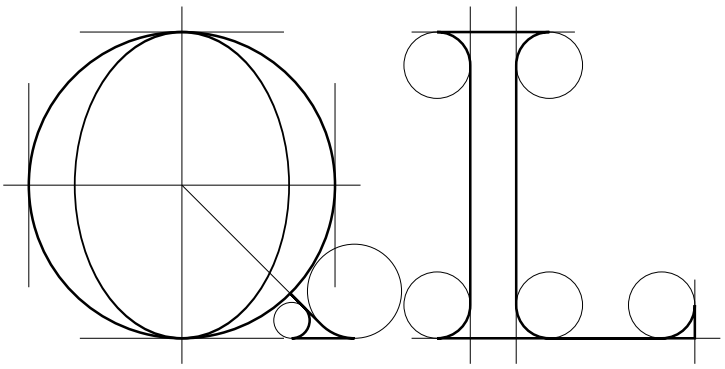


QuantLib

An open source library for quantitative finance

Version 0.3.11

Generated by Doxygen 1.4.4

28 Feb 2006

Contents

Chapter 1

Getting started

1.1 Introduction

QuantLib (http://quantlib.org/) is a C++ library for financial quantitative analysts and developers.

QuantLib is Non-Copylefted Free Software released under the modified BSD License. It is also OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

QuantLib is free software and you are allowed to use, copy, modify, merge, publish, distribute, and/or sell copies of it under the conditions stated in the QuantLib License.

QuantLib and its documentation are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the QuantLib License for more details.

1.1.1 Disclaimer

At this time, this documentation is widely incomplete and must be regarded as a work in progress. Contributions are welcome.

1.2 Project overview

The QuantLib project is at this time in *beta* status.

The following list is a (possibly outdated) overview of the existing code base.

The QuantLib-users and QuantLib-dev mailing lists are the preferred forum for proposals, suggestions and contributions regarding the future development of the library.

Date, calendars, and day count conventions

- Date class.
- Weekday, month, frequency, time unit enumerations.
- Period class (eg. 1y, 30d, 2m, etc.)
- IMM calculation.
- More than 30 business calendars.
- NullCalendar (no holidays) for theoretical calculations.
- Joint calendars made up as holiday union or intersection of base calendars.
- Rolling conventions: Preceding, ModifiedPreceding, Following, ModifiedFollowing, MonthEndReference.
- Schedule class for date stream generation.
- Day count conventions: Actual360, Actual365Fixed, ActualActual (Bond, ISDA, AFB), 30/360 (US, European, Italian), 1/1.

To do:

- Differentiate more calendars depending on country or exchange, instead of city.
- enable business day calculation in addition to calendar days calculation in DayCounter::daycount(). See DayTypeEnum in FpML.

Math

- Linear, log-linear, and cubic spline interpolation.
- Primitive, first and second derivative functions of cubic and linear interpolators.
- Cubic spline end conditions: first derivative value, second derivative value, not-a-knot.
- Monotone cubic spline with Hyman non-restrictive filter.
- Bicubic spline and bilinear interpolations.
- N-dimensional cubic spline interpolation.
- Normal and cumulative normal distributions.
- Inverse cumulative normal distribution: Moro and Acklam approximations.
- Bivariate cumulative normal distribution.
- Binomial coefficients, binomial distribution, cumulative binomial distribution, and Peizer-Pratt inversion (method 2.)

- Chi square and non-central chi square distributions.
- Beta functions.
- Poisson and cumulative Poisson distributions.
- Incomplete gamma functions.
- Gamma distribution.
- Factorials.
- Integration algorithms: segment, trapezoid, mid-point trapezoid, Simpson, Gauss-Kronrod.
- Error function.
- General 1-D statistics: mean, variance, standard deviation, skewness, kurtosis, error estimation, min, max.
- Multi-dimensional (sequence) statistics: all the 1-D methods plus covariance, correlation, L2-discrepancy calculation, etc.
- Risk measures for Gaussian and empirical distributions: semi-variance, regret, percentile, top percentile, value-at-risk, upside potential, shorfall, average shorfall, expected shortfall.
- Array and matrix classes for algebra.
- Singular value decomposition.
- Eigenvalues, eigenvectors for symmetric matrices.
- Cholesky decomposition.
- Schur decomposition.
- Spectral rank-reduced square root, spectral pseudo-square root.

To do:

- Periodic and Lagrange end conditions for cubic spline.
- Implement convexity-preserving filter for cubic spline.
- Log-linear interpolator primitive, first and second derivative functions.
- Revise end conditions for bicubic and N-dimensional spline.
- Trivariate and multi-variate distribution, see Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.
- Hypersphere decomposition, Higham algorithm for pseudo-square root.
- interface with GALib (genetic algorithm)
- Add COOOL algorithms
- Histogram class

1-dimensional solvers

- Bisection, false position, Newton, bounded Newton, Ridder, secant, Brent.

To do:

- Clean up the interface so that it is clear whether the accuracy is specified for x or f(x).

Optimization

- Conjugate gradient, simplex, steepest descent, line search, Armijo line search, least squares.
- Constrained (positive, boundary, etc.) and unconstrained optimization

Random-number generation

- Uniform pseudo-random sequences: Knuth, L'Ecuyer, Mersenne twister.
- Uniform quasi-random (low-discrepancy) sequences: Halton, Faure, Sobol up to dimension 21,200 (8,129,334 if you really want) with unit, Jäckel, Bradley-Fox, and Lemieux-Cieslak-Luttmer initialization numbers.
- Randomized quasi-random sequences (in progress)
- Randomized (shifted) low-discrepancy sequences.
- Primitive polynomials modulo 2 up to dimension 18 (available up to dimension 27)
- Gaussian random numbers from uniform random numbers using different algorithms: central limit theorem, Box-Muller, inverse cumulative (Moro and Acklam algorithms)

Patterns

- Bridge, composite, lazy object, observer/observable, singleton, strategy, visitor.

Finite differences

- Mixed theta, implicit, explicit, and Crank-Nicolson 1-dimensional schemes.
- Differential operators: D_0 , D_+ , D_- , D_+D_- .
- Shout, Bermudan and American exercises.

To do:

- Richardson extrapolation
- Introduce variable theta schemes.
- Introduce multi time-level schemes.
- Enable different solvers (SOR, etc.)
- Extend to time-dependant parameters.
- Extend to local volatility.
- Two-dimension schemes.
- Improve boundary conditions.
- Use DiscretisedAsset instead of array?

- Use TimeGrid
- Use assetGrid
- Handle barrier specification
- Handle variable asset step size
- Check (and improve) vega, rho, dividendRho greeks, solving their own equations

Lattices

- Binomial trees: Cox-Ross-Rubinstein, Jarrow-Rudd, additive equiprobabilities, Trigeorgis, Tian, Leisen-Reimer.
- Trinomial (interest-rate) tree.
- Discretized asset.
- Richardson extrapolation

To do:

- Merge finite differences with the lattice framework. Use same rollback scheme.
- Trinomial trees
- Implied trinomial trees
- Calculate binomial tree greeks

Monte Carlo

- One-factor and multi-factor path classes.
- Path-generator classes: incremental and Brownian-bridge one-factor path generation, incremental multi-factor path generation.
- General-purpose Monte Carlo model based on traits for path samples.
- Antithetic variance-reduction technique.
- Control variate technique.

To do:

- Greeks calculation.
- Allow easier selection between Incremental/BrownianBridge path generation.
- Review Monte Carlo engine for american options.
- Review multi-factor Monte Carlo simulation.
- Predictor-corrector scheme.
- Add Milstein scheme.
- Add Martingale control variate.

- Batch samples as N=n_batches*batch_size, and exploit it for randomized low discrepancy sequences (RQMC)

Pricing engines

- Analytic Black formula (plus greeks) for different payoffs.
- Analytic formula for American-style digital options with payoff at expiry.
- Analytic formula for American-style digital options with payoff at hit.
- Monte Carlo simulation base engine.
- Lattice short rate model base engine.
- Engines for options described by "vanilla" set of parameters: analytic digital American, analytic discrete-dividend European, analytic European, Barone-Adesi and Whaley approximation for American, Ju approximation for American, binomial (Cox-Ross-Rubinstein, Jarrow-Rudd, additive equiprobabilities, Trigeorgis, Tian, Leisen-Reimer), Bjerksund and Stensland approximation for American, integral European, Merton 76 jump-diffusion, Monte Carlo digital, Monte Carlo European, Bates and Heston models, finite-difference European, Bermudan and American.
- Engines for options described by "barrier" set of parameters: analytic down/up in/out, Monte Carlo down/up in/out
- Engines for Asian options: analytic discrete geometric average-price, analytic continuous geometric average-price, Monte Carlo discrete arithmetic average-price, Monte Carlo discrete geometric average-price.
- Engines for options described by "cliquet" set of parameters: analytic, analytic performance.
- Forward and forward-performance compound engines.
- Quanto compound engine.
- Quanto-forward and Quanto-forward-performance compound engines.
- Basket engine: analytic Stulz engine for max/min on two assets, Monte Carlo engine (in progress).
- Black model base class for vanilla interest rate derivatives
- Cap/floor pricing engines: analytic Black model, analytic affine models, tree based engine.
- Swaption pricing engines: analytic Black model, analytic affine models (Jamshidian), tree based engine.

To do:

- Add the trigger level Touch/NoTouch specification for American-style digitals.
- More vanilla engines: Roll-Geske-Whaley American Call, Geske-Johnson American Put, finite differences, Edgeworth expansion binomial tree, etc.
- Merge NesQuant SJD engine (http://www.nielses.dk/quantlib/nesquant/)
- Continuous geometric average-strike.
- Ensure all path-dependent options allow for evaluation with collected past observations.

- Define dividendRho for discrete dividends.

Pricers

- Cliquet option
- Analytic discrete geometric average-price option (European exercise).
- Analytic discrete geometric average-strike option (European exercise).
- Monte Carlo cliquet option.
- Monte Carlo discrete arithmetic average-price option.
- Monte Carlo discrete arithmetic average-strike option.
- Monte Carlo Everest option.
- Monte Carlo Himalaya option.
- Monte Carlo max basket option.
- Monte Carlo pagoda option.
- Monte Carlo forward performance option.

To do:

- Fix finite difference in presence of dividends
- All pricers should be moved to the pricing engine framework.

Financial Instruments

- Instrument base class: npv(), isExpired(), etc.
- Interest-rate swap: simple, normal.
- Swaption.
- Cap/floor.
- Fixed-rate coupon bond.
- Stock.
- One-asset option base class.
- Asian option.
- Barrier option.
- Cliquet option.
- Forward vanilla option.
- Quanto vanilla option.
- Quanto-forward vanilla option.
- Vanilla option.

- Multi-asset option base class.
- Basket option.

To do:

- Forward (stock) and FRA (forward-rate agreement).
- More bonds.

Yield term structures

- Term structure common interface.
- Term structure classes based on discount, zero, or forward underlying description.
- Term structure based on linear interpolation of zero yields.
- Term structure based on log-linear interpolation of discounts.
- Term structure based on constant flat forward.
- Term structure based on piecewise-constant flat forwards with libor-futures-swap bootstrapping algorithm.
- Spreaded term structures.
- Forward-date implied term structure.

To do:

- Future convexity adjustment
- End of year effect

Volatility

- Interface for cap/floor Black volatility term structures (unstable).
- Interface for swaption Black volatility term structures (unstable).
- Interface for equity Black volatility term structures based on volatility or variance underlying description: constant, time-dependant curve, time-strike surface, forward date implied term structure.
- Interface for equity local volatility term structures: constant, time-dependant curve, time-asset level surface (Gatheral's formula).

To do:

- Fix implementation of Gatheral's formula for local volatility.

Short rate models

- Single factor models: Hull-White, Black-Karasinski, Vasichek (untested), CIR (untested), Extended CIR (untested).
- Two factor models: G2 (untested).

Credit derivatives

To do:

- everything.

Test suite

Implemented by means of the Boost unit-test framework. More than 210 automated tests. An automatically-generated list is available in chapter 10.

To do:

- Add covariance/correlation test for SequenceStatistics.
- Increase coverage.

Miscellanea

- Index classes for handling of fixed-income libor indexes (fixings, forecasting, etc.)
- Cash-flow class.
- Currency class and enumeration.
- Money class with automatic exchange-rate capabilities.
- Output data formatters: long integers, Ordinal numerals, power of two, exponential, fixed digit, sequences, dates, etc.
- Input data parsers.
- Error classes and error handling.
- Exercise classes: European, Bermudan, American
- Payoff classes: plain, gap, asset-or-nothing, cash-or-nothing
- Grid classes for handling of equally and unequally spaced grids.
- History class for handling of historical data.
- Quote class for mutable data.
- Null types.
- User-configurable flag to disable usage of deprecated classes.

To do:

- Implement currency as per OMG definition

Documentation

- Documentation automatically generated with Doxygen (html, PDF, ps, WinHelp, man pages)

To do:

- Add a "Getting started" page to the site

1.3 Where to get QuantLib

1.3.1 QuantLib releases

Source code, documentation, modules, etc. of current and previous QuantLib releases can be downloaded from http://quantlib.org/download.shtml

1.3.2 Current CVS snapshot

Instructions for anonymous CVS access are available at http://quantlib.org/cvs.shtml

Access to the CVS repository is intended mainly for developers and is not recommended to end users which should download the latest stable release instead.

1.4 Installation

Before installing QuantLib, make sure that you have a working Boost installation; see http://www.boost.org/more/getting_started.html for instructions. Boost 1.31 or later is required; Boost 1.33 is suggested.

1.4.1 Linux/Unix/Mac OS X/Cygwin

A tarball of the source distribution is available from

http://quantlib.org/download.shtml

After uncompressing the sources:

1. 'cd' to the QuantLib directory and type './configure' to configure the package for your system; see the User configuration section for configuration options.
2. Type 'make' to compile the package.
3. Type 'make install' to install the library. This might require administrative privileges.

1.4.2 Win32

An installer for the source distribution is available at

http://quantlib.org/download.shtml

Before compiling the library, you might want to edit the file "ql/userconfig.hpp"; see the User configuration section for details.

Visual C++ 6.0/7.1 projects files are supplied for building the library.

Dev-C++ project files are provided in order to make easier the usage of Mingw/GCC under Win32.

If you use Borland command line compiler (5.5.1) the make options are: -D_DEBUG (debug), -D_RTLDLL (dynamic linking of runtime library), -D_MT__ (multi-thread) as in:

make all

make -D_DEBUG all

make -D__MT__ -D_RTLDLL -D_DEBUG all

or any other combination of options.

1.5 User configuration

A number of macros is provided for user configuration. Defining or undefining such macros triggers variations in some library functionality.

Under a Linux/Unix system, they are (un)set by `configure`; run

```
./configure --help
```

for a list of corresponding command-line options.

Under a Windows system, they must be (un)defined by editing the file <ql/userconfig.hpp> and commenting or uncommenting the relevant lines.

Such macros include:

```
#define QL_ERROR_FUNCTIONS
```

If defined, function information is added to the error messages thrown by the library. Undefined by default.

```
#define QL_ERROR_LINES
```

If defined, file and line information is added to the error messages thrown by the library. Undefined by default.

```
#define QL_ENABLE_TRACING
```

If enabled, tracing messages might be emitted by the library depending on run-time settings. Enabling this option can degrade performance. Undefined by default.

```
#define QL_NEGATIVE_RATES
```

If defined, negative yield rates are allowed in a few places where they are currently forbidden. It is still not clear whether this is safe. Undefined by default.

```
#define QL_EXTRA_SAFETY_CHECKS
```

If defined, extra run-time checks are added to a few functions. This can prevent their inlining and degrade performance. Undefined by default.

```
#define QL_TODAYS_PAYMENTS
```

If undefined (the default,) payments are considered to be settled at the beginning of the day. Therefore, payments occurring at today's date are not included in the NPV of an instrument.

```
#define QL_DISABLE_DEPRECATED
```

If defined, deprecated code will not be included in the library. Undefined by default.

```
#define QL_USE_INDEXED_COUPON
```

If defined, indexed coupons (see the documentation) are used in floating legs. If undefined (the default), par coupons are used.

```
#define QL_ENABLE_SESSIONS
```

If defined, singletons will return different instances for different sessions. You will have to provide and link with the library a sessionId() function in namespace QuantLib, returning a different session id for each session.

1.6 Usage

To use QuantLib classes in your own code just add

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

at the beginning of your source/header files. Depending on the number of your files in your project, this could cause a large increase in compilation time. If this were not acceptable, collective headers are also available for smaller parts of the library; in particular, each subdirectory of the `ql` directory contains a file `all.hpp` which makes available all classes and function in the respective subdirectory.

Under the Examples folder you can find examples of QuantLib usage, including input files for automake and makefiles for the Borland free compiler and Microsoft Visual C++. For the latter, project files are also available.

1.6.1 Microsoft Visual C++

A few suggestions for Visual C++ users wanting to use QuantLib into their own application:

1. you won't have to explicitly link your application to the QuantLib library. This is done automatically by compiler directives embedded in the sources.
2. Your project must be compiled with the same options that were used in compiling the QuantLib library you're linking with. For VC6 please

a) select the appropriate run-time library under project settings, "C/C++" tab, "Code Generation",

b) check the "Use RTTI" option under the "C++ Language" category.

For VC7 (.NET) under

a) Property Pages -> C/C++ -> Code Generation -> Runtime Library: please select the appropriate run-time library.

b) Property Pages -> C/C++ -> Code Generation -> Basic Runtime Checks: please select "Both (/RTC1, equiv. to /RTCsu)".

1.7 Frequently asked questions

1.7.1 Generic questions

Is it OK to email a QuantLib developer to ask questions, or seek help, or report a bug?

Yes, it is. However, we urge you to consider posting to the QuantLib mailing list instead. This is for two reasons. The first is that messages on the list are stored: the next one with your problem will be able to find the answer by searching the archives. The second is that you might get your answer sooner. For instance, it just so happens that I am writing this entry in the middle of a two-weeks period without an Internet connection. If anybody wrote me last Monday, the poor soul will wait two weeks for an answer which could have been given already by someone else on the list.

However, if your intent in writing was to call the developer names, disregard the above. By all means write personally to the developer. And possibly, add the line:

```
X-Bogosity: Yes
```

to the mail headers, so that our filters—I mean, WE can take immediate care of it.

How should I report a bug?

You can file a bug report using the SourceForge interface at <http://sourceforge.net/tracker/?group_id=12740&atid=112740>, or you could write to a QuantLib mailing list.

In any case please report as much details as possible.

If it is a compilation problem please state at least:

- OS system
- compiler (version number, patch level, etc.)
- Boost version
- the compilation error and the file affected

If the test suite fails please report the output obtained by executing the test suite with the following command line options:

```
--log_level=messages --build_info=yes --result_code=no --report_level=short
```

Thanks for this project. How can I give back to it?

In true open-source fashion, you can contribute code to the project; see the section 'Contributing to the project' below. This is by far the preferred contribution, closely followed by using the library intensively and reporting any bugs you might find—and possibly patches for fixing them.

However, if you made money by using QuantLib and feel that, as Christmas is getting near, you want to give us a token of your gratitude—well, who am I to discourage you? (for instance, < grin > Luigi's wish list on Amazon UK is at <http://www.amazon.co.uk/exec/obidos/registry/2PC411P4U28CG/ref=wl_em_to>, and Nando's is at <http://www.amazon.co.uk/exec/obidos/registry/Q94W7HUR49Z5/ref=wl_em_to>.)

Amazon Wish List? Aren't you ashamed of yourselves?

