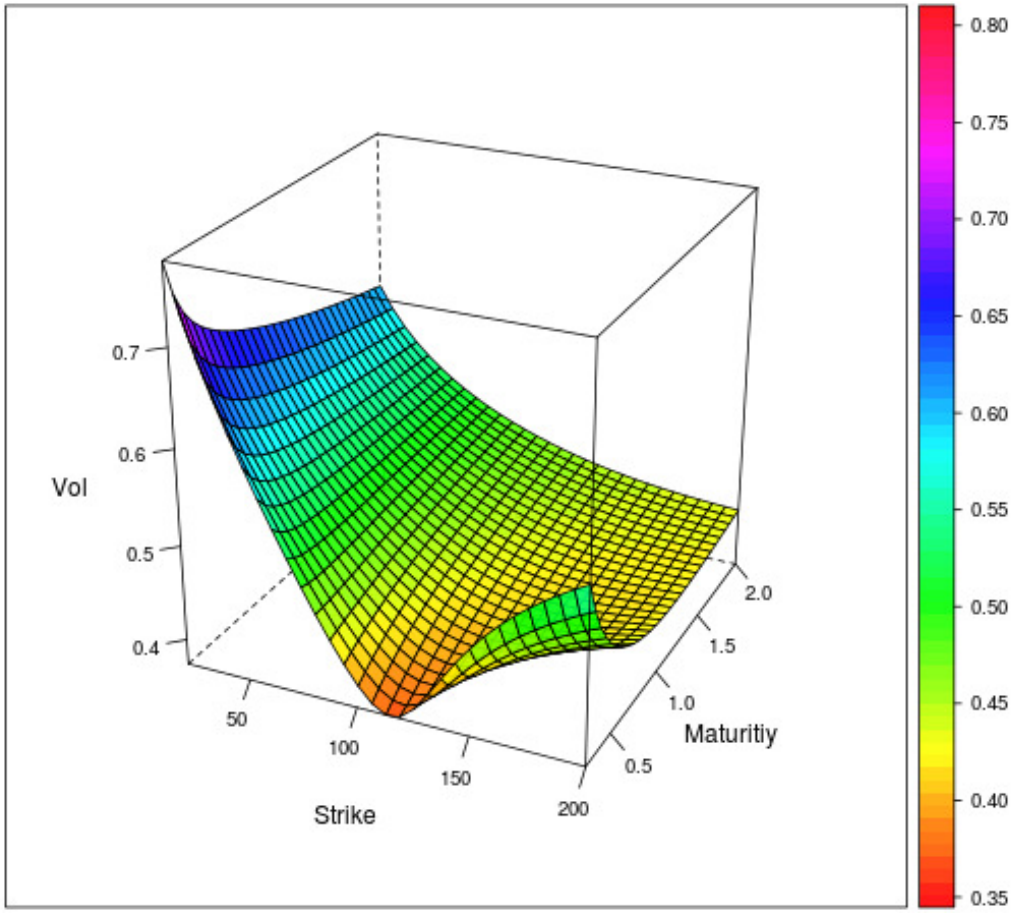
Example: Bates Stochastic Volatility Model (snipplet)

```
# Pricing engine for the Bates stochastic volatility model with jumps
batesEngine <- BatesEngine(BatesModel(BatesProcess(
  riskFreeRate, dividendYield, underlying,
  0.1, 1.5, 0.25, 0.75, -0.75, 0.75, -0.05, 0.3)), 128)

# calculate Black-Scholes implied volatility from the Bates model price
impliedVol <- function(strike, maturity) {
  # option exercise
  exercise <- EuropeanExercise(maturity)

  # define payoff object
  payoff <- PlainVanillaPayoff("Call", strike)

  # plain vanilla european option with given maturity and strike
  option <- VanillaOption(payoff, exercise)

  # option should be priced using the Bates pricing engine
  option$setPricingEngine(option, batesEngine)

  option$impliedVolatility(option, targetValue=option$NPV(),
                           process=bsProcess, accuracy=1e-16,
                           maxEvaluations=100, minVol=0.1, maxVol=2.0)
}
```

SWIG-Interface Layer

Example: Bates Stochastic Volatility Model

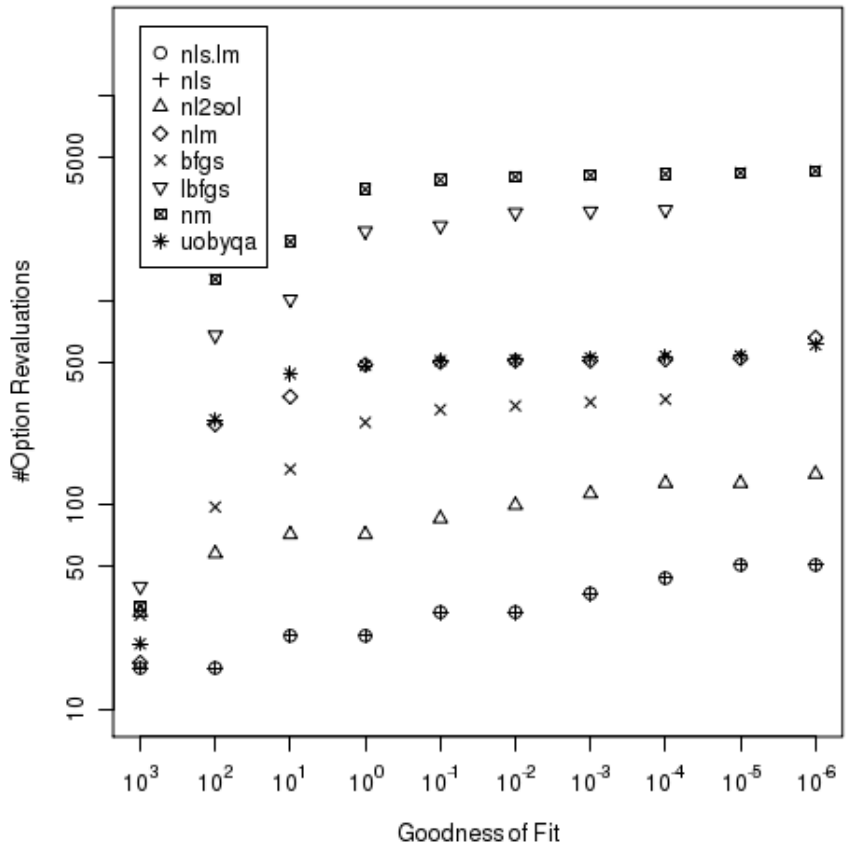
$r = 1\%, q = 4\%, v_0 = 0.1, \kappa = 1.5, \Theta = 0.25, \sigma = 0.75, \rho = -0.75, \lambda = 0.75, \mu = -0.05, \delta = 0.3$



SWIG-Interface Layer
Calibration of Stochastic Volatility Models

- Case study: Choose the best local optimizer for the calibration of a stochastic volatility model.

- **nls.lm**: Levenberg-Marquardt algorithm
- **nls**: Gauss-Newton algorithm
- **nl2sol**:based on the PORT library.
- **nlm**:Newton style minizer
- **bfgs:** quasi-Newton method
- **l-bfgs-b**: limited memory BFGS algorithm
- **cg**: conjugate gradient algorithm
- **nm**: Nelder-Mead method
- **bobyqa**:trust region method
- **newuoa**: trust region method
- **uobyqa**:trust region method



QuantLib Integration in R via Rcpp
Overview

- Rcpp (Dirk Eddelbuettel and Romain Francois) simplifies the integration of C++ code into R.
- Rcpp maps various types of R objects to the corresponding C++ classes.
- Data exchange between R and C++ is simple, flexible and supports STL-like idioms.
- The "inline" package compiles, links and loads a C/C++ function directly from the R-REPL.