< broad grin > No, we aren't.

1.7.2 Contributing to the project

I'm interested in getting involved with the project. What should I do?

Contact the QuantLib group by posting to the developers' mailing list (<quantlib-dev@lists.sourceforge.net>) and describe your experience and interests. Before doing this, please read:

- the generic introduction for new developers <http://quantlib.org/newdeveloper.shtml>
- the project overview <http://quantlib.org/reference/overview.html>, with its to-do suggestions
- the Developer FAQ <http://quantlib.org/developerFAQ.shtml>
- The Programming Style Guidelines <http://quantlib.org/style.shtml>
- the detailed low-level to-do list <http://quantlib.org/reference/todo.html>

Also, you might want to specify an area of the library you are particularly interested to, or which would be most useful to you. Asking the administrators to choose a task for you is ok, but it might take time to get an answer and it increases the odds that the chosen task will bore you or otherwise discourage you from completing it.

How do I contribute code to the project?

First of all, make sure that contributing code on your part cannot result in litigation about intellectual property. If you work at some financial institution, ask for permission before contributing any relevant portion of code—and get a statement in print.

As for the mechanics of contribution, the preferred way is to submit a patch to the SourceForge patch tracker at <http://sourceforge.net/tracker/?group_id=12740&atid=312740>. This will make it less likely that your files are forgotten in the depths of a developer's mailbox.

The preferred format is a diff file as created by the 'patch' utility. If possible, send differences against the CVS repository; diff files based on the latest release might not apply to the latest sources.

If 'patch' is not available on your system or you are not familiar with it, submit the modified files. However, keep in mind that integrating such a contribution will require more work and therefore will take longer.

Finally, contributions should be accompanied by one or more test cases checking the functionality of the new code. While this is not a strict requirement, complying with it will buy from the developers a lot more sympathy towards your contribution.

1.7.3 Building QuantLib

I'm having trouble building QuantLib with MinGW.

Terry August was kind enough to put together detailed instructions for MinGW users. They can be found at <http://www.stanford.edu/~taugust/quantlib.html>.

When trying to compile QuantLib in the Dev-C++ IDE, I get an error saying that "Could not create makefile: C:\...\Makefile.win"

The message is misleading. Close the IDE, create an empty sub-directory named "build" in the same directory as the Dev-C++ project, and retry.

Also, a user reported that this was necessary but not sufficient. His workaround was to change the relative paths in Project Options / Build Options to absolute paths (e.g., ".\lib" to "C:\QuantLib-0.3.11\lib").

When building QuantLib, I get a compile error about a missing boost/something header.

As mentioned in the readme, QuantLib depends on the Boost library (http://www.boost.org). You must download and install it before building QuantLib.

When building the test-suite, I encounter a linking error about libboost_unit_test_framework-xxx.

The folder including the Boost libraries in not in your link path. See the documentation of your compiler for instructions on how to add it.

But I have no such library on my machine!

Most likely, you downloaded the Boost distribution and just copied its header files somewhere in your include path. The Boost libraries must be built as well; see http://www.boost.org/more/getting_started.html for instructions.

Ok, now I have the library; and the library path is set correctly. But I still cannot link!

You're using Dev-C++ or MinGW, aren't you? gcc is looking for a library called libboost_unit_test_framework-xxx.a, but the Boost installation process created a libboost_unit_test_framework-xxx.lib instead. Make a copy of the latter in the same location and rename the copy so that it has the correct extension.

When building the test-suite, I encounter a number of compile errors. I'm using gcc and Boost 1.32.

This is a Boost problem; you have to apply a one-line patch to your Boost installation. Open boost/test/detail/wrap_stringstream.hpp and edit line 120,

```
#if !defined(BOOST_NO_STD_LOCALE) && BOOST_WORKAROUND(BOOST_MSVC, >= 1310)
```

so that it reads like:

```
#if !defined(BOOST_NO_STD_LOCALE) && ( !defined(BOOST_MSVC) || BOOST_WORKAROUND(BOOST_MSVC, >= 1310))
```

Also, this problem has been worked around in QuantLib itself since version 0.3.10. You might want to upgrade your QuantLib installation.

1.7.4 Testing QuantLib

The QuantLib test-suite fails when compiling under Mac OS X.

We are aware of the problem; apparently, there are issues with global and/or static variables when using shared libraries. As a workaround, compile QuantLib as a static library. This can be accomplished by running configure as:

```
configure --disable-shared
```

1.7.5 Using QuantLib

When linking QuantLib to my project under Visual C++, I encounter the following linking error:

```
LINK : fatal error LNK1104: cannot open file "QuantLib-vcX-xx-xxx-a_b_c.lib"
```

The folder including QuantLib-vcX-xx-xxx-a_b_c.lib is not in your link path (see Project Settings | Link | Input in VC6 or Property Pages | Linker | Input in VC7) or you haven't really built QuantLib-vcX-xx-xxx-a_b_c.lib yet. Note that each build configuration produces a different library.

Programs linking QuantLib fail to run correctly under Mac OS X.

This is the same problem reported in the 'Testing QuantLib' section; the same workaround applies.

1.7.6 QuantLib features

Why is feature X missing from QuantLib? It would be a very useful one.

See the section 'Contributing to the project' above.

1.7.7 QuantLib mailing lists

How do I prevent my auto-responder from posting to the list while I'm away?

It might be possible to configure the auto-responder so that it ignores posts to the mailing list. However, this depends on the particular software you are using.

If that is not possible, you can temporarily prevent the list from posting to your account. It is a bit more of a hassle, but the other list members will appreciate.

Go to your mailing-list configuration page (its direct address is sent to you monthly by SourceForge; alternatively, go to <http://lists.sourceforge.net/lists/listinfo/quantlib-users> and insert your mail address in the last form in the page.)

From there you can enable the "Disable mail delivery" option, causing the posts to the list not to be forwarded to your account.

The mail delivery can be re-enabled later in the same way; you'll have to check the list archives at <http://sourceforge.net/mailarchive/forum.php?forum=quantlib-users> to catch up on the posts you missed.

1.7.8 QuantLib cross-platform support

Does QuantLib support .NET?

C# has never been officially supported by the QuantLib team. Both <http://www.quantlib.net> and <http://www.capetools.net> have been "external" attempts which now look like abandoned projects. As such nobody but their authors could help you.

1.8 Version history

Release 0.3.11 - October 2005

GLOBAL FEATURES

- Added configuration option for adding current function information to error messages.
- Added hook for multiple sessions to Singleton.

CALENDARS

- Added Bombay and Taipei calendars.

CURRENCIES

- Added new Turkish lira.

INDEXES

- More accurate LIBOR calendars (thanks to Daniele de Francesco.)
- Added DKKLibor, EURLibor, and NZDLibor indexes.
- Added TRLibor index (thanks to Sercan Atalik.)

PRICING ENGINES

- Added Bates stochastic-volatility model; tests provided (thanks to Klaus Spanderen.)
- Added vega to analytic discrete-averaging Asian engine; test provided (thanks to Gary Kennedy.)
- Added stochastic process for caplet Libor market model; tests provided (thanks to Klaus Spanderen.)

TERM STRUCTURES

- Added fixed-coupon bond helper for curve bootstrapping (thanks to Toyin Akin.)

MATH

- Added tabulated Gauss-Legendre quadratures (thanks to Gary Kennedy.)
- Added more precise implementation of bivariate cumulative normal distribution (thanks to Gary Kennedy.)

Release 0.3.10 - July 14th, 2005

GLOBAL FEATURES

- The suggested syntax for setting and registering with the global evaluation date is now:

```
Settings::instance().evaluationDate() = date;
registerWith(Settings::instance().evaluationDate());
```

CALENDARS

- Istanbul calendar added (thanks to Serkan Atalik.)

LATTICE FRAMEWORK

- Faster implementation of binomial and trinomial trees.

MONTE CARLO FRAMEWORK

- Added generic multi-dimensional stochastic process.
- Added stochastic process array (thanks to Klaus Spanderen.)
- Multi-path generator now takes a generic stochastic process; tests provided.
- New Path class implemented which stores asset values rather than variations; this makes pricers independent on whether or not log-variations were calculated. The new class is enabled when QL_DISABLE_DEPRECATED is defined; the old class is used otherwise.

INSTRUMENTS

- Multi-asset option now takes a generic stochastic process.

MODELS

- Added Heston stochastic-volatility model; tests provided (thanks to Klaus Spanderen.) Provided code include:
  - a corresponding stochastic process;
  - analytic and Monte Carlo option-pricing engines;
  - parameter calibration.

CASH FLOWS

- Cash-flow analyses such as NPV, IRR, convexity and duration added (thanks to Charles Whitmore.)

MATH

- Added Gaussian orthogonal polynomials and Gaussian quadratures; tests provided (thanks to Klaus Spanderen.)
- Convergence statistics added; tests provided (thanks to Gary Kennedy.)

Release 0.3.9 - May 2nd, 2005

GLOBAL FEATURES

- QL_SQRT, QL_MIN etc. deprecated in favor of std::sqrt, std::min...
- Added a tentative tracing facility to ease debugging.
- Formatters deprecated in favor of output manipulators. A number of data types can now be sent directly to output streams.

- Stream-based implementation of QL_REQUIRE, QL_TRACE and similar macros. Together with manipulators, this allows one to write simpler error messages, as in:

```
QL_FAIL("forward at date " << d << " is " << io::rate(f));
```

INSTRUMENTS

- Improved Bond class
  - yield-related calculation can be performed with either compounded or continuous compounding;
  - added theoretical price based on discount curve;
  - fixed-rate coupon bonds can define different rates for each coupon;
  - added zero-coupon and floating-rate bonds (thanks to StatPro.)
- Option instruments now take a generic StochasticProcess; however, most pricing engines still require a BlackScholesProcess. They should be checked to see whether the requirement can be relaxed. Following this change, Merton76Process no longer inherits from BlackScholesProcess. This avoids erroneous upcasts.
- Partial fix for Bermudan swaptions with exercise lag (thanks to Luca Berardi for the report and discussion.)
- Fix for analytic cap/floor engine; caplets/floorlets whose fixing is in the past are now calculated correctly (thanks to Aurelien Chanudet.)

CALENDARS

- Added Bratislava and Prague calendars.

INDICES

- Fixed calendars for LIBOR fixings (thanks to Daniele De Francesco.)

FINITE_DIFFERENCES FRAMEWORK

- Migrated finite-difference pricers to pricing-engine framework (thanks to Joseph Wang.)

YIELD TERM STRUCTURES

- Added generic piecewise yield term structure. Client code can choose what to interpolate (discounts, zero yields, forwards) and how (linear, log-linear, flat) by instantiating types such as:

```
PiecewiseYieldCurve<Discount,LogLinear>
PiecewiseYieldCurve<ZeroYield,Linear>
PiecewiseYieldCurve<ForwardRate,Linear>
```

- Interpolated discount, zero-yield and forward-rate curves can now be set any interpolation.
- FlatForward can now take rates with compounding other than continuous.
- Fix for extrapolation in zero-spreaded and forward-spreaded yield term structure (thanks to Adjriou Belak for the report.)

MATH

- Added backward- and forward-flat interpolations.

Release 0.3.8 - December 22nd, 2004

REQUIRED PACKAGES

- Boost version 1.31.0 or later is now required.

DOCUMENTATION

- Documentation now includes a FAQ page.

GLOBAL FEATURES

- Global evaluation date added through Settings class. Used for index-fixing and exchange-rate lookup.
- added InterestRate class, which encapsulate the interest rate compounding algebra. It manages day-counting convention, compounding convention, conversion between different conventions, and discount/compounding factor calculations. It also has its own formatter.

INSTRUMENTS

- Bond and FixedCouponBond classes added (thanks to Jeff Yu) providing price/yield conversions; tests provided.

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

- Reworked Date interface. Added nextWeekday() and nthWeekday() static methods to the class Date. Added nextIMM() for the calculation of the next IMM date.
- Added WeekdayFormatter and FrequencyFormatter
- Added "1/1" day counter. The Actual365 is deprecated: as per ISDA documentation "Actual/365" is the same as "Actual/Actual". Use the ActualActual class instead, or the Actual365Fixed class.
- Added dayCounterFromString(std::string) to QuantLibFunctions.
- Improved Beijing calendar (thanks to Zhou Wu.)

CURRENCIES AND FX RATES

- Added currency classes; CurrencyTag replaced in library code.
- Added money class providing arithmetic with or without conversions; tests provided.
- Added exchange-rate class; tests provided.
- Added exchange-rate manager with smart rate lookup, i.e., able to derive a missing exchange rate as a chain of provided rates; tests provided.

MONTE CARLO FRAMEWORK

- Added Faure low-discrepancy sequence (thanks to Gianni Piolanti;) tests provided.
- Added randomized (shifted) low discrepancy sequences that will be used for randomized quasi Monte Carlo.
- Added SeedGenerator class, for random generation of seeds when they are not given by the user.
- Added the implementation of Sobol sequences using the coefficients of the free direction integers as provided by Bratley and Fox, who credited unpublished work of Sobol's and Levitan's.
- Added an implementation of Sobol sequences using the coefficients of the free direction integers of Lemieux, Cieslak, and Luttmer. Coefficients for d<=40 are the same as in Bradley-Fox. For dimension 40<d<=360 the coefficients have been calculated as optimal values based on the "resolution" criterion. The values has been provided by Christiane Lemieux, private communication, September 2004.
- PathGenerator now works correctly with processes describing S instead of log S. Geometric Brownian process added (thanks to Walter Penschke.)

LATTICE FRAMEWORK

- Reworked the DiscretizedAsset interface.

PRICING ENGINES FRAMEWORK

- Added pricing engine for American options with Ju quadratic approximation.
- Average-price Asian pricers have been deprecated. New equivalent pricing engines added.

FIXED INCOME

- Added current coupon to discretized swap and cap/floor.
- Added IndexManager as a singleton (will replace XiborManager–already obsoleted in library code.)
- Added DayCounter parameter to ParCoupon (to be used for accruing spreads and past fixings.) When missing, it defaults to that of the term structure.
- Added compilation flag to select default floating-coupon type.
- IndexedCoupon can now take a generic index rather than a Libor (thanks to Daniele De Francesco.)
- Added hooks for convexity adjustment in floating-rate coupons; implemented adjustment for InArrearIndexedCoupon.

YIELD TERM STRUCTURE

- TermStructure renamed to YieldTermStructure (the former name was deprecated.)
- New base class BaseTermStructure which can calculate its reference date based on the global evaluation date. YieldTermStructure, BlackVolTermStructure, LocalVolTermStructure, CapFlatVolatilityStructure, CapletForwardVolatilityStructure, and SwaptionVolatilityStructure are now derived from BaseTermStructure so that they inherit its functionality.

PATTERNS

- Added Singleton pattern.

MATH

- Added N-dimensional cubic spline (thanks to Roman Gitlin.)
- Added CovarianceDecomposition class (decomposes a covariance matrix into standard deviations and correlations)

MISCELLANEA

- Renamed RelinkableHandle to Handle.

PORTABILITY

- Support for Dev-C++ IDE added.
- Fixes for gcc 2.95 added (thanks to Michael Dirkmann.)

Release 0.3.7 - July 23rd, 2004

IMPORTANT

QuantLib now depends on the Boost library (www.boost.org).

You will need a working Boost installation in order to compile and use QuantLib. Instructions for installing Boost from sources are available at <http://www.boost.org/more/getting_started.html>. Pre-packaged binaries might be available from other sources. Google is your friend (or Debian, or Fink...)

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

- Working on differentiating calendars depending on country or exchange, instead of city.
- Added Italy (Settlement, Exchange), United Kingdom (Settlement, Exchange, Metals), United States (Settlement, Exchange, GovermentBond), Xetra.
- Milan, London, and NewYork calendars have been deprecated.
- Added (old-style) calendars: Beijing, Hong Kong, Riyadh, Seoul, Singapore, Taiwan.
- RollingConvention has been renamed BusinessDayConvention, as for ISDA definitions.

MATH

- Added rounding algorythms as per OMG enumeration/definition.

TEST SUITE

- Moved to Boost unit test framework. CppUnit is no longer needed.
- Added test for quanto and forward compound engines.
- Added test for roundings.

- Added test for discrete dividend European options.
- Added test for cliquet options.

MISCELLANEA

- enable/disableExtrapolation() methods were added to a few classes such as TermStructure. They make it possible to persistently allow extrapolation without the need of specifying it at every method call.
- Added user-configurable flag to disable usage of deprecated classes.

PORTABILITY

- Fink package available
- Visual C++ 7.x project files added

Release 0.3.6 - April 15th, 2004

Bug-fix release for QuantLib 0.3.5. A bug was removed where calls to impliedVolatility() would break the state of the option and of all options sharing the same stochastic process.

Release 0.3.5 - March 31th, 2004

BOOST SUPPORT

- When available, QuantLib 0.3.5 now uses parts of the Boost library. The presence of Boost is detected automatically under Unix/Linux systems; on Windows systems, it must be enabled by uncommenting the relevant line in ql/userconfig.hpp.
- In the next QuantLib release, the presence of the Boost library will be mandatory.

MONTE CARLO FRAMEWORK

- Modified MultiPath interface to remove drifts. They are now in the stochastic processes.
- Preliminary implementation of Longstaff-Schwartz least-squares
- Monte Carlo pricer for European basket options
- Brownian-bridge bugs fixed
- StochasticProcess base class and derived classes (diffusion, jump-diffusion, etc.) have been created.

PRICING ENGINES FRAMEWORK

- Pricing engines now use Payoff and Exercise classes.
- American basket options.
- Binary barrier option replaced by vanilla option with digital payoff.
- Stulz engine for max and min basket calls and puts on two assets.
- American binary option added (a.k.a. one-touch, american digital, americal barrier, etc.) with different payoffs (cash/asset at hit/expiry, etc.)

- Added engine for Merton 1976 jump-diffusion process.
- Added Bjerksund and Stensland approximation for American option (still unstable.)
- Added Barone-Adesi and Whaley approximation for American option.
- Improved Black formula engine with more greeks added.
- Discrete geometric asian option.
- Added Leisen-Reimer binomial tree.

SHORT RATE MODELS

- Model renamed to ShortRateModel. A typedef is provided for backward compatibility–it will be removed in subsequent releases.

VOLATILITY FRAMEWORK

- bug fix for short time (0<=t<=Tmin) interpolation

OPTIMIZATION FRAMEWORK

- Method renamed to OptimizationMethod. A typedef is provided for backward compatibility–it will be removed in subsequent releases.

PATTERNS

- Composite pattern

MATH

- Improved cubic spline interpolation. It now handles end conditions such as first derivative value, second derivative value, not-a-knot. Hyman filter for monotonically constrained interpolation has been implemented. Primitive calculation has been enabled in addition to derivative and second derivative.
- Primitive, first derivative, and second derivative functions are available for linear interpolator.
- Singular value decomposition improved.
- Added bivariate cumulative normal distribution.
- Added binomial coefficient calculation, binomial distribution, cumulative binomial distribution, and Peizer-Pratt inversion (method 2.)
- Added beta functions.
- Added Poisson distribution and cumulative distribution.
- Added incomplete gamma functions.
- Added factorial calculation.
- Added rank-reduced square root and improved pseudo-square root of square symmetric matrices.

- Added Cholesky decomposition.

TEST SUITE

- Added test for cubic spline interpolation.
- Added test for singular value decomposition.
- Added test for two-asset baskets using the Stulz pricing engine.
- Added test for Monte Carlo American cash-at-hit options.
- Added test for jump-diffusion engine.
- Added test for American and European digital options.