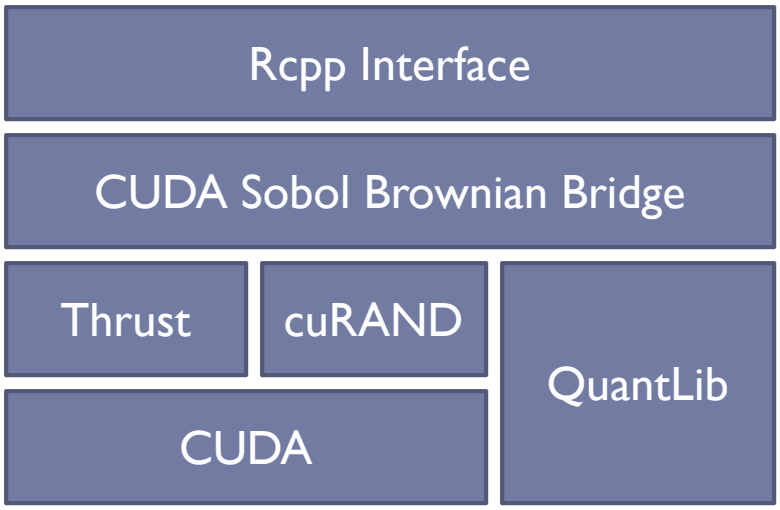
QuantLib Integration in R via Rcpp

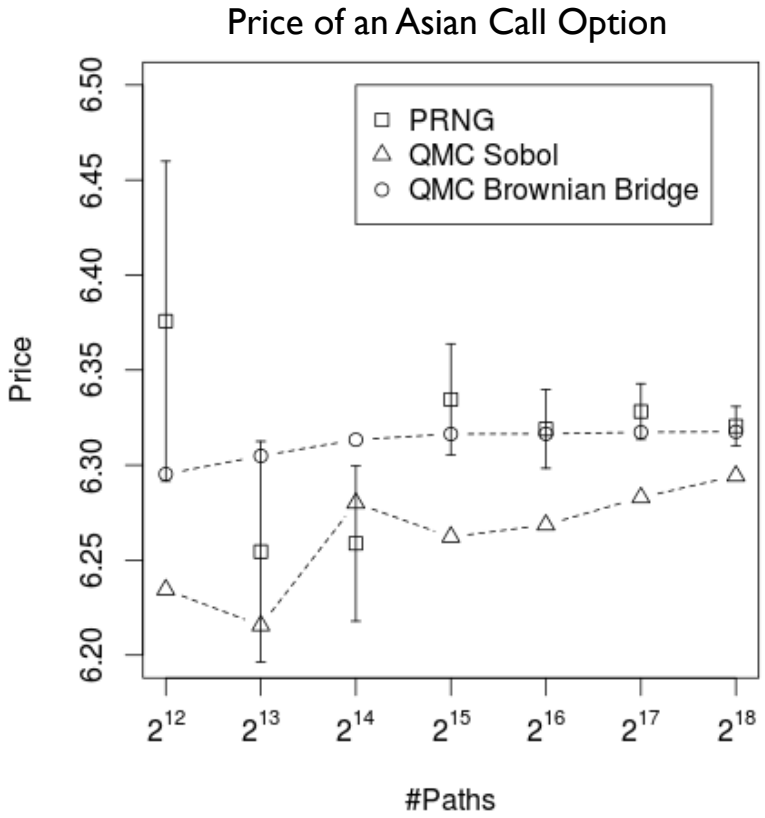
Example: QuantLib's Black Formula

```
library(Rcpp)
library(inline)

# register QuantLib plugin with include helper and linker information
registerPlugin("QuantLib",
  Rcpp:::Rcpp.plugin.maker(include.before='#include <ql/quantlib.hpp>',
                           libs="-lQuantLib"))

# define new R function, which invokes the QuantLib Black formula
blackFormulaCall <- cxxfunction(
  signature(strike="numeric", fwd="numeric",
            stdDev="numeric", discount="numeric"),
  body='
    // C++ body
    using namespace QuantLib;
    return wrap(blackFormula(           // C++ to R using Rcpp::wrap
      Option::Call,
      as<Real>(strike), as<Real>(fwd), // R to C++ using Rcpp::as<>
      as<Real>(stdDev), as<Real>(discount)));',

  plugin="QuantLib")
```

QuantLib Integration in R via Rcpp
Quasi-Monte-Carlo with Brownian Bridges

- Brownian Bridges are a useful tool to use Quasi-Monte-Carlo methods for high dimensional problems.
  - Mark Joshi: More Mathematical Finance