MISCELLANEA

- Inner namespaces have been deprecated.
- Added frequency enumeration, including 'once'.
- MarketElement renamed to Quote.
- Handling strike=0.0 where possible.
- More Payoff classes have been introduced: gap, asset-or-nothing, cash-or-nothing. Payoff is now extensively used.
- Exercise class is now polymorphic. More derived classes have been introduced, and they are now extensively used.
- Introduced QL_FAIL macro.
- Added calendar for Copenhagen
- 14 April 2004 (election day) added to Johannesburg calendar as a one-off holiday.
- Documentation generated with Doxygen 1.3.6.
- Win32 installer generated with NSIS 2.0.

Release 0.3.4 - November 21th, 2003

MONTE CARLO FRAMEWORK

- MC European in one step with strike-independent vol curve (hopefully)
- Path pricer for Binary options. It should cover both European and American style options. Also known as: Digital, Binary, Cash-At-Hit, Cash-At-Expiry.
- Path pricers for barrier options

PRICING ENGINES FRAMEWORK

- More options moved to the new pricing engine framework: binary, barrier
- Changed setupEngine() into setupArguments(args)
- Moved pricing-engine machinery up to Instrument class

FIXED INCOME

- New basis-point sensitivity functions
- Added Swap::startDate() and maturity()
- Cap/floor fixing days taken into account

SHORT RATE MODELS

- An additional constraint can now be passed to the calibration

VOLATILITY FRAMEWORK

- Visitable volatility term structures

OPTIMIZATION FRAMEWORK

- Added composite constraint

PATTERNS

- Visitor, Alexandrescu-style (saves some code duplication)

MATH

- Added more integration algorithms contributed by Roman Gitlin
- Relaxed constaints on interval boundaries for integration algorithms
- Interpolation traits

TEST SUITE

- Added implied cap/floor term volatility test
- Added test for binary options in PricingEngine Framework.
- Added tests for Barrier options in PricingEngine Framework. Some Monte Carlo tests, but not comprehensive.

MISCELLANEA

- Conditionally allowed negative yields (disabled by default)
- Null calendar and simple day counter for reproducing theoretical calculations
- Fixed for VC++.Net compilation
- Added spec file for RPMs
- Added global flag for early/late payments
- Enabled test suite for Borland
- Removed OnTheEdge VC++ configurations

- Added VC++ configurations for static and dynamic Multithread libraries
- Upgraded to use Doxygen 1.3.4

Release 0.3.3 - September 3rd, 2003

MONTE CARLO FRAMEWORK

- Re-templatized Monte Carlo model based on traits.
- New path generator based on DiffusionProcess, TimeGrid, and externally initialized random number generator.
- Added Halton low discrepancy sequence.
- Added sequence generators: random sequence generator creates a sequence generator out of a random number generator. InvCumGaussianRsg creates a gaussian sequence generator out of a uniform (random or low discrepancy) sequence generator.
- RNG as constructor input constructor(long seed) deprecated.
- Mersenne Twister random number generator added
- Old PathPricers, PathGenerators, etc are available with a trailing _old
- Added Jäckel's Brownian Bridge (not used yet.)
- Sobol Random Sequence Generator. Unit and Jäckel.
- Added randomized Halton sequences.

FINITE DIFFERENCE FRAMEWORK

- Old class Grid no longer exists, use CenteredGrid to obtain the same result.

LATTICE FRAMEWORK

- Abstracted discretized option.
- Additive binomial trees. All binomial trees now use DiffusionProcess.
- Added Tian binomial tree.

PRICING ENGINES FRAMEWORK

- Partially implemented.
- Quanto forward compounded engines.
- Integral (european) pricing engine.

YIELD TERM STRUCTURE

- ZeroCurve: a term structure based on linear interpolation of zero yields.

FIXED INCOME

- Up-front and in-arrear indexed coupon.

- Specific implementation of compound forward rate from zero yield.
- Added compound forward and zero coupon implementations.
- Added Futures rate helper with specified maturity date.
- Added bucketed bps calculation.
- Added swap constructor using specified maturity date as well as added functionality in Scheduler.
- Added date-bucketed basis point sensitivity based on 1st derivative of zero coupon rate.

OPTIMIZATION FRAMEWORK

- Solvers now take any function. ObjectiveFunction disappeared.

PATTERNS

- Abstracted lazy object.
- Abstracted the curiously recurring template pattern.

DATE AND CALENDARS

- Added joint calendars.
- Tokyo, Stockholm, Johannesburg calendar improved.
- "MonthEndReference" business day rolling convention. Similar to "ModifiedFollowing", unless where original date is last business day of month all resulting dates will also be last business day of month.
- Added basic date generation starting from the end.

MATH

- Added Gauss-Kronrod integration algorithm.
- Added primitive polynomial modulo 2 up to dimension 18 (available up to dimension 27.)
- Added BicubicSplineInterpolation.
- Numerical Recipes algorithm is back since there is a problem with Nicolas' code: it is unable to fit a straight line, it waves around the line.
- Prime number generation.
- Acklam's approximation for inverse cumulative normal distribution function (replaced Moro's algorithm as default.)
- Added error function.
- Improved Cumulative Normal Distribution function using the error function.
- Matrix pseudo square algorithm using salvaging algorithm(s).
- Added SequenceStatistics.
- Major Statistic reworking.

- Added DiscrepancyStatistic that inherits from SequenceStatistic and extends it with the calculation of L2-discrepancy.
- HStatistics.
- Added first and second derivative ot cubic splines.

RISK MEASURES

- Introduced semiVariance and regret.
- Redefinition of average shorfall (normalization factor now is cumulative(target) instead of 1.0)

MISCELLANEA

- QuEP 9 "generic disposable objects" implemented.
- Added test suite.
- Dataformatters extended to format long integers, Ordinal numerals, power of two formatting.
- Exercise class adopted.
- Added user configuration section.
- Inhibited automatic conversion of Handle<T> to RelinkableHandle<T>.
- Diffusion process extended.
- Added strikeSensitivity to the Greeks.
- BS does handle t==0.0 and sigma==0.0.
- TimeGrid has been reworked.
- Added payoff file for Payoff classes. Added Cash-Or-Nothing and Asset-Or-Nothing payoff classes.
- Upgraded to use Doxygen 1.3.

Release 0.3.1 - February 4th, 2003

FINITE DIFFERENCE FRAMEWORK

- partially implemented QuEP 2 (http://quantlib.org/quep.shtml)

VOLATILITY FRAMEWORK

- added Black and local volatility interface

PRICING ENGINES FRAMEWORK

- partially implemented QuEP 5 (http://quantlib.org/quep.shtml)

YIELD TERM STRUCTURE

- interface revisited
- added discrete time forward methods
- added DiscountCurve (loglinear interpolated) and CompoundForward term structures
- ForwardSpreadedTermStructure moved under QuantLib::TermStructures namespace

FIXED INCOME

- Modified coupons so that the payment date can be after the end of the accrual period

MISCELLANEA

- added/verified holidays of many calendars
- added new calendars
- added new currencies
- more date formatters
- added Period(std::string&)
- it is now possible to advance a calandar using a Period
- added LogLinear Interpolation
- the allowExtrapolation boolean in interpolation classes has been removed from constructors and added to the operator()
- Renamed Solver1D::lowBound and hiBound
- bug fixes

BUILD PROCESS

- More autoconfiscated time functions and types
- Migrated to latest autotools
- added patches for Darwin and Solaris

Release 0.3.0 - May 6th, 2002

MONTE CARLO FRAMEWORK

- Path and MultiPath are time-aware
- McPricer: extended interface, improved convergency algorithm

FINITE DIFFERENCE FRAMEWORK

- added mixed (implicit/explicit) scheme, from which Crank-Nicolson, ImplicitEuler, and ExplicitEuler are now derived
- Finite Difference exercise conditions are now in the FiniteDifferences folder/namespace
- Finite Difference pricers now start with 'Fd' letters

- BSMNumericalOption became BsmFdOption

LATTICE FRAMEWORK

- introduced first version of the framework
- CRR and JR binomial trees

VOLATILITY FRAMEWORK

- early works on reorganization of vol structures

YIELD TERM STRUCTURE

- new TermStructure class based on affine model
- yield curves can be spreaded in term of zeros (ZeroSpreadedTermStructure) and forwards (ForwardSpreadedTermStructure)
- Added dates() and times() to PiecewiseFlatForward
- discount factor accuracy in the yield curve bootstrapping is an input
- added single factor short-rate models (Hull-White, Black-Karasinski)
- added two factor short-rate models framework
- cap/floor and swaption calibration helpers
- added bermudan swaption pricing example (including BK and HW calibrations)

FIXED INCOME

- cap/floor and swaption tree pricer
- cap/floor analytical pricer
- vanilla swaption Jamshidian pricer
- Added accruedAmount() to coupons
- Made cash flow vector builders into functions

OPTIMIZATION FRAMEWORK

- added conjugate gradient, simplex

PATTERNS

- implemented QuEP 8 and 10

MISCELLANEA

- added allowExtrapolation parameter to interpolaton classes
- added 2D bilinear interpolation

- better spline interpolation algorithm
- Added non-central chi-square distribution function.
- Improved Inverse Cumulative Normal Distribution using Moro's algorithm
- Introduced class representing stochastic processes
- added isExpired() to Instrument interface
- added functions folder and namespace for QuantLibXL and any other function-like interface to QuantLib
- Handle is now castable to an Handle of a compatible type
- added downsideVariance to the Statistics class
- kustosis() and skewness() now handles the case of stddev == 0 and/or variance == 0
- added Correlation Matrix to MultiVariateAccumulator
- enforced MS VC compilation settings
- added "-debug" to the QL_VERSION version string ifdef QL_DEBUG
- "make check" runs the example programs under Borland C++
- fixed compilation with "g++ -pedantic"
- Spread as market element
- new calendars introduced
- new Xibor Indexes introduced
- Added optional day count to libor indexes
- Shortened file names within 31 char limit to support HFS

Release 0.2.1 - December 3rd, 2001

MONTE CARLO FRAMEWORK

- Path and MultiPath are now classes on their own
- PathPricer now handles both Path and MultiPath
- MonteCarloModel now handles both single factor and multi factors simulations.
- McPricer now handles both single factor and multi factors pricing. New pricing interface
- antithetic variance-reduction technique made possible in Monte Carlo for both single factor and multi factors
- Control Variate specific class removed: control variation technique is now handled by the general MC model
- average price and average strike asian option refactored
- Sample as a (value,weight) struct
- random number generators moved under RandomNumbers folder and namespace

FINITE DIFFERENCE FRAMEWORK

- BackwardEuler and ForwardEuler renamed ImplicitEuler and ExplicitEuler, respectively
- refactoring of TridiagonalOperator and derived classes

YIELD TERM STRUCTURE AND FIXED INCOME

- Added some useful methods to term structure classes
- Allowed passing a quote to RateHelpers as double
- added FuturesRateHelpers (no convexity adjustment yet)
- PiecewiseFlatForward now observer of rates passed as MarketElements
- Unified Date and Time interface in TermStructure
- Added BPS to generic swap legs
- added term_structure+swap example
- Fixing days introduced for floating-coupon bond

PATTERNS

- Added factory pattern
- Calendar and DayCounter now use the Strategy pattern

VARIOUS

- used do-while-false idiom in QL_REQUIRE-like macros
- now using size_t where appropriate
- dividendYield is now a Spread instead of a Rate (that is: cost of carry is allowed)
- RelinkableHandle initialized with an optional Handle
- Worked around VC++ problems in History constructor
- added QL_VERSION and QL_HEX_VERSION
- generic bug fixes
- removed classes deprecated in 0.2.0

INSTALLATION FACILITIES

- improved and smoother Win32 binary installer

DOCUMENTATION

- general re-hauling
- improved and extended Monte Carlo documentation
- improved and extended examples

- Upgraded to Doxygen 1.2.11.1
- Added man pages for installed executables
- added docs in Windows Help format
- added info on "Win32 OnTheEdgeRelease" and "Win32 OnTheEdgeDebug" MS VC++ configurations
- additional information on how to create a MS VC++ project based on QuantLib

Release 0.2.0 - September 18th, 2001

- Library:
  - source code moved under ql, better GNU standards
  - gcc build dir can now be separated from source tree
  - gcc 3.0.1 port
  - clean compilation (no warnings)
  - bootstrap script on cygwin
  - Fixed automatic choice of seed for random number generators
  - Actual/actual classes
  - extended platform support (see table in documentation)
  - antithetic variance-reduction technique made possible in Monte Carlo
  - added dividend-Rho greek
  - First implementation of segment integral (to be redesigned)
  - Knuth random generator
  - Cash flows, scheduler, and swap (both generic and simple) added
  - added ICGaussian random generator
  - generic bug fixes
- Installation facilities:
  - improved and smoother Win32 binary installer
  - better distribution
  - debian packages available
- Documentation:
  - general re-hauling
  - added examples of using QuantLib and of projects based on QL

Release 0.1.9 - May 31st, 2001

- Library:
  - Style guidelines introduced (see http://quantlib.org/style.shtml) and partially enforced
  - full support for Microsoft Visual Studio
  - full support for Linux/gcc

  - momentarily broken support for Metrowerks CodeWarrior
  - autoconfiscation (with specialized config.*.hpp files for platforms without automake/autoconf support)
  - Include files moved under Include/ql folder and referenced as "ql/header.hpp"
  - Implemented expression templates techniques for array algebra optimization
  - Added custom iterators
  - Improved term structure
  - Added Asian, Bermudan, Shout, Cliquet, Himalaya, and Barrier options (all with greeks calculation, control variated where possible)
  - Added Helsinki and Wellington calendars
  - Improved Normal distribution related functions: cumulative, inverse cumulative, etc.
  - Added uniform and Gaussian random number generators
  - Added Statistics class (mean, variance, skewness, downside variance, etc.)
  - Added RiskMeasures class: VAR, average shortfall, expected shortfall, etc.
  - Added RiskStatistics class combining Statistics and RiskMeasures
  - Added sample accumulator for multivariate analysis
  - Added Monte Carlo tools
  - Added matrix-related functions (square root, symmetric Schur decomposition)
  - Added interpolation framework (linear and cubic spline interpolation implemented).

- Installation facilities:
  - Added Win32 GUI installer for binaries

- Documentation:
  - support for Doxygen 1.2.7
  - Added man documentation

Release 0.1.1 - November 21st, 2000

Initial release.

1.9 Additional resources

The main QuantLib resource is the QuantLib web site (http://quantlib.org).

Additional resources available from the above site include:

- current news (http://sourceforge.net/news/?group_id=12740);
- the QuantLib mailing lists and forums (http://quantlib.org/mailinglists.shtml);
- the QuantLib programming style guidelines (http://quantlib.org/style.shtml);
- a link to the QuantLib project page on SourceForge.net (http://sourceforge.net/projects/quantlib);
- links to pages for bug reports (http://sourceforge.net/tracker/?group_id=12740&atid=112740), patch submissions (http://sourceforge.net/tracker/?group_id=12740&atid=312740), and feature requests (http://sourceforge.net/tracker/?group_id=12740&atid=362740);
- a page (http://quantlib.org/extensions.shtml) about how to use QuantLib in other languages/platforms;
- QuantLib web-site statistics (http://sourceforge.net/project/stats/?group_id=12740);
- as well as links to additional quantitative finance resources.

1.10 The QuantLib Group

1.10.1 Authors

The QuantLib Group members are:

- Ferdinando Ametrano, Monte Paschi Asset Management sgr, administrator
- Luigi Ballabio, StatPro Italia srl, administrator
- Mario Aleppo, StatPro Italia srl
- Nicolas Di Césaré
- Dirk Eddelbuettel
- Neil Firth, Mathematical Institute, University of Oxford
- André Louw, Decillion Pty
- Marco Marchioro, StatPro Italia srl
- Sadruddin Rejeb
- Niels Elken Sønderby
- Enrico Sirola, StatPro Italia srl
- Liguo Song
- Joseph Wang

QuantLib also includes code taken from Peter Jäckel's book "Monte Carlo Methods in Finance".

1.10.2 Contributors

We gratefully acknowledge contributions from Xavier Abulker, Toyin Akin, Sercan Atalik, James Battle, Christopher Baus, Thomas Becker, Adolfo Benin, Luca Berardi, David Binderman, Antoine Cellerier, Aurelien Chanudet, Jon Davidson, Daniele De Francesco, Matteo Gallivanoni, Roman Gitlin, Tomoya Kawanishi, Gary Kennedy, Enrico Michelotti, Gilbert Peffer, Walter Penschke, Gianni Piolanti, Peter Schmitteckert, David Schwartz, Maxim Sokolov, Klaus Spanderen, Marco Tarenghi, Charles Whitmore, Bernd Johannes Wuebben, and Jeff Yu.

1.11 QuantLib License

```
QuantLib is
    Copyright (C) 2002, 2003, 2004, 2005 Ferdinando Ametrano
    Copyright (C) 2000, 2001, 2002, 2003, 2004, 2005 StatPro Italia srl

    Copyright (C) 2002, 2003, 2004 Decillion Pty(Ltd)
    Copyright (C) 2001, 2002, 2003 Nicolas Di Césaré
    Copyright (C) 2003, 2004 Neil Firth
    Copyright (C) 2001, 2002, 2003 Sadruddin Rejeb
    Copyright (C) 2003 Niels Elken Sønderby

    Copyright (C) 2004 FIMAT Group
    Copyright (C) 2003, 2004 Roman Gitlin
    Copyright (C) 2004 M-Dimension Consulting Inc.
    Copyright (C) 2004 Mike Parker
    Copyright (C) 2004 Walter Penschke
    Copyright (C) 2004 Gianni Piolanti
    Copyright (C) 2003 Kawanishi Tomoya
    Copyright (C) 2004 Jeff Yu

    Copyright (C) 2005 Sercan Atalik
    Copyright (C) 2005 Gary Kennedy
    Copyright (C) 2004, 2005 Klaus Spanderen
    Copyright (C) 2005 Joseph Wang
    Copyright (C) 2005 Charles Whitmore

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

    Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the documentation
    and/or other materials provided with the distribution.

    Neither the names of the copyright holders nor the names of the QuantLib
    Group and its contributors may be used to endorse or promote products
    derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

1.11.1 Comments on Copyright and License

QuantLib is Non-Copylefted Free Software [1] released under the modified BSD License [2] (also know as XFree86-style license).

QuantLib is Open Source [3] because of its license: it is OSI Certified Open Source Software [4]. OSI Certified is a certification mark of the Open Source Initiative [5].

The modified BSD License is GPL compatible as confirmed by the Free Software Foundation [6].

This license has been adopted to allow free use of QuantLib and its source, to make QuantLib flourish as a free-software/open-source project. It allows proprietary extensions to be commercialized.

[1] http://www.gnu.org/philosophy/categories.html#Non-CopyleftedFreeSoftware

[2] http://www.opensource.org/licenses/bsd-license.html

[3] http://www.opensource.org/docs/definition.html

[4] http://www.opensource.org/docs/certification_mark.html

[5] http://www.opensource.org

[6] http://www.gnu.org/philosophy/bsd.html

Chapter 2

QuantLib Module Index

2.1 QuantLib Modules

Here is a list of all modules:

Chapter 3

QuantLib Hierarchical Index

3.1 QuantLib Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

QuantLib Class Index

4.1 QuantLib Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

QuantLib File Index

5.1 QuantLib File List

Here is a list of all documented files with brief descriptions:

Chapter 6

QuantLib Module Documentation

6.1 Numeric types

6.1.1 Detailed Description

A number of numeric types are defined in order to add clarity to function and method declarations.

Typedefs

- typedef QL_INTEGER QuantLib::Integer
  integer number
- typedef QL_BIG_INTEGER QuantLib::BigInteger
  large integer number
- typedef unsigned QL_INTEGER QuantLib::Natural
  positive integer
- typedef QL_REAL QuantLib::Real
  real number
- typedef Real QuantLib::Decimal
  decimal number
- typedef std::size_t QuantLib::Size
  size of a container
- typedef Real QuantLib::Time
  continuous quantity with 1-year units
- typedef Real QuantLib::DiscountFactor
  discount factor between dates
- typedef Real QuantLib::Rate
  interest rates

- typedef Real QuantLib::Spread
  spreads on interest rates
- typedef Real QuantLib::Volatility
  volatility

6.2 Currencies and FX rates

Classes

- class ZARCurrency
  South-African rand.
- class ARSCurrency
  Argentinian peso.
- class BRLCurrency
  Brazilian real.
- class CADCurrency
  Canadian dollar.
- class CLPCurrency
  Chilean peso.
- class COPCurrency
  Colombian peso.
- class MXNCurrency
  Mexican peso.
- class TTDCurrency
  Trinidad & Tobago dollar.
- class USDCurrency
  U.S. dollar.
- class VEBCurrency
  Venezuelan bolivar.
- class BDTCurrency
  Bangladesh taka.
- class CNYCurrency
  Chinese yuan.
- class HKDCurrency
  Honk Kong dollar.
- class ILSCurrency
  Israeli shekel.
- class INRCurrency
  Indian rupee.
- class IQDCurrency

Iraqi dinar.

- class IRRCurrency
  Iranian rial.
- class JPYCurrency
  Japanese yen.
- class KRWCurrency
  South-Korean won.
- class KWDCurrency
  Kuwaiti dinar.
- class NPRCurrency
  Nepal rupee.
- class PKRCurrency
  Pakistani rupee.
- class SARCurrency
  Saudi riyal.
- class SGDCurrency
  Singapore dollar.
- class THBCurrency
  Thai baht.
- class TWDCurrency
  Taiwan dollar.
- class BGLCurrency
  Bulgarian lev.
- class BYRCurrency
  Belarussian ruble.
- class CHFCurrency
  Swiss franc.
- class CYPCurrency
  Cyprus pound.
- class CZKCurrency
  Czech koruna.
- class DKKCurrency
  Danish krone.

- class EEKCurrency
  Estonian kroon.
- class EURCurrency
  European Euro.
- class GBPCurrency
  British pound sterling.
- class HUFCurrency
  Hungarian forint.
- class ISKCurrency
  Iceland krona.
- class LTLCurrency
  Lithuanian litas.
- class LVLCurrency
  Latvian lat.
- class MTLCurrency
  Maltese lira.
- class NOKCurrency
  Norwegian krone.
- class PLNCurrency
  Polish zloty.
- class ROLCurrency
  Romanian leu.
- class SEKCurrency
  Swedish krona.
- class SITCurrency
  Slovenian tolar.
- class SKKCurrency
  Slovak koruna.
- class TRLCurrency
  Turkish lira.
- class TRYCurrency
  New Turkish lira.
- class ATSCurrency
  Austrian shilling.

- class BEFCurrency
  Belgian franc.
- class DEMCurrency
  Deutsche mark.
- class ESPCurrency
  Spanish peseta.
- class FIMCurrency
  Finnish markka.
- class FRFCurrency
  French franc.
- class GRDCurrency
  Greek drachma.
- class IEPCurrency
  Irish punt.
- class ITLCurrency
  Italian lira.
- class LUFCurrency
  Luxembourg franc.
- class NLGCurrency
  Dutch guilder.
- class PTECurrency
  Portuguese escudo.
- class AUDCurrency
  Australian dollar.
- class NZDCurrency
  New Zealand dollar.

6.3 Date and time calculations

6.3.1 Detailed Description

The concrete class `QuantLib::Date` implements the concept of date. Its functionalities include:

- providing basic information such as weekday, day of the month, day of the year, month, and year;
- comparing two dates to determine whether they are equal, or which one is the earlier or later, or the difference between them expressed in days;
- incrementing or decrementing a date of a given number of days, or of a given period expressed in weeks, months, or years.

Modules

- Calendars
- Day counters

Classes

- class Calendar

  calendar class

- class Period

  Time period described by a number of a given time unit.

- class Date

  Concrete date class.

- class DayCounter

  day counter class

Typedefs

- typedef Integer QuantLib::Day

  Day number.

- typedef Integer QuantLib::Year

  Year number.

Enumerations

- enum QuantLib::BusinessDayConvention {

  QuantLib::Unadjusted, QuantLib::Preceding, QuantLib::ModifiedPreceding, QuantLib::Following,

  QuantLib::ModifiedFollowing, QuantLib::MonthEndReference }

  Business Day conventions.

- enum QuantLib::Weekday {

  Sunday = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,

  Thursday = 5, **Friday** = 6, **Saturday** = 7, **Sun** = 1,

  Mon = 2, **Tue** = 3, **Wed** = 4, **Thu** = 5,

  Fri = 6, **Sat** = 7 }

- enum QuantLib::Month {

  January = 1, **February** = 2, **March** = 3, **April** = 4,

  May = 5, **June** = 6, **July** = 7, **August** = 8,

  September = 9, **October** = 10, **November** = 11, **December** = 12,

  Jan = 1, **Feb** = 2, **Mar** = 3, **Apr** = 4,

  Jun = 6, **Jul** = 7, **Aug** = 8, **Sep** = 9,

  Oct = 10, **Nov** = 11, **Dec** = 12 }

  Month names.

- enum QuantLib::IMMMonth { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }

  Main cycle of the International Money Market (a.k.a. IMM) Months.

- enum QuantLib::Frequency {

  QuantLib::NoFrequency = -1, QuantLib::Once = 0, QuantLib::Annual = 1, QuantLib::Semiannual = 2,

  QuantLib::EveryFourthMonth = 3, QuantLib::Quarterly = 4, QuantLib::Bimonthly = 6, QuantLib::Monthly = 12 }

  Frequency of events.

- enum QuantLib::TimeUnit { **Days**, **Weeks**, **Months**, **Years** }

  Units used to describe time periods.

6.3.2 Enumeration Type Documentation

6.3.2.1 enum BusinessDayConvention

Business Day conventions.

These conventions specify the algorithm used to adjust a date in case it is not a valid business day.

Enumerator:

Unadjusted Do not adjust.

Preceding Choose the first business day before the given holiday.

ModifiedPreceding Choose the first business day before the given holiday unless it belongs to a different month, in which case choose the first business day after the holiday.

Following Choose the first business day after the given holiday.

ModifiedFollowing Choose the first business day after the given holiday unless it belongs to a different month, in which case choose the first business day before the holiday.

MonthEndReference Choose the first business day after the given holiday, if the original date falls on last business day of month result reverts to first business day before month-end

Examples:
BermudanSwaption.cpp, and swapvaluation.cpp.

6.3.2.2 enum Weekday

Day's serial number MOD 7; WEEKDAY Excel function is the same except for Sunday = 7.

6.3.2.3 enum IMMMonth

Main cycle of the International Money Market (a.k.a. IMM) Months.

Deprecated
use IMM::Month instead

6.3.2.4 enum Frequency

Frequency of events.

Enumerator:
NoFrequency null frequency

Once only once, e.g., a zero-coupon

Annual once a year

Semiannual twice a year

EveryFourthMonth every fourth month

Quarterly every third month

Bimonthly every second month

Monthly once a month

Examples:
BermudanSwaption.cpp, and swapvaluation.cpp.

6.4 Calendars

6.4.1 Detailed Description

The class `QuantLib::Calendar` provides the interface for determining whether a date is a business day or a holiday for a given exchange or a given country, and for incrementing/decrementing a date of a given number of business days. A number of calendars is contained in the `ql/Calendars` directory.

Classes

- class Beijing
  Beijing calendar
- class Bombay
  Bombay calendar
- class Bratislava
  Bratislava calendar
- class Budapest
  Budapest calendar
- class Copenhagen
  Copenhagen calendar
- class Germany
  German calendars.
- class Helsinki
  Helsinki calendar
- class HongKong
  Hong Kong calendar.
- class Istanbul
  Istanbul calendar
- class Italy
  Italian calendars.
- class Johannesburg
  Johannesburg calendar
- class JointCalendar
  Joint calendar.
- class NullCalendar
  Calendar for reproducing theoretical calculations.

- class Oslo
  Oslo calendar
- class Prague
  Prague calendar
- class Riyadh
  Riyadh calendar
- class Seoul
  Seoul calendar
- class Singapore
  Singapore calendar
- class Stockholm
  Stockholm calendar
- class Sydney
  Sydney calendar (New South Wales, Australia)
- class Taipei
  Taipei calendar
- class Taiwan
  Taiwan calendar
- class TARGET
  TARGET calendar
- class Tokyo
  Tokyo calendar
- class Toronto
  Toronto calendar
- class UnitedKingdom
  United Kingdom calendars.
- class UnitedStates
  United States calendars.
- class Warsaw
  Warsaw calendar
- class Wellington
  Wellington calendar
- class Zurich
  Zurich calendar

6.5 Day counters

6.5.1 Detailed Description

The class `QuantLib::DayCounter` provides more advanced means of measuring the distance between two dates according to a given market convention, both as number of days of fraction of year. A number of such conventions is contained in the `ql/DayCounters` directory.

Classes

- class Actual360

  Actual/360 day count convention.

- class Actual365Fixed

  Actual/365 (Fixed) day count convention.

- class ActualActual

  Actual/Actual day count.

- class OneDayCounter

  1/1 day count convention

- class SimpleDayCounter

  Simple day counter for reproducing theoretical calculations.

- class Thirty360

  30/360 day count convention

6.6 Pricing engines

Modules

- Asian option engines
- Barrier option engines
- Basket option engines
- Cap/floor engines
- Cliquet option engines
- Forward option engines
- Quanto option engines
- Swaption engines
- Vanilla option engines

6.7 Asian option engines

Classes

- class AnalyticContinuousGeometricAveragePriceAsianEngine
  Pricing engine for European continuous geometric average price Asian.
- class AnalyticDiscreteGeometricAveragePriceAsianEngine
  Pricing engine for European discrete geometric average price Asian.
- class MCDiscreteArithmeticAPEngine
  Monte Carlo pricing engine for discrete arithmetic average price Asian.
- class MCDiscreteGeometricAPEngine
  Monte Carlo pricing engine for discrete geometric average price Asian.
- class MCDiscreteAveragingAsianEngine
  Pricing engine for discrete average Asians using Monte Carlo simulation.

6.8 Barrier option engines

Classes

- class AnalyticBarrierEngine

  Pricing engine for barrier options using analytical formulae.

- class MCBarrierEngine

  Pricing engine for barrier options using Monte Carlo simulation.

6.9 Basket option engines

Classes

- class MCAmericanBasketEngine

  least-square Monte Carlo engine

- class MCBasketEngine

  Pricing engine for basket options using Monte Carlo simulation.

- class StulzEngine

  Pricing engine for 2D European Baskets.

6.10 Cap/floor engines

Classes

- class AnalyticCapFloorEngine

  Analytic engine for cap/floor.

- class BlackCapFloorEngine

  Black-formula cap/floor engine.

- class TreeCapFloorEngine

  Numerical lattice engine for cap/floors.

6.11 Cliquet option engines

Classes

- class AnalyticCliquetEngine

  Pricing engine for Cliquet options using analytical formulae.

- class AnalyticPerformanceEngine

  Pricing engine for performance options using analytical formulae.

6.12 Forward option engines

Classes

- class ForwardEngine

  Forward engine base class.

- class ForwardPerformanceEngine

  Forward performance engine.

6.13 Quanto option engines

Classes

- class QuantoEngine

  Quanto engine base class.

6.14 Swaption engines

Classes

- class BlackSwaptionEngine

  Black-formula swaption engine.

- class G2SwaptionEngine

  Swaption priced by means of the Black formula

- class JamshidianSwaptionEngine

  Jamshidian swaption engine.

- class TreeSwaptionEngine

  Numerical lattice engine for swaptions.

6.15 Vanilla option engines

Classes

- class AnalyticDigitalAmericanEngine
- class AnalyticDividendEuropeanEngine

  Analytic pricing engine for European options with discrete dividends.

- class AnalyticEuropeanEngine

  Pricing engine for European vanilla options using analytical formulae.

- class AnalyticHestonEngine

  analytic Heston-model engine based on Fourier transform

- class BaroneAdesiWhaleyApproximationEngine
- class BatesEngine

  Bates model engines based on Fourier transform.

- class BinomialVanillaEngine

  Pricing engine for vanilla options using binomial trees.

- class BjerksundStenslandApproximationEngine
- class FDAmericanEngine

  Finite-differences pricing engine for American one asset options.

- class FDBermudanEngine

  Finite-differences Bermudan engine.

- class FDDividendAmericanEngine

  Finite-differences pricing engine for dividend American options.

- class FDDividendEngine

  Base finite-differences pricing engine for dividend options.

- class FDDividendEuropeanEngine

  Finite-differences pricing engine for dividend European options.

- class FDDividendShoutEngine

  Finite-differences shout engine with dividends.

- class FDEuropeanEngine

  Pricing engine for European options using finite-differences.

- class FDShoutEngine

  Finite-differences pricing engine for shout vanilla options.

- class FDStepConditionEngine

  Finite-differences pricing engine for American-style vanilla options.

- class FDVanillaEngine

Finite-differences pricing engine for BSM one asset options.

- class IntegralEngine
- class JumpDiffusionEngine

  Jump-diffusion engine for vanilla options.

- class JuQuadraticApproximationEngine
- class MCDigitalEngine

  Pricing engine for digital options using Monte Carlo simulation.

- class MCEuropeanEngine

  European option pricing engine using Monte Carlo simulation.

- class MCEuropeanHestonEngine

  Monte Carlo Heston-model engine for European options.

- class MCHestonEngine

  Monte Carlo Heston-model engine.

- class MCVanillaEngine

  Pricing engine for vanilla options using Monte Carlo simulation.

6.16 Finite-differences framework

6.16.1 Detailed Description

Warning: this section of the documentation is currently outdated. You will need to compare the information on this page with the present code for working pricers, such as FdAmericanOption.

This framework (corresponding to the ql/FiniteDifferences directory) contains basic building blocks for the numerical solution of a generic differential equation

$$\frac{\partial f}{\partial t} = Lf$$

where L is a differential operator in "space", i.e., one which does not contain partial derivatives in t but can otherwise contain any derivative in any other variable of the problem.

Writing the equation in the above form allows us to implement separately the discretization of the differential operator L and the time scheme used for the evolution of the solution. The QuantLib::FiniteDifferenceModel class acts as a glue for such two steps—which are outlined in the following sections—and provides the interface of the resulting finite difference model for the end user. Furthermore, it provides the possibility of checking and operating on the solution array at each step—which is typically used to apply an exercise condition for an option. This is also outlined in a section below.

6.16.2 Differential operators

The discretization of the differential operator L depends on the discretization chosen for the solution f of the given equation.

Such choice is obvious in the 1-D case where the domain $[a, b]$ of the equation is discretized as a series of points $x_i, i = 0 \dots N-1$ (note that the index is zero based) where $x_i = a + hi$ and $h = (b-a)/(N-1)$. In turn, the solution f of the equation is discretized as an array $u_i, i = 0 \dots N-1$ whose elements are defined as $u_i = f(x_i)$. The discretization of the differential operator follows by substituting the derivatives with the corresponding incremental ratios defined in terms of the f_i . A number of basic operators are defined in the framework which can be composed to form more complex operators, namely:

the first derivative $\partial/\partial x$ is discretized as the operator D_+ , defined as

$$D_+ u_i = \frac{u_{i+1} - u_i}{h}$$

and implemented in class QuantLib::DPlus; the operator D_- , defined as

$$D_- u_i = \frac{u_i - u_{i-1}}{h}$$

and implemented in class QuantLib::DMinus; and the operator D_0 , defined as

$$D_0 u_i = \frac{u_{i+1} - u_{i-1}}{2h}$$

and implemented in class QuantLib::DZero. The discretization error of the above operators is $O(h)$ for D_+ and D_- and $O(h^2)$ for D_0 ;

the second derivative $\partial^2/\partial x^2$ is discretized as the operator $D_+ D_-$, defined as

$$D_+ D_- u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

and implemented in class QuantLib::DPlusDMinus. Its discretization error is $O(h^2)$.

The boundary condition for the above operators is by default linear extrapolation. Methods are currently provided for setting other kinds of boundary conditions to a tridiagonal operator which these operators inherit, namely, Dirichlet—i.e., constant value—and Neumann—i.e., constant derivative—boundary conditions. This might change in the future as boundary conditions could be astracted and passed as an additional argument to the model.

A programmer can also implement its own operator. However, in order to fit into this framework it will have to implement a required interface depending on the chosen evolver (see below). Also, it is currently required to manage itself any boundary conditions. Again, this could change in the future.

On the other hand, there is no obvious choice in the 2-D case. While it is immediate to discretize the domain into a series of points (x_i, y_j) and the solution into a matrix $f_{ij} = f(x_i, y_j)$, there is a number of ways into which the f_{ij} can be arranged into an array—each of them determining a different discretization of the differential operators. One of such ways was implemented in the LexicographicalView class, while others will be implemented in the future. No 2-D operator is currently implemented.

6.16.3 Time schemes

Once the differential operator L has been discretized, a number of choices are available for discretizing the time derivative at the left-hand side of the equation.

In this framework, such choice is encapsulated in so-called evolvers which, given L and the solution $u^{(k)}$ at time t_k , yield the solution $u^{(k-1)}$ at the previous time step.

A number of evolvers are currently provided in the library which implement well-known schemes, namely,

the forward Euler explicit scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = Lu^{(k)}$$

hence

$$u^{(k-1)} = (I - \Delta t L)\, u^{(k)}$$

from which $u^{(k-1)}$ can be obtained directly;

the backward Euler implicit scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = Lu^{(k-1)}$$

hence

$$(I + \Delta t L)\, u^{(k-1)} = u^{(k)}$$

from which $u^{(k-1)}$ can be obtained by solving a linear system;

the Crank-Nicolson scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = L\frac{u^{(k)} + u^{(k-1)}}{2}$$

hence

$$\left(I + \frac{\Delta t}{2} L\right) u^{(k-1)} = \left(I - \frac{\Delta t}{2} L\right) u^{(k)}$$

from which $u^{(k-1)}$ can be obtained by solving a linear system.

Each of the above evolvers forces a set of interface requirements upon the differential operator which are detailed in the documentation of the corresponding class, namely, QuantLib::ExplicitEuler, QuantLib::ImplicitEuler, and QuantLib::CrankNicolson, respectively.

A programmer could implement its own evolver, which does not need to inherit from any base class.

However, it must implement the following interface:

```
class Evolver {
  public:
    typedef ... arrayType;
    typedef ... operatorType;
    // constructors
    Evolver(const operatorType& D);
    // member functions
    void step(arrayType& a, Time t) const;
    void setStep(Time dt);
};
```

Finally, we note again that the pricing of an option requires the finite difference model to solve the corresponding equation *backwards* in time. Therefore, given a discretization u of the solution at a given time t , the call

```
evolver.step(u,t)
```

must calculate the discrete solution at the *previous* time, $t - dt$.

6.16.4 Step conditions

A finite difference model can be passed a step condition to be applied at each step during the rollback of the solution (e.g. the early exercise American condition). Such condition must be embodied in a class derived from QuantLib::StepCondition and must implement the interface of the latter, namely,

```
class MyCondition : public StepCondition<arrayType> {
  public:
    void applyTo(arrayType& a, Time t) const;
};
```

6.16.5 An example of finite difference model

The Black-Scholes equation can be written in the above form as

$$\frac{\partial f}{\partial t} = -\frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2} - \nu\frac{\partial f}{\partial x} + rf.$$

It can be seen that the operator L_{BS} is

$$L_{BS} = -\frac{\sigma^2}{2}\frac{\partial^2}{\partial x^2} - \nu\frac{\partial}{\partial x} + rI$$

and can be built from the basic operators provided in the library as

$$L_{BS} = -\frac{\sigma^2}{2}D_+D_- - \nu D_0 + rI.$$

Its implementation closely reflects the above decomposition and can be written as

```
class BlackScholesOperator : public TridiagonalOperator {
  public:
    BlackScholesOperator(
        double sigma, double nu,    // parameters of the
        Rate r,                     // Black-Scholes equation;
        unsigned int points,        // number of discretized points;
        double h)                   // grid spacing.
    : TridiagonalOperator(
        // build the operator by adding basic ones
        - (sigma*sigma/2.0) * DPlusDMinus(points,h)
        - nu * DZero(points,h)
        + r * TridiagonalOperator::identity(points)
    ) {}
};
```

taking as inputs the relevant parameters of the equation (σ , ν and r) as well as model parameters such as the number N of grid points and their spacing h .

As simple example cases, we will use the above operator to price both an European and an American option. The parameters of the two options will be the same, namely, they will be both call options with underlying price $u = 100$, strike $s = 95$, residual time $T = 1$ year, dividend yield $q = 3\%$ and volatility $\sigma = 10\%$. The risk-free rate will be $r = 5\%$. Such parameters are expressed using QuantLib types as

```
Option::Type type = Option::Call;
double underlying = 100.0, strike = 95.0;
Time residualTime = 1.0;
Rate dividendYield = 0.03, riskFreeRate = 0.05;
double volatility = 0.10;
```

The grid upon which the model will act will be a logarithmic grid of underlying prices, i.e., f will be defined in a range $[\ln u_{min}, \ln u_{max}]$ discretized as an array $x_i, i = 0 \ldots N-1$ with $x_i = \ln u_{min} + ih$ and $h = (\ln u_{max} - \ln u_{min})/(N-1)$. Such a grid and the corresponding vector of actual prices can be built as shown in the code below. The domain of the model will be defined as $[\ln u - \Delta, \ln u + \Delta]$ where $\Delta = 4\sigma\sqrt{T}$. A number of grid points $N = 101$ will be used.