QuantLib Integration in R via Rcpp

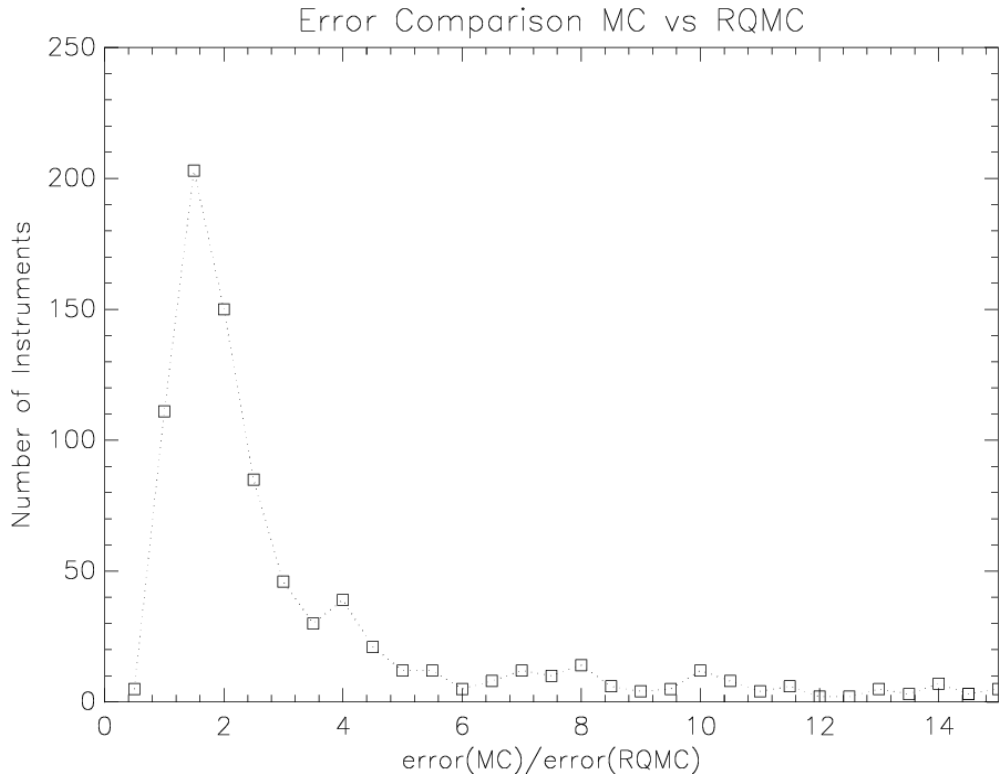
Quasi-Monte-Carlo with Brownian Bridges

```
#include <Rcpp.h>
#include <gpubrownianbridge.hpp>

using namespace QuantLib;

RcppExport SEXP sobolBrownianBridge(SEXP steps, SEXP factors, SEXP paths)
{
    const Size nSteps   = Rcpp::as<Size>(steps)
    const Size nFactors = Rcpp::as<Size>(factors);
    const Size nPaths   = Rcpp::as<Size>(paths);

    // CUDA implementation of a Sobol Brownian Bridge algorithm
    const std::vector<Real> results
        = GPUBrownianBridge(nFactors, nSteps).getPaths(nPaths);

    // prepare Quasi-Monte-Carlo paths as a R matrix
    Rcpp::NumericMatrix m(nPaths, nSteps*nFactors);
    std::copy(results.begin(), results.end(), m.begin());

    return m;
}
```

```
path <- .Call("sobolBrownianBridge", steps=365, factors=1, paths=1023)
```

QuantLib Integration in R via Rcpp
Quasi-Monte-Carlo with Brownian Bridges

- Error reduction for a real portfolio of 835 structured equity linked notes.



Accelerating Exotic Option Pricing and Model Calibration Using GPUs, Bernemann et al in High Performance Computational Finance (WHPCF), 2010, IEEE Workshop on, pages 1–7, Nov. 2010.

R Integration in QuantLib via RInside
Overview

- RInside makes it easy to embed R code in C++ applications.
- The RInside constructor starts a regular R instance.
- Commands are submitted via C++ strings, which are parsed and evaluated.
- Higher-level abstractions from Rcpp keep it simple to get data in and out from the enclosing C++ application.

R Integration in QuantLib via RInside

Using R as a Payoff Scripting Language (snippet)

```
RInside R(argc, argv); // create an embedded R instance

R.parseEvalQ(
    "payoff <- function(path) {"
    "  max(0, path[length(path)] - strike);" // return call payoff value
    "});

R["strike"] = 100.0; // assign value to R's strike variable

for (Size i=0; i < nSimulations; ++i) {
    // QuantLib path generator
    const path_type& path = generator.next().value;

    // assign path to R's path variable
    R["path"] = std::vector<Real>(path[0].begin(), path[0].end());

    // execute payoff script
    const Real npv = R.parseEval("payoff(path)");

    stat.add(npv * discountFactor);
}

std::cout << stat.mean() << " " << stat.errorEstimate()) << std::endl;
```

R Integration in QuantLib via RInside
RInside and Rcpp

- Calling QuantLib methods from R within a C++ application.