```
unsigned int gridPoints = 101;
Array grid(gridPoints), prices(gridPoints);
double x0 = QL_LOG(underlying);
double Delta = 4.0*volatility*QL_SQRT(residualTime);
double xMin = x0 - Delta, xMax = x0 + Delta;
double h = (xMax-xMin)/(gridPoints-1);
for (unsigned int i=0; i<gridPoints; i++) {
    grid[i] = xMin + i*h;
    prices[i] = QL_EXP(grid[i]);
}
```

The initial condition is determined by the values of the option at maturity, i.e., either the difference between underlying price and strike if such difference is positive, or 0 if that is not the case (the above will have to be suitably modified for a put option or a straddle.) Such "initial" condition will be rolled back in time by our model.

```
Array exercisingValue(gridPoints);
for (unsigned int i=0; i<gridPoints; i++)
    exercisingValue[i] = QL_MAX(prices[i]-strike,0.0);
```

Now the differential operator can be initialized. Also, Neumann initial conditions are set which correspond to the initial value of the derivatives at the boundaries (see the BoundaryCondition class documentation for details).

```
double nu = riskFreeRate - dividendYield - volatility*volatility/2.0;
TridiagonalOperator L = BlackScholesOperator(volatility, nu,
    riskFreeRate, gridPoints, h);
L.setLowerBC(BoundaryCondition(BoundaryCondition::Neumann,
    exercisingValue[1]-exercisingValue[0]));
L.setUpperBC(BoundaryCondition(BoundaryCondition::Neumann,
    exercisingValue[gridPoints_-1]-exercisingValue[gridPoints_-2]));
```

We are now already set for the pricing of the European option. Also, the exercise condition is the only thing still to be defined for the American option to be priced. Such condition is equivalent to the statement that at each time step, the value of the option is the maximum between the profit realized in exercising the option (which we already calculated and stored in `exercisingValue`) and the value of the option should we keep it (which corresponds to the solution rolled back to the current time step). This logic can be implemented as:

```
class ExerciseCondition : public StepCondition<Array> {
  public:
    ExerciseCondition(const Array& exercisingValue)
    : exercisingValue_(exercisingValue) {}
    void applyTo(Array& a, Time) const {
        for (unsigned int i = 0; i < a.size(); i++)
            a[i] = QL_MAX(a[i], exercisingValue_[i]);
    }
  private:
    Array exercisingValue_;
};
```

Everything is now ready. The model can be created gluing the piece together by means of the QuantLib::FiniteDifferenceModel class. The current value of the option is calculated by rolling back the solution to the current time, i.e., $t = 0$, and by taking the value corresponding at the current underlying price—which by construction corresponds to the central value provided that the number of grid points is odd.

```
unsigned int timeSteps = 365;

// build the model - Crank-Nicolson scheme chosen
FiniteDifferenceModel<CrankNicolson<TridiagonalOperator> > model(L);

// European option
Array f = exercisingValue;  // initial condition
model.rollback(f, residualTime, 0.0, timeSteps);
double europeanValue = valueAtCenter(f);

// American option
f = exercisingValue;        // reset
Handle<StepCondition<Array> > condition(
    new ExerciseCondition(exercisingValue));
model.rollback(f, residualTime, 0.0, timeSteps, condition);
double americanValue = valueAtCenter(f);
```

Classes

- class BoundaryCondition

  Abstract boundary condition class for finite difference problems.

- class NeumannBC

  Neumann boundary condition (i.e., constant derivative).

- class DirichletBC
  Neumann boundary condition (i.e., constant value).
- class BSMOperator
  Black-Scholes-Merton differential operator.
- class BSMTermOperator
  Black-Scholes-Merton differential operator.
- class CrankNicolson
  Crank-Nicolson scheme for finite difference methods.
- class DMinus
  D_- *matricial representation*
- class DPlus
  D_+ *matricial representation*
- class DPlusDMinus
  D_+D_- *matricial representation*
- class DZero
  D_0 *matricial representation*
- class ExplicitEuler
  Forward Euler scheme for finite difference methods.
- class FiniteDifferenceModel
  Generic finite difference model.
- class ImplicitEuler
  Backward Euler scheme for finite difference methods.
- class MixedScheme
  Mixed (explicit/implicit) scheme for finite difference methods.
- class OneFactorOperator
  Interest-rate single factor model differential operator.
- class StepConditionSet
  Parallel evolver for multiple arrays.
- class StepCondition
  condition to be applied at every time step
- class NullCondition
  null step condition
- class TridiagonalOperator
  Base implementation for tridiagonal operator.

6.17 Short-rate modelling framework

6.17.1 Detailed Description

This framework (corresponding to the ql/ShortRateModels directory) implements some single-factor and two-factor short rate models. The models implemented in this library are widely used by practitionners. For the moment, the ShortRateModels::Model class defines the short-rate dynamics with stochastic equations of the type

$$dx_i = \mu(t, x_i)dt + \sigma(t, x_i)dW_t$$

where $r = f(t, x)$. If the model is affine (i.e. derived from the QuantLib::AffineModel class), analytical formulas for discount bonds and discount bond options are given (useful for calibration).

6.17.2 Single-factor models

The Hull & White model

$$dr_t = (\theta(t) - \alpha(t)r_t)dt + \sigma(t)dW_t$$

When α and σ are constants, this model has analytical formulas for discount bonds and discount bond options.

The Black-Karasinski model

$$d\ln r_t = (\theta(t) - \alpha \ln r_t)dt + \sigma dW_t$$

No analytical tractability here.

The extended Cox-Ingersoll-Ross model

$$dr_t = (\theta(t) - kr_t)dt + \sigma\sqrt{r_t}dW_t$$

There are analytical formulas for discount bonds (and soon for discount bond options).

6.17.3 Calibration

The class CalibrationHelper is a base class that facilitates the instanciation of market instruments used for calibration. It has a method marketValue() that gives the market price using a Black formula, and a modelValue() method that gives the price according to a model

Derived classed are QuantLib::CapHelper and QuantLib::SwaptionHelper.

For the calibration itself, you must choose an optimization method that will find constant parameters such that the value:

$$V = \sqrt{\sum_{i=1}^{n} \frac{(T_i - M_i)^2}{M_i}},$$

where T_i is the price given by the model and M_i is the market price, is minimized. A few optimization methods are available in the ql/Optimization directory.

6.17.4 Two-factor models

6.17.5 Pricers

Analytical pricers

If the model is affine, i.e. discount bond options formulas exist, caps are easily priced since they are a portfolio of discount bond options. Such a pricer is implemented in QuantLib::AnalyticalCapFloor. In the case of single-factor affine models, swaptions can be priced using the Jamshidian decomposition, implemented in QuantLib::JamshidianSwaption.

Using Finite Differences

(Doesn't work for the moment) For the moment, this is only available for single-factor affine models. If $x = x(t, r)$ is the state variable and follows this stochastic process:

$$dx_t = \mu(t, x)dt + \sigma(t, x)dW_t$$

any european-style instrument will follow the following PDE:

$$\frac{\partial P}{\partial t} + \mu\frac{\partial P}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 P}{\partial x^2} = r(t, x)P$$

The adequate operator to feed a Finite Difference Model instance is defined in the QuantLib::OneFactorOperator class.

Using Trees

Each model derived from the single-factor model class has the ability to return a trinomial tree. For yield-curve consistent models, the fitting parameter can be determined either analytically (when possible) or numerically. When a tree is built, it is then pretty straightforward to implement a pricer for any path-independant derivative. Just implement a class derived from NumericalDerivative (see QuantLib::NumericalSwaption for example) and roll it back until the present time... Just look at QuantLib::TreeCapFloor and QuantLib::TreeSwaption for working pricers.

Classes

- class AffineModel

  Affine model class.

- class TermStructureConsistentModel

  Term-structure consistent model class.

- class ShortRateModel

  Abstract short-rate model class.

- class OneFactorModel

  Single-factor short-rate model abstract class.

- class OneFactorAffineModel

  Single-factor affine base class.

- class BlackKarasinski

  Standard Black-Karasinski model class.

- class CoxIngersollRoss

  Cox-Ingersoll-Ross model class.

- class ExtendedCoxIngersollRoss

  Extended Cox-Ingersoll-Ross model class.

- class HullWhite

  Single-factor Hull-White (extended Vasicek) model class.

- class Vasicek

  Vasicek model class

- class TwoFactorModel

  Abstract base-class for two-factor models.

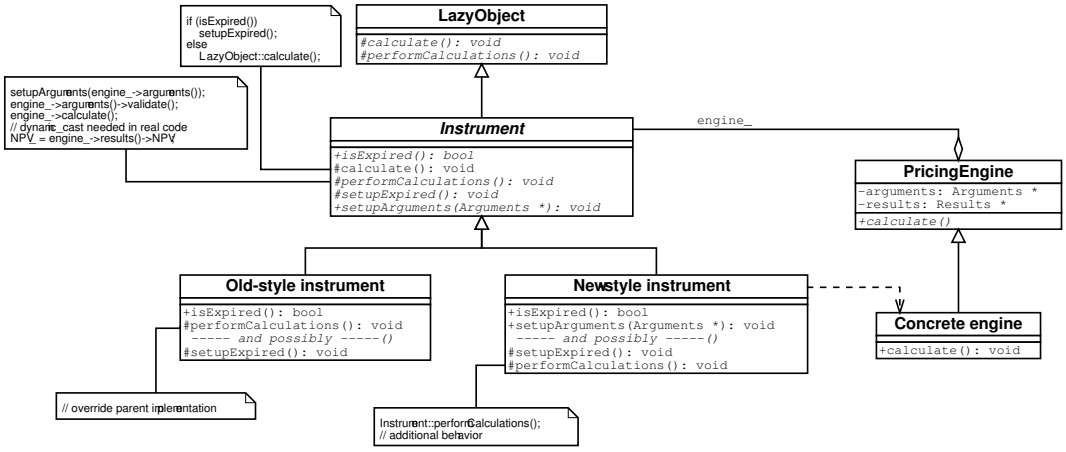
- class G2

  Two-additive-factor gaussian model class.

6.18 Financial instruments

6.18.1 Detailed Description

Since version 0.3.4, the `Instrument` class was reworked as shown in the following figure.



On the one hand, the checking of the expiration condition is now performed in a method `isExpired()` separated from the actual calculation, and a `setupExpired()` method is provided. The latter sets the NPV to 0.0 and can be extended in derived classes should any other results be returned.

On the other hand, the pricing-engine machinery previously contained in the Option class was moved upwards to the Instrument class. Also, the `setupEngine()` method was replaced by a `setupArguments(Arguments*)` method. This allows one to cleanly implement containment of instruments with code such as:

```
class FooArguments : public Arguments { ... };

class Foo : public Instrument {
  public:
    void setupArguments(Arguments*);
    ...
};

class FooOptionArguments : public FooArguments { ... };

class FooOption : public Option {
  private:
    Foo underlying_;
  public:
    void setupArguments(Arguments* args) {
        underlying_.setupArguments(args);
        // set the option-specific part
    }
    ...
};
```

which was more difficult to write with `setupEngine()`.

Therefore, there are now two ways to inherit from `Instrument`, namely:

1. implement the `isExpired` method, and completely override the `performCalculations` method so that it bypasses the pricing-engine machinery. If the class declared any other

results beside `NPV_` and `errorEstimate_`, the `setupExpired` method should also be extended so that those results are set to a value suitable for an expired instrument. This was the migration path taken for all instruments not previously deriving from the `Option` class.

2. define suitable argument and result classes for the instrument and implement the `isExpired` and `setupArguments` methods, reusing the pricing-engine machinery provided by the default `performCalculations` method. The latter can be extended by first calling the default implementation and then performing any additional tasks required by the instrument—most often, copying additional results from the pricing engine results to the corresponding data members of the instrument. As in the previous case, the `setupExpired` method can be extended to account for such extra data members.

Classes

- class ContinuousAveragingAsianOption

  Continuous-averaging Asian option.

- class DiscreteAveragingAsianOption

  Discrete-averaging Asian option.

- class BarrierOption

  Barrier option on a single asset.

- class BasketOption

  Basket option on a number of assets.

- class Bond

  Base bond class.

- class CapFloor

  Base class for cap-like instruments.

- class Cap

  Concrete cap class.

- class Floor

  Concrete floor class.

- class Collar

  Concrete collar class.

- class CliquetOption

  cliquet (Ratchet) option

- class DividendVanillaOption

  Single-asset vanilla option (no barriers) with discrete dividends.

- class EuropeanOption

  European option on a single asset.

- class FixedCouponBond
  fixed-coupon bond
- class FloatingRateBond
  floating-rate bond
- class ForwardVanillaOption
  Forward version of a vanilla option.
- class QuantoForwardVanillaOption
  Quanto version of a forward vanilla option.
- class QuantoVanillaOption
  quanto version of a vanilla option
- class SimpleSwap
  Simple fixed-rate vs Libor swap.
- class Stock
  Simple stock class.
- class Swap
  Interest rate swap.
- class Swaption
  Swaption class
- class VanillaOption
  Vanilla option (no discrete dividends, no barriers) on a single asset.
- class ZeroCouponBond
  zero-coupon bond

6.19 Lattice methods

6.19.1 Detailed Description

The framework (corresponding to the ql/Lattices directory) contains basic building blocks for pricing instruments using lattice methods (trees). A lattice, i.e. an instance of the abstract class QuantLib::Lattice, relies on one or several trees (each one approximating a diffusion process) to price an instance of the DiscretizedAsset class. Trees are instances of classes derived from QuantLib::Tree, classes which define the branching between nodes and transition probabilities.

6.19.2 Binomial trees

The binomial method is the simplest numerical method that can be used to price path-independent derivatives. It is usually the preferred lattice method under the Black-Scholes-Merton model. As an example, let's see the framework implemented in the bsmlattice.hpp file. It is a method based on a binomial tree, with constant short-rate (discounting). There are several approaches to build the underlying binomial tree, like Jarrow-Rudd or Cox-Ross-Rubinstein.

6.19.3 Trinomial trees

When the underlying stochastic process has a mean-reverting pattern, it is usually better to use a trinomial tree instead of a binomial tree. An example is implemented in the QuantLib::Trinomial-Tree class, which is constructed using a diffusion process and a time-grid. The goal is to build a recombining trinomial tree that will discretize, at a finite set of times, the possible evolutions of a random variable y satisfying

$$dy_t = \mu(t, y_t)dt + \sigma(t, y_t)dW_t.$$

At each node, there is a probability p_u, p_m and p_d to go through respectively the upper, the middle and the lower branch. These probabilities must satisfy

$$p_u y_{i+1,k+1} + p_m y_{i+1,k} + p_d y_{i+1,k-1} = E_{i,j}$$

and

$$p_u y^2_{i+1,k+1} + p_m y^2_{i+1,k} + p_d y^2_{i+1,k-1} = V^2_{i,j} + E^2_{i,j},$$

where k (the index of the node at the end of the middle branch) is the index of the node which is the nearest to the expected future value, $E_{i,j} = \mathbf{E}\left(y(t_{i+1})|y(t_i) = y_{i,j}\right)$ and $V^2_{i,j} = \mathbf{Var}\{y(t_{i+1})|y(t_i) = y_{i,j}\}$. If we suppose that the variance is only dependant on time $V_{i,j} = V_i$ and set y_{i+1} to $V_i\sqrt{3}$, we find that

$$p_u = \frac{1}{6} + \frac{(E_{i,j} - y_{i+1,k})^2}{6V_i^2} + \frac{E_{i,j} - y_{i+1,k}}{2\sqrt{3}V_i},$$

$$p_m = \frac{2}{3} - \frac{(E_{i,j} - y_{i+1,k})^2}{3V_i^2},$$

$$p_d = \frac{1}{6} + \frac{(E_{i,j} - y_{i+1,k})^2}{6V_i^2} - \frac{E_{i,j} - y_{i+1,k}}{2\sqrt{3}V_i}.$$

6.19.4 Bidimensional lattices

To come...

6.19.5 The QuantLib::DiscretizedAsset class

This class is a representation of the price of a derivative at a specific time. It is roughly an array of values, each value being associated to a state of the underlying stochastic variables. For the moment, it is only used when working with trees, but it should be quite easy to make a use of it in finite-differences methods. The two main points, when deriving classes from QuantLib::DiscretizedAsset, are:

1. Define the initialisation procedure (e.g. terminal payoff for european stock options).
2. Define the method adjusting values, when necessary, at each time steps (e.g. apply the step condition for american or bermudan options). Some examples are found in QuantLib::DiscretizedSwap and QuantLib::DiscretizedSwaption.

Classes

- class BinomialTree

  Binomial tree base class.

- class EqualProbabilitiesBinomialTree

  Base class for equal probabilities binomial tree.

- class EqualJumpsBinomialTree

  Base class for equal jumps binomial tree.

- class JarrowRudd

  Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

- class CoxRossRubinstein

  Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

- class AdditiveEQPBinomialTree

  Additive equal probabilities binomial tree.

- class Trigeorgis

  Trigeorgis (additive equal jumps) binomial tree

- class Tian

  Tian tree: third moment matching, multiplicative approach

- class LeisenReimer

  Leisen & Reimer tree: multiplicative approach.

- class BlackScholesLattice

  Simple binomial lattice approximating the Black-Scholes model.

- class Lattice

  Lattice-method base class.

- class Lattice1D

One-dimensional lattice.

- class Lattice2D

  Two-dimensional lattice.

- class Tree

  Tree approximating a single-factor diffusion

- class TrinomialTree

  Recombining trinomial tree class.

6.20 Math tools

Math facilities of the library include:

6.20.1 Pseudo-random number and low-discrepancy sequence generators

Implementations of pseudo-random number and low-discrepancy sequence generators. They share the ql/RandomNumbers directory.

6.20.2 One-dimensional solvers

The abstract class QuantLib::Solver1D provides the interface for one-dimensional solvers which can find the zeroes of a given function.

A number of such solvers is contained in the ql/Solvers1D directory.

The implementation of the algorithms was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery - Chapter 9

Some work is needed to resolve the ambiguity of the root finding accuracy defition: for some algorithms it is the x-accuracy, for others it is f(x)-accuracy.

6.20.3 Optimizers

The optimization framework (corresponding to the ql/Optimization directory) implements some multi-dimensional minimizing methods. The function to be minimized is to be derived from the QuantLib::CostFunction base class (if the gradient is not analytically implemented, it will be computed numerically).

The simplex method

This method, implemented in QuantLib::Simplex, is rather raw and requires quite a lot of computing resources, but it has the advantage that it does not need any evaluation of the cost function's gradient, and that it is quite easily implemented. First, we must choose N+1 starting points, given here by a starting point $\mathbf{P}_0$ and N points such that

$$\mathbf{P_i} = \mathbf{P}_0 + \lambda \mathbf{e_i},$$

where λ is the problem's characteristic length scale). These points will form a geometrical form called simplex. The principle of the downhill simplex method is, at each iteration, to move the worst point (highest cost function value) through the opposite face to a better point. When the simplex seems to be constrained in a valley, it will be contracted downhill, keeping the best point unchanged.

The conjugate gradient method

We'll now continue with a bit more sophisticated method, implemented in QuantLib::ConjugateGradient . At each step, we minimize (using Armijo's line search algorithm, implemented in QuantLib::ArmijoLineSearch) the function along a line defined by

$$\mathbf{d_i} = -\nabla f(\mathbf{x_i}) + \frac{\left\|\nabla f(\mathbf{x_i})\right\|^2}{\left\|\nabla f(\mathbf{x_{i-1}})\right\|^2}\mathbf{d_{i-1}},$$

$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0).$$

As we can see, this optimization method requires the knowledge of the gradient of the cost function. See QuantLib::ConjugateGradient .

6.21 Monte Carlo framework

6.21.1 Detailed Description

Warning: this section of the documentation is currently outdated.

This framework (corresponding to the ql/MonteCarlo directory) contains basic building blocks for the numerical calculation of the integral

$$\int_{\Omega} f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

where $p(\mathbf{x})$ is a normalized probability function. Monte Carlo methods solve the above integral by approximating it with the discrete sum

$$\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)w(\mathbf{x}_i)$$

where the $\mathbf{x}_i$ are drawn from $p(\mathbf{x})$, possibly with a weight $w(\mathbf{x}_i)$ — which otherwise can be considered uniformly equal to 1.

The above sum has a straightforward interpretation in the case of a derivative product, namely, the $\mathbf{x}_i$ are N generated random paths which the value of the underlying can possibly follow, while the $f(\mathbf{x}_i)$ are the values of the derivative on each of such paths. The sum above can therefore be taken as an estimate of the price of the derivative, namely, the average of its value on all possible paths — or rather all considered paths. Such a method enables the user to construct pricing classes for an unlimited range of derivatives, most notably path-dependent ones which cannot be priced by means of finite difference methods.

It must also be mentioned that for all such methods, the error e on the estimated value is proportional to the square root of the number of samples N . A number of so-called *variance-reduction* methods have been found which allows one to reduce the coefficient of proportionality between e and $1/\sqrt{N}$.

Separate implementations are provided in the library for the three components of the above average, namely, the random drawing of the $\mathbf{x}_i$, the evaluation of the $f(\mathbf{x}_i)$, and the averaging process itself. The QuantLib::MonteCarloModel class acts as a glue for such three steps — which are outlined in the following sections — and provides the interface of the resulting Monte Carlo model to the end user.

6.21.2 Path generation

The Black-Scholes equation

$$\frac{\partial f}{\partial t} + \frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2} + \nu\frac{\partial f}{\partial x} - rf = 0,$$

where r is the risk-free rate, σ is the volatility of the underlying, and $\nu = r - \sigma^2/2$, has the form of a diffusion process. According to this heuristic interpretation (1), paths followed by the logarithm of the underlying would be Brownian random walks with a constant drift ν per unit time and a standard deviation $\sigma\sqrt{T}$ over a time T .

Therefore, the paths to be generated for a Monte Carlo model of the Black-Scholes equation will be vectors of successive variations of the logarithm of the underlying price over M consecutive time intervals $\Delta t_i, i = 0 \ldots M-1$. Each such variation will be drawn from a Gaussian distribution with average $\nu\Delta T_i$ and standard deviation $\sigma\sqrt{\Delta T_i}$ — or possibly $\nu_i\Delta T_i$ and $\sigma_i\sqrt{\Delta T_i}$ should ν and σ vary in time.

The QuantLib::Path class stores the variation vector decomposed in its drift (determined) and diffusion (random) components. As shown below, this allows the implementation of antithetic variance reduction techniques.

The QuantLib::MultiPath class is a straightforward extension which acts as a vector of Path objects.

Classes are provided which generate paths and multi-paths with the desired drift and diffusion components, namely, QuantLib::PathGenerator and QuantLib::MultiPathGenerator.

For the time being, the path generator is initialized with a constant drift and variance. This requirement will most likely be relaxed in the next release. The multi-path generator is initialized with an array of constant drifts—one for each single asset—and a covariance matrix which encapsulates the relations between the diffusion components of the single assets.

The time discretization of the (multi)paths can be specified either as a given number of equal time steps over a given time span, or as a vector of explicitly specified times at which the path will be sampled.

6.21.3 Pricing an instrument on a path

The QuantLib::PathPricer class is the base class from which path pricers must inherit. The only method which subclasses are required to implement is

```
double operator()(const P&) const;
```

where P can be Path or MultiPath depending on the derivative whose value must be calculated.

Similarly, the term *path* will be used in the following discussion as meaning either path or multi-path depending on the context. The term *single path* is not to be taken as opposite to multi-path, but rather as meaning "a single instance of a (multi)path" as opposed to the set of all generated (multi)paths.

The above method encapsulates the pricing of the derivative on a single path and must return its value had the evolution of the underlying(s) followed the path passed as argument. For this reason, control variate techniques (see below) must not be implemented at this level since they would cause the returned value to differ from the actual price of the derivative on the path.

Instead, antithetic variance-reduction techniques can be effectively implemented at this level and indeed are used in the pricers currently included in the library.

In short, such techniques consist in pricing an option on both the given path and its antithetic, the latter being a path with the same drift and the opposite diffusion component. The value of the sample is defined as the average of the prices on the two paths.

A generic implementation of antithetic techniques could consist of a path pricer class which takes a concrete path pricer upon construction and whose operator() simply proxies two calls to the contained pricer, passing the given path and its antithetic, and averages the result. However, this would not take full advantage of the technique.

In fact, it must be noted that using antithetic paths not only reduces the variance *per se* but also allows to factor out calculations commons to a path and its antithetic, thus reducing greatly the computation time. Therefore, such techniques are best implemented inside the path pricer itself, whose algorithm can fully exploit such factorization.

A number of path pricers are available in the library and listed in reference manual.

6.21.4 Accumulating and averaging samples

The class QuantLib::MonteCarloModel encapsulates the general structure of a Monte Carlo calculations, namely, the generation of a number of paths, the pricing of the derivative on each path, and the averaging of the results to yield the actual derivative price.

As outlined above, the first two steps are delegated to a path generator and a path pricer. The third step is also delegated to an object which accumulates weighted values and returns the statistic properties of the set of such values. One such class provided by the library is QuantLib::Statistics.

The concern of the Monte Carlo model is therefore to act as a glue between such three components and can be expressed by the following pseudo-code:

```
given pathGenerator, pathPricer, accumulator;
for i in number of samples {
    path,weight = pathGenerator.next();
    price = pathPricer(path);
    accumulator.add(price,weight);
}
```

The Monte Carlo model also provides the user with the possibility to take advantage of control-variate techniques.

Such techniques consist in pricing a portfolio from which the price of the derivative can be deduced, but with a lower variance than the derivative alone.

In our current implementation, static-hedge control variate is used, namely, the formed portfolio is long of the derivative we need to price and short of a similar derivative whose price can be calculated analytically. The value of the portfolio on a given path will of course be given by the difference of the values of the two derivatives on such path. However, due to the similarity between the derivatives, the portfolio price will have a lower variance than either derivative alone since any variation in the price of the latter will be partly compensated by a similar variation in the price of the other. Lastly, given the portfolio price, the price of the derivative we are interested in can be deduced by adding the analytic value of the other.

In order to use such technique, the user must provide the model with a path pricer for the additional option and the value of the latter. The action of the Monte Carlo model is in this case expressed as:

```
given pathGenerator, pathPricer, cvPathPricer, cvPrice, accumulator;
for i in number of samples {
    path,weight  = pathGenerator.next();
    portfolioPrice = pathPricer(path) - cvPathPricer(path);
    accumulator.add(portfolioPrice+cvPrice,weight);
}
```

Martingale (a.k.a. dynamic-hedge) control variate techniques are planned for future releases.

A QuantLib::McPricer class is also available which wraps the typical usage of a Monte Carlo model.

Details on the Monte Carlo Pricer interface will be available in the Pricers section.

6.21.5 Examples of Monte Carlo models

As a simple example, we will use the outlined tools to price an European option by means of Monte Carlo techniques.

Given a current underlying price u_0 and a path $p = [p_1, \ldots, p_N]$ where every variation p_i is the sum of a drift term d_i and a random diffusion term r_i , the price of the underlying at maturity is

$$u = u_0 \prod_1^N e^{p_i} = u_0 \exp\left(\sum_1^N p_i\right) = u_0 \exp\left(\sum_1^N d_i + \sum_1^N r_i\right)$$

while the price on the antithetic path — i.e., same drift and opposite diffusion — is

$$u_0 \exp\left(\sum_1^N d_i - \sum_1^N r_i\right).$$

The corresponding path pricer can be implemented as:

```
class EuropeanPathPricer : public PathPricer<Path> {
  public:
    EuropeanPathPricer(Option::Type type, double underlying,
                       double strike, DiscountFactor discount,
                       bool useAntithetic)
    // just store the needed parameters
    : type_(type), underlying_(underlying), strike_(strike),
      discount_(discount), useAntithetic_(useAntithetic) {}
    // here is the logic
    double operator()(const Path& path) const {

        size_t n = path.size();

        // factor out the sums in the formula above
        double sum_d = 0.0, sum_r = 0.0;
        for (size_t i = 0; i < n; i++) {
            sum_d += path.drift()[i];
            sum_r += path.diffusion()[i];
        }

        // calculate final underlying price on path
        double price = underlying_*QL_EXP(sum_d+sum_r);

        // calculate payoff
        double payoff;
        switch (type_) {
          case Option::Call;
            payoff = QL_MAX(price-strike,0.0);
            break;
          // other cases are left as an exercise to the reader
          ...
        }

        // current value of the option is the discounted payoff
        double optionValue = payoff*discount_;

        // stop here if not antithetic...
        if (!useAntithetic_)
            return optionValue;

        // ...otherwise calculate the value on the antithetic path
        double antiPrice = underlying_*QL_EXP(sum_d-sum_r);

        // calculate payoff and option value as above
        ...

        // return the average of the results on the two paths
        return (optionValue + antiOptionValue)/2.0;
    }
  private:
```

```
    // stored parameters
    ...
};
```

The path pricer can now be used in a model. Let us assume the following parameters:

```
Option::Type type = Option::Call;
double underlying = 100.0, strike = 95.0;
Time residualTime = 1.0;
Rate dividendYield = 0.03, riskFreeRate = 0.05;
double volatility = 0.10;
```

The path generator can be instantiated as

```
// parameters of the Black-Scholes equation
double vol2 = volatility*volatility;
double nu = riskFreeRate - dividendYield - vol2/2.0;
// in this case we are only interested in the final underlying price.
// Therefore, we can cover all the residual time in one big time step.
int timeSteps = 1;

Handle<GaussianPathGenerator> pathGenerator(
    new GaussianPathGenerator(nu,vol2,residualTime,timeSteps));
```

where QuantLib::GaussianPathGenerator is a typedef to a path generator using the default choice for a Gaussian random number generator.

The path pricer is instantiated as

```
// discount at maturity
DiscountFactor discount = QL_EXP(-riskFreeRate*residualTime);
bool antithetic = true;

Handle<PathPricer<Path> > pathPricer(
    new EuropeanPathPricer(type,underlying,strike,discount,antithetic));
```

The model can now be created and used as following:

```
// number of samples to be generated
size_t samples = 1000000;

// pass the path generator and pricer we just created and a
// newly instantiated Statistics object
MonteCarloModel<Statistics,GaussianPathGenerator,PathPricer> model(
    pathGenerator,pathPricer,Statistics());

model.addSamples(samples);

// now get the results: the option price is given by value with
// a confidence level given by error
value = model.sampleAccumulator().mean();
error = model.sampleAccumulator().errorEstimate();
```

More examples of path pricers can be found in the ql/MonteCarlo directory, while examples of more sophisticated pricers which uses them in Monte Carlo models can be found in the ql/Pricers directory.

6.21.6 Notes

(1) A more rigorous approach would lead us to integrate the above equation by means of Green functions or Laplace transforms. Both such methods would show that the price at time $t = 0$ of an option with payoff $G(S(T))$ where $S(T)$ is the underlying price at expiry is given by the integral

$$\int_{-\infty}^{\infty} e^{-rT} G(S_0 e^{\xi}) \frac{1}{\sqrt{2\pi\sigma^2 T}} \exp\left(-\frac{(\xi - \nu T)^2}{2\sigma^2 T}\right) d\xi$$

where S_0 is the price of the underlying at $t = 0$. It can be seen that the above integral is of the form shown at the beginning of this section, namely, the pricing function is

$$f(x) = e^{-rT} G(S_0 e^{x})$$

and can be interpreted as the option payoff discounted to the present time, while the probability distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2 T}} \exp\left(-\frac{(x - \nu T)^2}{2\sigma^2 T}\right).$$

which again shows that the logarithms of the underlying prices at time T are distributed as a Gaussian with average νT and standard deviation $\sigma\sqrt{T}$.

Classes

- class BrownianBridge

  Builds Wiener process paths using Gaussian variates.

- class MonteCarloModel

  General purpose Monte Carlo model for path samples.

- class MultiPath

  Correlated multiple asset paths.

- class MultiPathGenerator

  Generates a multipath from a random number generator.

- class Path
- class PathGenerator

  Generates random paths using a sequence generator.

- class PathPricer

  base class for path pricers

- struct Sample

  weighted sample

6.22 Design patterns

Classes

- class Bridge

  The Bridge pattern made explicit.

- class Composite

  Composite pattern.

- class CuriouslyRecurringTemplate

  Support for the curiously recurring template pattern.

- class LazyObject

  Framework for calculation on demand and result caching.

- class Observable

  Object that notifies its changes to a set of observables.

- class Observer

  Object that gets notified when a given observable changes.

- class Singleton

  Basic support for the singleton pattern.

- class AcyclicVisitor

  degenerate base class for the Acyclic Visitor pattern

6.23 Term structures

6.23.1 Detailed Description

The abstract class QuantLib::YieldTermStructure provides the common interface to concrete yield-rate term structure models. Among others, methods are declared which return instantaneous forward rate, discount factor, and zero rate at a given date. Adapter classes are provided which already implement part of the required methods, thus allowing the programmer to define only the non-redundant part.

Classes

- class InterpolatedDiscountCurve

  Term structure based on interpolation of discount factors.

- class FlatForward

  Flat interest-rate curve.

- class InterpolatedForwardCurve

  Term structure based on interpolation of forward rates.

- class ForwardSpreadedTermStructure

  Term structure with added spread on the instantaneous forward rate.

- class ForwardRateStructure

  Forward rate term structure.

- class ImpliedTermStructure

  Implied term structure at a given date in the future.

- class PiecewiseYieldCurve

  Piecewise yield term structure.

- class InterpolatedZeroCurve

  Term structure based on interpolation of zero yields.

- class ZeroSpreadedTermStructure

  Term structure with an added spread on the zero yield rate.

- class ZeroYieldStructure

  Zero-yield term structure.

- class YieldTermStructure

  Interest-rate term structure.

Typedefs

- typedef InterpolatedDiscountCurve< LogLinear > QuantLib::DiscountCurve
  Term structure based on log-linear interpolation of discount factors.
- typedef InterpolatedForwardCurve< BackwardFlat > QuantLib::ForwardCurve
  Term structure based on flat interpolation of forward rates.
- typedef PiecewiseYieldCurve< Discount, LogLinear > QuantLib::PiecewiseFlatForward
  Piecewise flat-forward term structure.
- typedef InterpolatedZeroCurve< Linear > QuantLib::ZeroCurve
  Term structure based on linear interpolation of zero yields.

6.23.2 Typedef Documentation

6.23.2.1 typedef InterpolatedDiscountCurve<LogLinear> DiscountCurve

Term structure based on log-linear interpolation of discount factors.

Log-linear interpolation guarantees piecewise-constant forward rates.

6.24 Utilities

Iterators are meant to build a sequence on the fly from one or more other sequences, without having to allocate place for storing it. A couple of examples: suppose we have a function which calculates the average of a sequence, and that for genericity we have implemented it as a template function which takes the beginning and the end of the sequence, so that its declaration is:

```
template <class Iterator>
typename Iterator::value_type
average(const Iterator& begin, const Iterator& end)
```

This kind of genericity allows one to use the same function to calculate the average of a std::vector, a std::list, a QuantLib::History, any other container, of a subset of any of the former.

Now let's say we have two sequences of numbers, and we want to calculate the average of their products. One approach could be to store the products in another sequence, and to calculate the average of the latter, as in:

```
// we have sequence1 and sequence2 and assume equal size:
// first we store their product in a vector...
std::vector<double> products;
std::transform(sequence1.begin(),sequence1.end(), // first sequence
               sequence2.begin(),                 // second sequence
               std::back_inserter(products),      // output
               std::multiplies<double>());        // operation to perform
// ...then we calculate the average
double result = average(products.begin(),products.end());
```

The above works, however, it might be not particularly efficient since we have to allocate the product vector, quite possibly just to throw it away when the calculation is done.

QuantLib::coupling_iterator allows us to do the same thing without allocating the extra vector: what we do is simply:

```
// we have sequence1 and sequence2 and assume equal size:
double result = average(
    make_coupling_iterator(sequence1.begin(),
                           sequence2.begin(),
                           std::multiplies<double()),
    make_coupling_iterator(sequence1.end(),
                           sequence2.end(),
                           std::multiplies<double()));
```

The call to make_coupling_iterator creates an iterator which is really a reference to the two iterators and the operation we passed. Dereferencing such iterator returns the result of applying such operation to the values pointed to by the two contained iterators. Advancing the coupling iterator advances the two underlying ones. One can see how iterating on such iterator generates the products one by one so that they can be processed by `average()`, but does not need allocating memory for storing the results. The product sequence is generated on the fly.

The other iterators share the same principle but have different functionalities:

- combining_iterator is the same as coupling_iterator, but works on N sequences while the latter works on 2;
- filtering_iterator generates the elements of a given sequence which satisfy a given predicate, i.e., it takes a sequence $[x_0, x_1, \ldots]$ and a predicate p and generates the sequence of those x_i for which $p(x_i)$ returns `true`;

- processing_iterator takes a sequence $[x_0, x_1, \ldots]$ and a function f and generates the sequence $[f(x_0), f(x_1), \ldots]$;
- stepping_iterator takes a sequence $[x_0, x_1, \ldots]$ and a step m and generates the sequence $[x_0, x_m, x_{2m}, \ldots]$

6.25 QuantLib macros

6.25.1 Detailed Description

Global definitions and a few macros which help porting the code to different compilers.

Modules

- Generic macros
- Debugging macros

Defines

- #define QL_VERSION "0.3.11"

  version string

- #define QL_HEX_VERSION 0x000311f0

  version hexadecimal number

- #define QL_LIB_VERSION "0_3_11"

  version string for output lib name

6.26 Generic macros

6.26.1 Detailed Description

Miscellaneous macros for compiler idiosyncrasies not fitting other categories.

Defines

- #define QL_DUMMY_RETURN(x)
  Is a dummy return statement required?
- #define QL_IO_INIT
  I/O initialization.

6.26.2 Define Documentation

6.26.2.1 #define QL_DUMMY_RETURN(x)

Is a dummy return statement required?

Some compilers will issue a warning if it is missing even though it could never be reached during execution, e.g., after a block like

```
if (condition)
    return validResult;
else
    QL_FAIL("whatever the reason");
```

On the other hand, other compilers will issue a warning if it is present because it cannot be reached. For the code to be portable this macro should be used after the block.

6.26.2.2 #define QL_IO_INIT

I/O initialization.

Sometimes, programs compiled with the free Borland compiler will crash miserably upon attempting to write on std::cout. Strangely enough, issuing the instruction

```
std::cout << std::string();
```

at the beginning of the program will prevent other accesses to `std::cout` from crashing the program. This macro, to be called at the beginning of `main()`, encapsulates the above enchantment for Borland and is defined as empty for the other compilers.

Examples:
AmericanOption.cpp, BermudanSwaption.cpp, DiscreteHedging.cpp, EuropeanOption.cpp, and swapvaluation.cpp.

6.27 Numeric limits

6.27.1 Detailed Description

Some compilers do not give an implementation of <limits> yet. For the code to be portable these macros should be used instead of the corresponding method of std::numeric_limits or the corresponding macro defined in <limits.h>.

Defines

- #define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())
- #define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())
- #define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())
- #define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())
- #define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())
- #define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())
- #define **QL_NULL_INTEGER** ((std::numeric_limits<int>::max)())
- #define **QL_NULL_REAL** ((std::numeric_limits<float>::max)())

6.27.2 Define Documentation

6.27.2.1 #define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())

Defines the value of the largest representable negative integer value

6.27.2.2 #define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())

Defines the value of the largest representable integer value

6.27.2.3 #define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())

Defines the value of the largest representable negative floating-point value

6.27.2.4 #define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())

Defines the value of the smallest representable positive double value

6.27.2.5 #define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())

Defines the value of the largest representable floating-point value

6.27.2.6 #define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())

Defines the machine precision for operations over doubles

6.28 Template capabilities

6.28.1 Detailed Description

Some compilers still do not fully implement the template syntax. These macros can be used to select between alternate implementations of blocks of code, namely, one that takes advantage of template programming techniques and a less efficient one which is compatible with all compilers.

Defines

- #define QL_TYPENAME typename

6.28.2 Define Documentation

6.28.2.1 #define QL_TYPENAME typename

In Visual C++ 6, typename can only be used in template declarations and not in template definitions.

6.29 Iterator support

6.29.1 Detailed Description

Some compilers still define the iterator struct outside the std namespace, only partially implement it, or do not implement it at all. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define QL_FULL_ITERATOR_SUPPORT

6.29.2 Define Documentation

6.29.2.1 #define QL_FULL_ITERATOR_SUPPORT

Some compilers (most notably, Visual C++ 6) still do not fully support iterators in their STL implementation. This macro can be used to select between alternate implementations of blocks of code, namely, one that takes advantage of full iterator support and a less efficient one which is compatible with all compilers.

6.30 Output manipulators

6.30.1 Detailed Description

Helper functions for creating formatted output.

Functions

- detail::long_weekday_holder QuantLib::io::long_weekday (Weekday)
  output weekdays in long format
- detail::short_weekday_holder QuantLib::io::short_weekday (Weekday)
  output weekdays in short format (three letters)
- detail::shortest_weekday_holder QuantLib::io::shortest_weekday (Weekday)
  output weekdays in shortest format (two letters)
- detail::long_period_holder QuantLib::io::long_period (const Period &)
  output periods in long format (e.g. "2 weeks")
- detail::short_period_holder QuantLib::io::short_period (const Period &)
  output periods in short format (e.g. "2w")
- detail::short_date_holder QuantLib::io::short_date (const Date &)
  output dates in short format (mm/dd/yyyy)
- detail::long_date_holder QuantLib::io::long_date (const Date &)
  output dates in long format (Month ddth, yyyy)
- detail::iso_date_holder QuantLib::io::iso_date (const Date &)
  output dates in ISO format (yyyy-mm-dd)
- template<typename T> detail::null_checker< T > QuantLib::io::checknull (T)
  check for nulls before output
- detail::ordinal_holder QuantLib::io::ordinal (Size)
  outputs naturals as 1st, 2nd, 3rd...
- template<typename T> detail::power_of_two_holder< T > QuantLib::io::power_of_two (T)

  output integers as powers of two
- detail::percent_holder QuantLib::io::percent (Real)
  output reals as percentages
- detail::percent_holder QuantLib::io::rate (Rate)
  output rates and spreads as percentages
- detail::percent_holder QuantLib::io::volatility (Volatility)
  output volatilities as percentages

6.31 Debugging macros

6.31.1 Detailed Description

For debugging purposes, macros can be used to output information about the code being executed.

Defines

- #define QL_TRACE_ENABLE
  enable tracing
- #define QL_TRACE_DISABLE
  disable tracing
- #define QL_TRACE_ON(out)
  set tracing stream
- #define QL_TRACE(message)
  output tracing information
- #define QL_TRACE_ENTER_FUNCTION
  output tracing information
- #define QL_TRACE_EXIT_FUNCTION
  output tracing information
- #define QL_TRACE_LOCATION
  output tracing information
- #define QL_TRACE_VARIABLE(variable)
  output tracing information

6.31.2 Define Documentation

6.31.2.1 #define QL_TRACE_ENABLE

enable tracing

The statement

```
QL_TRACE_ENABLE;
```

can be used to enable tracing. Such statement might be ignored; refer to QL_TRACE for details.

Examples:
tracing_example.cpp.

6.31.2.2 #define QL_TRACE_DISABLE

disable tracing

The statement

```
QL_TRACE_DISABLE;
```

can be used to disable tracing. Such statement might be ignored; refer to QL_TRACE for details.

6.31.2.3 #define QL_TRACE_ON(out)

set tracing stream

The statement

```
QL_TRACE_ON(stream);
```

can be used to set the stream where tracing messages are output. Such statement might be ignored; refer to QL_TRACE for details.

6.31.2.4 #define QL_TRACE(message)

output tracing information

The statement

```
QL_TRACE(message);
```

can be used to output a trace of the code being executed. If tracing was disabled during configuration, such statements are removed by the preprocessor for maximum performance; if it was enabled, whether and where the message is output depends on the current settings.

Examples:
tracing_example.cpp.

6.31.2.5 #define QL_TRACE_ENTER_FUNCTION

output tracing information

The statement

```
QL_TRACE_ENTER_FUNCTION;
```

can be used at the beginning of a function to trace the fact that the program execution is entering such function. It should be paired with a corresponding QL_TRACE_EXIT_FUNCTION macro. Such statement might be ignored; refer to QL_TRACE for details. Also, function information might not be available depending on the compiler.

Examples:
tracing_example.cpp.

6.31.2.6 #define QL_TRACE_EXIT_FUNCTION

output tracing information

The statement

```
QL_TRACE_EXIT_FUNCTION;
```

can be used before returning from a function to trace the fact that the program execution is exiting such function. It should be paired with a corresponding QL_TRACE_ENTER_FUNCTION macro. Such statement might be ignored; refer to QL_TRACE for details. Also, function information might not be available depending on the compiler.

Examples:
tracing_example.cpp.

6.31.2.7 #define QL_TRACE_LOCATION

output tracing information

The statement

```
QL_TRACE_LOCATION;
```

can be used to trace the current file and line. Such statement might be ignored; refer to QL_TRACE for details.

Examples:
tracing_example.cpp.

6.31.2.8 #define QL_TRACE_VARIABLE(variable)

output tracing information

The statement