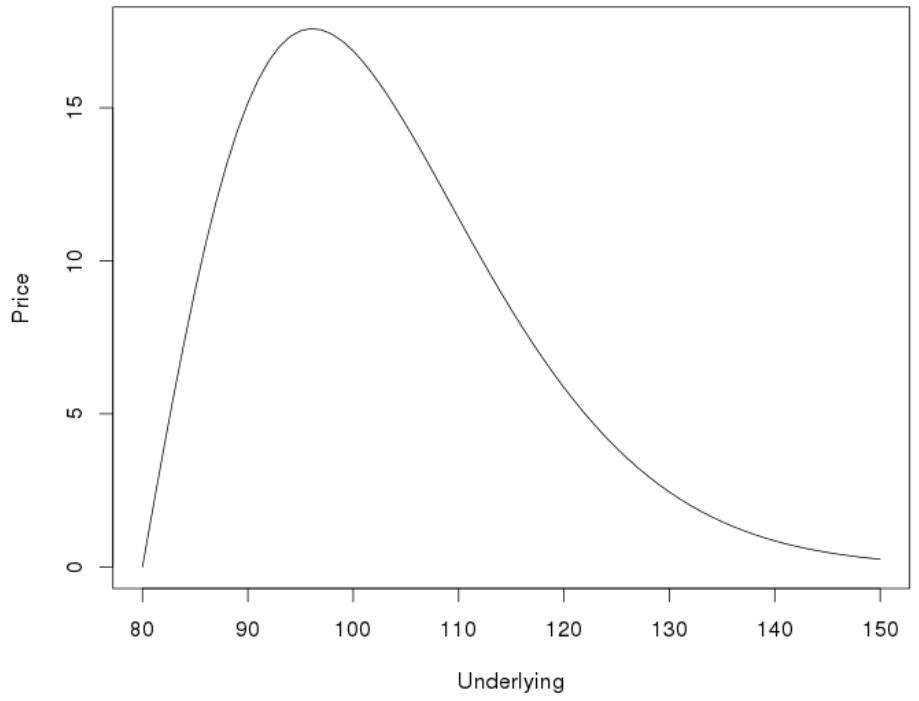
```
#include <RInside.h>
#include <ql/quantlib.hpp>

int main(int argc, char* argv[])
{
    RInside R(argc, argv); // create an embedded R instance

    // assign the QuantLib betaFunction method to R's ql_beta variable
    R["ql_beta"] = Rcpp::InternalFunction( &QuantLib::betaFunction );

    // evaluating the QuantLib beta function via R
    const Real ql_beta = R.parseEval("ql_beta(2, 3)");

    // evaluating R's build-in beta function
    const Real r_beta  = R.parseEval("beta(2, 3)");

    std::cout << r_beta << " " << ql_beta << std::endl;

    return 0;
}
```

The RQuantLib Package
Overview

- The RQuantLib (Dirk Eddelbuettel) package exposes a subset of the QuantLib functionality to R.
- Technically RQuantLib uses Rcpp to bridge between QuantLib's C++ world and R.
- RQuantLib covers a number of option pricing functions, a broader range of fixed income functions and also calendar and holiday utilities.
- RQuantLib is the easiest way to start with QuantLib in R.

The RQuantLib Package
Example: Down-and-Out Put

```
library(RQuantLib)

s <- seq(80, 150)

barrierPrice <- function(s) {
  BarrierOption(
    "downout", "put",
    underlying=s,
    strike=120,
    dividendYield=0.01,
    riskFreeRate=0.03,
    maturity=0.1,
    volatility=0.4,
    barrier=80)$value
}

plot(s, sapply(s,barrierPrice),
     ylab="Price",
     xlab="Underlying",type="l")
```



Conclusion

- R and QuantLib are complementing each other.
- A subset of QuantLib can be accessed by R via the QuantLib-SWIG layer.
- As for any other C/C++ library Rcpp together with RInsinde make it very easy to integrate QuantLib functionality in R or via versa.
- RQuantLib makes parts of QuantLib visible in R using Rcpp.