```
QL_TRACE_VARIABLE(variable);
```

can be used to trace the current value of a variable. Such statement might be ignored; refer to QL_TRACE for details. Also, the variable type must allow sending it to an output stream.

Examples:
tracing_example.cpp.

Chapter 7

QuantLib Class Documentation

7.1 Actual360 Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/actual360.hpp>`

Inheritance diagram for Actual360:

7.1.1 Detailed Description

Actual/360 day count convention.

Actual/360 day count convention, also known as "Act/360", or "A/360".

7.2 Actual365Fixed Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/actual365fixed.hpp>`

Inheritance diagram for Actual365Fixed:

7.2.1 Detailed Description

Actual/365 (Fixed) day count convention.

"Actual/365 (Fixed)" day count convention, also know as "Act/365 (Fixed)", "A/365 (Fixed)", or "A/365F".

Warning:

According to ISDA, "Actual/365" (without "Fixed") is an alias for "Actual/Actual (ISDA)" (see ActualActual.) If Actual/365 is not explicitly specified as fixed in an instrument specification, you might want to double-check its meaning.

7.3 ActualActual Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/actualactual.hpp>`

Inheritance diagram for ActualActual:

7.3.1 Detailed Description

Actual/Actual day count.

The day count can be calculated according to:

- the ISDA convention, also known as "Actual/Actual (Historical)", "Actual/Actual", "Act/Act", and according to ISDA also "Actual/365", "Act/365", and "A/365";
- the ISMA and US Treasury convention, also known as "Actual/Actual (Bond)";
- the AFB convention, also known as "Actual/Actual (Euro)".

For more details, refer to http://www.isda.org/publications/pdf/Day-Count-Fracation1999.pdf

Test
the correctness of the results is checked against known good values.

Public Types

- enum **Convention** {
  ISMA, **Bond**, **ISDA**, **Historical**,
  AFB, **Euro** }

Public Member Functions

- **ActualActual** (Convention c=ActualActual::ISDA)

7.4 AcyclicVisitor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/visitor.hpp>`

Inheritance diagram for AcyclicVisitor:

7.4.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

7.5 AdditiveEQPBinomialTree Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for AdditiveEQPBinomialTree:

7.5.1 Detailed Description

Additive equal probabilities binomial tree.

Public Member Functions

- **AdditiveEQPBinomialTree** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)

7.6 AffineModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/model.hpp>`

Inheritance diagram for AffineModel:

7.6.1 Detailed Description

Affine model class.

Base class for analytically tractable models.

Public Member Functions

- virtual DiscountFactor discount (Time t) const =0

  Implied discount curve.

- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const =0

7.7 AffineTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/affinetermstructure.hpp>`

Inheritance diagram for AffineTermStructure:

7.7.1 Detailed Description

Term-structure implied by an affine model.

This class defines a term-structure that is based on an affine model, e.g. Vasicek or Cox-Ingersoll-Ross. It either be instanced using a model with defined arguments, or the model can be calibrated to a set of rate helpers. Of course, there is no point in using a term-structure consistent affine model, since the implied term-structure will just be the initial term-structure on which the model is based.

Public Member Functions

- AffineTermStructure (const Date &referenceDate, const boost::shared_ptr< AffineModel > &model, const DayCounter &dayCounter)

  constructor using a fixed model

- AffineTermStructure (const Date &referenceDate, const boost::shared_ptr< AffineModel > &model, const std::vector< boost::shared_ptr< RateHelper > > &, const boost::shared_ptr< OptimizationMethod > &, const DayCounter &dayCounter)

  constructor using a model that has to be calibrated

- AffineTermStructure (Integer settlementDays, const Calendar &calendar, const boost::shared_ptr< AffineModel > &model, const DayCounter &dayCounter)

  constructor using a fixed model

- AffineTermStructure (Integer settlementDays, const Calendar &calendar, const boost::shared_ptr< AffineModel > &model, const std::vector< boost::shared_ptr< RateHelper > > &, const boost::shared_ptr< OptimizationMethod > &, const DayCounter &dayCounter)

  constructor using a model that has to be calibrated

- DayCounter dayCounter () const

  the day counter used for date/time conversion

- Date maxDate () const

  the latest date for which the curve can return rates

- void update ()

Protected Member Functions

- DiscountFactor discountImpl (Time) const

  discount calculation

7.7.2 Member Function Documentation

7.7.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from LazyObject.

7.8 AmericanCondition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/americancondition.hpp>`

Inheritance diagram for AmericanCondition:

7.8.1 Detailed Description

American exercise condition.

Todo
unify the intrinsicValues/Payoff thing

Public Member Functions

- **AmericanCondition** (Option::Type type, Real strike)
- **AmericanCondition** (const Array &intrinsicValues)
- void **applyTo** (Array &a, Time t) const

7.9 AmericanExercise Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp>`

Inheritance diagram for AmericanExercise:

7.9.1 Detailed Description

American exercise.

An American option can be exercised at any time between two predefined dates

Todo

check that everywhere the American condition is applied from earliestDate and not earlier

Public Member Functions

- **AmericanExercise** (const Date &earliestDate, const Date &latestDate, bool payoffAtExpiry=false)

7.10 AmericanPayoffAtExpiry Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/americanpayoffatexpiry.hpp>`

7.10.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options

Todo
calculate greeks

Public Member Functions

- **AmericanPayoffAtExpiry** (Real spot, DiscountFactor discount, DiscountFactor dividendDiscount, Real variance, const boost::shared_ptr< StrikedTypePayoff > &payoff)
- Real **value** () const

7.11 AmericanPayoffAtHit Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/americanpayoffathit.hpp>`

7.11.1 Detailed Description

Analytic formula for American exercise payoff at-hit options

Todo
calculate greeks

Public Member Functions

- **AmericanPayoffAtHit** (Real spot, DiscountFactor discount, DiscountFactor dividendDiscount, Real variance, const boost::shared_ptr< StrikedTypePayoff > &payoff)
- Real **value** () const
- Real **delta** () const
- Real **gamma** () const
- Real **rho** (Time maturity) const

7.12 AnalyticBarrierEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Barrier/analyticbarrierengine.hpp>`

Inheritance diagram for AnalyticBarrierEngine:

7.12.1 Detailed Description

Pricing engine for barrier options using analytical formulae.

The formulas are taken from "Option pricing formulas", E.G. Haug, McGraw-Hill, p.69 and following.

Test
the correctness of the returned value is tested by reproducing results available in literature.

Todo
rework to avoid repeated casts inside utility methods

Public Member Functions

- void **calculate** () const

7.13 AnalyticCapFloorEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp>`

Inheritance diagram for AnalyticCapFloorEngine:

7.13.1 Detailed Description

Analytic engine for cap/floor.

Public Member Functions

- **AnalyticCapFloorEngine** (const boost::shared_ptr< AffineModel > &model)
- void **calculate** () const

7.14 AnalyticCliquetEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Cliquet/analyticcliquetengine.hpp>`

Inheritance diagram for AnalyticCliquetEngine:

7.14.1 Detailed Description

Pricing engine for Cliquet options using analytical formulae.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- void **calculate** () const

7.15 AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Asian/analytic_cont_geom_av_price.hpp>`

Inheritance diagram for AnalyticContinuousGeometricAveragePriceAsianEngine:

7.15.1 Detailed Description

Pricing engine for European continuous geometric average price Asian.

This class implements a continuous geometric average price Asian option with European exercise. The formula is from "Option Pricing Formulas", E. G. Haug (1997) pag 96-97.

Test

- the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Todo

handle seasoned options

Public Member Functions

- void **calculate** () const

7.16 AnalyticDigitalAmericanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticdigitalamericanengine.hpp>`

Inheritance diagram for AnalyticDigitalAmericanEngine:

7.16.1 Detailed Description

Pricing engine for American vanilla options with digital payoff using analytic formulae

Todo

add more greeks (as of now only delta and rho available)

Test

- the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

Public Member Functions

- void **calculate** () const

7.17 AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Asian/analytic_discr_geom_av_price.hpp>`

Inheritance diagram for AnalyticDiscreteGeometricAveragePriceAsianEngine:

7.17.1 Detailed Description

Pricing engine for European discrete geometric average price Asian.

This class implements a discrete geometric average price Asian option, with European exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag 65-97

Todo

implement correct theta, rho, and dividend-rho calculation

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the available greeks is tested against numerical calculations.

Public Member Functions

- void **calculate** () const

7.18 AnalyticDividendEuropeanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticdividendeuropeanengine.hpp>`

Inheritance diagram for AnalyticDividendEuropeanEngine:

7.18.1 Detailed Description

Analytic pricing engine for European options with discrete dividends.

Test
the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- void **calculate** () const

7.19 AnalyticEuropeanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticeuropeanengine.hpp>`

Inheritance diagram for AnalyticEuropeanEngine:

7.19.1 Detailed Description

Pricing engine for European vanilla options using analytical formulae.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the implied-volatility calculation is tested by checking that it does not modify the option.
- the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Public Member Functions

- void **calculate** () const

7.20 AnalyticHestonEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analytichestonengine.hpp>`

Inheritance diagram for AnalyticHestonEngine:

7.20.1 Detailed Description

analytic Heston-model engine based on Fourier transform

References:

Heston, Steven L., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The review of Financial Studies, Volume 6, Issue 2, 327-343.

Dupire, Bruno, 1994. Pricing with a smile. Risk Magazine, 7, 18-20.

A. Sepp, Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

Test

the correctness of the returned value is tested by reproducing results available in web/literature and comparison with Black pricing.

Public Member Functions

- **AnalyticHestonEngine** (const boost::shared_ptr< HestonModel > &model, Size integrationOrder=64)
- void **calculate** () const
- virtual std::complex< Real > **jumpDiffusionTerm** (Real phi, Time t, Size j) const

7.21 AnalyticPerformanceEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Cliquet/analyticperformanceengine.hpp>`

Inheritance diagram for AnalyticPerformanceEngine:

7.21.1 Detailed Description

Pricing engine for performance options using analytical formulae.

Test
the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- void **calculate** () const

7.22 Arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/argsandresults.hpp>`

Inheritance diagram for Arguments:

7.22.1 Detailed Description

base class for generic argument groups

Public Member Functions

- virtual void **validate** () const =0

7.23 ArmijoLineSearch Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/armijo.hpp>`

Inheritance diagram for ArmijoLineSearch:

7.23.1 Detailed Description

Armijo line search.

Let α and β be 2 scalars in $[0,1]$. Let x be the current value of the unknown, d the search direction and t the step. Let f be the function to minimize. The line search stops when t verifies

$$f(x + t \cdot d) - f(x) \leq -\alpha t f'(x + t \cdot d)$$

and

$$f(x + \frac{t}{\beta} \cdot d) - f(x) > -\frac{\alpha}{\beta} t f'(x + t \cdot d)$$

(see Polak. Algorithms and consistent approximations, Optimization, volume 124 of Applied Mathematical Sciences. Springer-verlag, N-Y, 1997)

Public Member Functions

- ArmijoLineSearch (Real eps=1e-8, Real alpha=0.05, Real beta=0.65)
  Default constructor.
- virtual Real operator() (const Problem &P, Real t_ini)
  Perform line search.

7.24 Array Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/array.hpp>`

7.24.1 Detailed Description

1-D array used in linear algebra.

This class implements the concept of vector as used in linear algebra. As such, it is **not** meant to be used as a container - `std::vector` should be used instead.

Test
construction of arrays is checked in a number of cases

Public Types

- typedef Real * **iterator**
- typedef const Real * **const_iterator**
- typedef boost::reverse_iterator< iterator > **reverse_iterator**
- typedef boost::reverse_iterator< const_iterator > **const_reverse_iterator**

Public Member Functions

Constructors, destructor, and assignment

- Array (Size size=0)
  creates the array with the given dimension
- Array (Size size, Real value)
  creates the array and fills it with `value`
- Array (Size size, Real value, Real increment)
  creates the array and fills it according to $a_0 = value, a_i = a_{i-1} + increment$
- **Array** (const Array &)
- **Array** (const Disposable< Array > &)
- Array & **operator=** (const Array &)
- Array & **operator=** (const Disposable< Array > &)

Vector algebra

`v += x` *and similar operation involving a scalar value are shortcuts for* $\forall i : v_i = v_i + x$

`v *= w` *and similar operation involving two vectors are shortcuts for* $\forall i : v_i = v_i \times w_i$

Precondition:
all arrays involved in an algebraic expression must have the same size.

- const Array & **operator+=** (const Array &)
- const Array & **operator+=** (Real)
- const Array & **operator-=** (const Array &)
- const Array & **operator-=** (Real)
- const Array & **operator *=** (const Array &)

- const Array & **operator** *= (Real)
- const Array & **operator/=** (const Array &)
- const Array & **operator/=** (Real)

Element access

- Real operator[ ] (Size) const
  read-only
- Real & operator[ ] (Size)
  read-write

Inspectors

- Size size () const
  dimension of the array
- bool empty () const
  whether the array is empty

Iterator access

- const_iterator **begin** () const
- iterator **begin** ()
- const_iterator **end** () const
- iterator **end** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rend** () const
- reverse_iterator **rend** ()

Utilities

- void **swap** (Array &)

Related Functions

(Note that these are not member functions.)

- Real **DotProduct** (const Array &, const Array &)
- const Disposable< Array > **operator+** (const Array &v)
- const Disposable< Array > **operator-** (const Array &v)
- const Disposable< Array > **operator+** (const Array &, const Array &)
- const Disposable< Array > **operator+** (const Array &, Real)
- const Disposable< Array > **operator+** (Real, const Array &)
- const Disposable< Array > **operator-** (const Array &, const Array &)
- const Disposable< Array > **operator-** (const Array &, Real)
- const Disposable< Array > **operator-** (Real, const Array &)
- const Disposable< Array > **operator** * (const Array &, const Array &)
- const Disposable< Array > **operator** * (const Array &, Real)
- const Disposable< Array > **operator** * (Real, const Array &)

- const Disposable< Array > **operator/** (const Array &, const Array &)
- const Disposable< Array > **operator/** (const Array &, Real)
- const Disposable< Array > **operator/** (Real, const Array &)
- const Disposable< Array > **Abs** (const Array &)
- const Disposable< Array > **Sqrt** (const Array &)
- const Disposable< Array > **Log** (const Array &)
- const Disposable< Array > **Exp** (const Array &)
- void **swap** (Array &, Array &)
- std::ostream & **operator**<< (std::ostream &, const Array &)

7.25 ARSCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for ARSCurrency:

7.25.1 Detailed Description

Argentinian peso.

The ISO three-letter code is ARS; the numeric code is 32. It is divided in 100 centavos.

7.26 AssetOrNothingPayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for AssetOrNothingPayoff:

7.26.1 Detailed Description

Binary asset-or-nothing payoff.

Public Member Functions

- **AssetOrNothingPayoff** (Option::Type type, Real strike)
- Real **operator()** (Real price) const

7.27 ATSCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for ATSCurrency:

7.27.1 Detailed Description

Austrian shilling.

The ISO three-letter code was ATS; the numeric code was 40. It was divided in 100 groschen.

7.28 AUDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/oceania.hpp>`

Inheritance diagram for AUDCurrency:

7.28.1 Detailed Description

Australian dollar.

The ISO three-letter code is AUD; the numeric code is 36. It is divided into 100 cents.

7.29 AUDLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/audlibor.hpp>`

Inheritance diagram for AUDLibor:

7.29.1 Detailed Description

AUD LIBOR rate

Australian Dollar LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Public Member Functions

- **AUDLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.30 Average Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

7.30.1 Detailed Description

placeholder for enumerated averaging types

Public Types

- enum **Type** { **Arithmetic**, **Geometric** }

7.31 BackwardFlat Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/backwardflatinterpolation.hpp>`

7.31.1 Detailed Description

Backward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

Public Member Functions

- template<class I1, class I2> Interpolation **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

7.32 BackwardFlatInterpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/backwardflatinterpolation.hpp>`

Inheritance diagram for BackwardFlatInterpolation:

7.32.1 Detailed Description

Backward-flat interpolation between discrete points.

Public Member Functions

- template<class I1, class I2> BackwardFlatInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.32.2 Constructor & Destructor Documentation

7.32.2.1 BackwardFlatInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:
the x values must be sorted.

7.33 BaroneAdesiWhaleyApproximationEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/baroneadesiwhaleyengine.hpp>`

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:

7.33.1 Detailed Description

Pricing engine for American options with Barone-Adesi and Whaley approximation (1987)

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Member Functions

- void **calculate** () const

Static Public Member Functions

- static Real **criticalPrice** (const boost::shared_ptr< StrikedTypePayoff > &payoff, DiscountFactor riskFreeDiscount, DiscountFactor dividendDiscount, Real variance, Real tolerance=1e-6)

7.34 Barrier Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/barrieroption.hpp>`

7.34.1 Detailed Description

Placeholder for enumerated barrier types.

Public Types

- enum **Type** { **DownIn**, **UpIn**, **DownOut**, **UpOut** }

7.35 BarrierOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/barrieroption.hpp>`

Inheritance diagram for BarrierOption:

7.35.1 Detailed Description

Barrier option on a single asset.

The analytic pricing engine will be used if none if passed.

Public Member Functions

- **BarrierOption** (Barrier::Type barrierType, Real barrier, Real rebate, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Protected Member Functions

- void performCalculations () const

Protected Attributes

- Barrier::Type **barrierType_**
- Real **barrier_**
- Real **rebate_**

Classes

- class arguments

  Arguments for barrier option calculation

- class engine

  Barrier engine base class

7.35.2 Member Function Documentation

7.35.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.35.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from OneAssetStrikedOption.

7.36 BarrierOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/barrieroption.hpp>`

7.36.1 Detailed Description

Arguments for barrier option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Barrier::Type **barrierType**
- Real **barrier**
- Real **rebate**

7.37 BarrierOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/barrieroption.hpp>`

Inheritance diagram for BarrierOption::engine:

7.37.1 Detailed Description

Barrier engine base class

7.38 BasketOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/basketoption.hpp>`

Inheritance diagram for BasketOption:

7.38.1 Detailed Description

Basket option on a number of assets.

Public Types

- enum **BasketType** { **Min**, **Max** }

Public Member Functions

- **BasketOption** (const BasketType basketType, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< PlainVanillaPayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Classes

- class arguments

  Arguments for basket option calculation

- class engine

  Basket option engine base class

7.38.2 Member Function Documentation

7.38.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from MultiAssetOption.

7.39 BasketOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/basketoption.hpp>`

Inheritance diagram for BasketOption::arguments:

7.39.1 Detailed Description

Arguments for basket option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- BasketType **basketType**

7.40 BasketOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/basketoption.hpp>`

Inheritance diagram for BasketOption::engine:

7.40.1 Detailed Description

Basket option engine base class

7.41 BatesEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/batesengine.hpp>`

Inheritance diagram for BatesEngine:

7.41.1 Detailed Description

Bates model engines based on Fourier transform.

this classes price european options under the following processes

1. Jump-Diffusion with Stochastic Volatility

$$\begin{aligned} dS(t,S) &= (r-d-\lambda m)Sdt + \sqrt{v}SdW_1 + (e^J-1)SdN \\ dv(t,S) &= \kappa(\theta - v)dt + \sigma\sqrt{v}dW_2 \\ dW_1dW_2 &= \rho dt \end{aligned}$$

N is a Poisson process with the intensity λ . When a jump occurs the magnitude J has the probability distribution function $\omega(J)$.

1.1 Log-Normal Jump Diffusion: BatesEngine

Logarithm of the jump size J is normally distributed

$$\omega(J) = \frac{1}{\sqrt{2\pi\delta^2}}\exp\left[-\frac{(J-\nu)^2}{2\delta^2}\right]$$

1.2 Double-Exponential Jump Diffusion: BatesDoubleExpEngine

The jump size has an asymmetric double exponential distribution

$$\begin{aligned} \omega(J) &= p\tfrac{1}{\eta_u}e^{-\frac{1}{\eta_u}J}1_{J>0} + q\tfrac{1}{\eta_d}e^{\frac{1}{\eta_d}J}1_{J<0} \\ p+q &= 1 \end{aligned}$$

2. Stochastic Volatility with Jump Diffusion and Deterministic Jump Intensity

$$\begin{aligned} dS(t,S) &= (r-d-\lambda m)Sdt + \sqrt{v}SdW_1 + (e^J-1)SdN \\ dv(t,S) &= \kappa(\theta - v)dt + \sigma\sqrt{v}dW_2 \\ d\lambda(t) &= \kappa_\lambda(\theta_\lambda - \lambda)dt \\ dW_1dW_2 &= \rho dt \end{aligned}$$

2.1 Log-Normal Jump Diffusion with Deterministic Jump Intensity BatesDetJumpEngine

2.2 Double-Exponential Jump Diffusion with Deterministic Jump Intensity BatesDoubleExpDet-JumpEngine

References:

D. Bates, Jumps and stochastic volatilit: exchange rate processes implicit in Deutsche mark options", Review of Financial Sudies 9, 69-107.

A. Sepp, Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

Test

the correctness of the returned value is tested by reproducing results available in web/literature, testing against QuantLib's jump diffusion engine and comparison with Black pricing.

Public Member Functions

- **BatesEngine** (const boost::shared_ptr< BatesModel > &model, Size integrationOrder=64)

Protected Member Functions

- std::complex< Real > **jumpDiffusionTerm** (Real phi, Time t, Size j) const

7.42 BatesModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/batesmodel.hpp>`

Inheritance diagram for BatesModel:

7.42.1 Detailed Description

extended versions of Heston model for the stochastic volatility of an asset including jumps.

References: A. Sepp, Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

Test
calibration is tested against known values.

Public Member Functions

- **BatesModel** (const boost::shared_ptr< HestonProcess > &process, Real lambda=0.1, Real nu=0.0, Real delta=0.1)
- Real **nu** () const
- Real **delta** () const
- Real **lambda** () const

7.43 BDTCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for BDTCurrency:

7.43.1 Detailed Description

Bangladesh taka.

The ISO three-letter code is BDT; the numeric code is 50. It is divided in 100 paisa.

7.44 BEFCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for BEFCurrency:

7.44.1 Detailed Description

Belgian franc.

The ISO three-letter code is BEF; the numeric code is 56. It has no subdivisions.

7.45 Beijing Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/beijing.hpp>`

Inheritance diagram for Beijing:

7.45.1 Detailed Description

Beijing calendar
Holidays:

- Saturdays
- Sundays
- New Year's day, January 1st
- Labour Day, first week in May
- National Day, one week from October 1st

Other holidays for which no rule is given:

- Lunar New Year (data available for 2004 only)
- Spring Festival
- Last day of Lunar Year

7.46 BermudanExercise Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp>`

Inheritance diagram for BermudanExercise:

7.46.1 Detailed Description

Bermudan exercise.

A Bermudan option can only be exercised at a set of fixed dates.

Todo

it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

Public Member Functions

- **BermudanExercise** (const std::vector< Date > &dates, bool payoffAtExpiry=false)

7.47 BGLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for BGLCurrency:

7.47.1 Detailed Description

Bulgarian lev.

The ISO three-letter code is BGL; the numeric code is 100. It is divided in 100 stotinki.

7.48 Bicubic Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bicubicsplineinterpolation.hpp>`

7.48.1 Detailed Description

bicubic-spline interpolation factory

Public Member Functions

- template<class I1, class I2, class M> Interpolation2D **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z) const

7.49 BicubicSpline Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bicubicsplineinterpolation.hpp>`

Inheritance diagram for BicubicSpline:

7.49.1 Detailed Description

bicubic-spline interpolation between discrete points

Todo
revise end conditions

Public Member Functions

- template<class I1, class I2, class M> BicubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)

7.49.2 Constructor & Destructor Documentation

7.49.2.1 BicubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, const I2 & *yEnd*, const M & *zData*)

Precondition:
the x and y values must be sorted.

7.50 Bilinear Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bilinearinterpolation.hpp>`

7.50.1 Detailed Description

bilinear interpolation factory

Public Member Functions

- template<class I1, class I2, class M> Interpolation2D **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z) const

7.51 BilinearInterpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bilinearinterpolation.hpp>`

Inheritance diagram for BilinearInterpolation:

7.51.1 Detailed Description

bilinear interpolation between discrete points

Public Member Functions

- template<class I1, class I2, class M> BilinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)

7.51.2 Constructor & Destructor Documentation

7.51.2.1 BilinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, const I2 & *yEnd*, const M & *zData*)

Precondition:
the x and y values must be sorted.

7.52 BinomialDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/binomialdistribution.hpp>`

7.52.1 Detailed Description

Binomial probability distribution function.

formula here ... Given an integer k it returns its probability in a Binomial distribution with parameters p and n.

Public Member Functions

- **BinomialDistribution** (Real p, BigNatural n)
- Real **operator()** (BigNatural k) const

7.53 BinomialTree Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for BinomialTree:

7.53.1 Detailed Description

template<class T> class QuantLib::BinomialTree< T >

Binomial tree base class.

Public Types

- enum { **branches** = 2 }

Public Member Functions

- **BinomialTree** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps)
- Size **size** (Size i) const
- Size **descendant** (Size, Size index, Size branch) const

Protected Attributes

- Real **x0_**
- Real **driftPerStep_**
- Time **dt_**

7.54 BinomialVanillaEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/binomialengine.hpp>`

Inheritance diagram for BinomialVanillaEngine:

7.54.1 Detailed Description

template<class T> class QuantLib::BinomialVanillaEngine< T >

Pricing engine for vanilla options using binomial trees.

Test
the correctness of the returned value is tested by checking it against analytic results.

Public Member Functions

- **BinomialVanillaEngine** (Size timeSteps)
- void **calculate** () const

7.55 Bisection Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/bisection.hpp>`

Inheritance diagram for Bisection:

7.55.1 Detailed Description

Bisection 1-D solver

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.56 BivariateCumulativeNormalDistributionDr78 Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bivariatenormaldistribution.hpp>`

7.56.1 Detailed Description

Cumulative bivariate normal distribution function.

Drezner (1978) algorithm, six decimal places accuracy.

For this implementation see "Option pricing formulas", E.G. Haug, McGraw-Hill 1998

Todo

check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Test

the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **BivariateCumulativeNormalDistributionDr78** (Real rho)
- Real **operator()** (Real a, Real b) const

7.57 BivariateCumulativeNormalDistributionWe04DP Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bivariatenormaldistribution.hpp>`

7.57.1 Detailed Description

Cumulative bivariate normal distibution function (West 2004).

The implementation derives from the article "Better Approximations To Cumulative Normal Distibutions", Graeme West, Dec 2004 available at www.finmod.co.za. Also available in Wilmott Magazine, 2005, (May), 70-76, The main code is a port of the C++ code at www.finmod.co.za/cumfunctions.zip.

The algorithm is based on the near double-precision algorithm described in "Numerical Computation of Rectangular Bivariate an Trivariate Normal and t Probabilities", Genz (2004), Statistics and Computing 14, 151-160. (available at www.sci.wsu.edu/math/faculty/henz/homepage)

The QuantLib implementation mainly differs from the original code in two regards;

- The implementation of the cumulative normal distribution is QuantLib::CumulativeNormalDistribution
- The arrays XX and W are zero-based

Test
the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **BivariateCumulativeNormalDistributionWe04DP** (Real rho)
- Real **operator()** (Real a, Real b) const

7.58 BjerksundStenslandApproximationEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/bjerksundstenslandengine.hpp>`

Inheritance diagram for BjerksundStenslandApproximationEngine:

7.58.1 Detailed Description

Pricing engine for American options with Bjerksund and Stensland approximation (1993)

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Member Functions

- void **calculate** () const

7.59 BlackCapFloorEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/blackcapfloorengine.hpp>`

Inheritance diagram for BlackCapFloorEngine:

7.59.1 Detailed Description

Black-formula cap/floor engine.

Public Member Functions

- **BlackCapFloorEngine** (const boost::shared_ptr< BlackModel > &model)
- void **calculate** () const

7.60 BlackConstantVol Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackconstantvol.hpp>`

Inheritance diagram for BlackConstantVol:

7.60.1 Detailed Description

Constant Black volatility, no time-strike dependence.

This class implements the BlackVolatilityTermStructure interface for a constant Black volatility (no time/strike dependence).

Public Member Functions

- **BlackConstantVol** (const Date &referenceDate, Volatility volatility, const DayCounter &dayCounter)
- **BlackConstantVol** (const Date &referenceDate, const Handle< Quote > &volatility, const DayCounter &dayCounter)
- **BlackConstantVol** (Integer settlementDays, const Calendar &, Volatility volatility, const DayCounter &dayCounter)
- **BlackConstantVol** (Integer settlementDays, const Calendar &, const Handle< Quote > &volatility, const DayCounter &dayCounter)

BlackVolTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- virtual Volatility blackVolImpl (Time t, Real) const
  Black volatility calculation.

7.61 BlackFormula Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/blackformula.hpp>`

7.61.1 Detailed Description

Black-formula calculator.

Bug

When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

Public Member Functions

- **BlackFormula** (Real forward, DiscountFactor discount, Real variance, const boost::shared_ptr< StrikedTypePayoff > &payoff)
- Real **value** () const
- Real **delta** (Real spot) const
- Real elasticity (Real spot) const

  Sensitivity in percent to a percent movement in the underlying.

- Real **gamma** (Real spot) const
- Real **deltaForward** () const
- Real elasticityForward () const

  Sensitivity in percent to a percent movement in the forward price.

- Real **gammaForward** () const
- Real **theta** (Real spot, Time maturity) const
- Real **thetaPerDay** (Real spot, Time maturity) const
- Real **vega** (Time maturity) const
- Real **rho** (Time maturity) const
- Real **dividendRho** (Time maturity) const
- Real itmCashProbability () const
- Real itmAssetProbability () const
- Real **strikeSensitivity** () const
- Real **alpha** () const
- Real **beta** () const

7.61.2 Member Function Documentation

7.61.2.1 Real itmCashProbability () const

Probability of being in the money in the bond martingale measure. It is a risk-neutral probability, not the real world probability.

7.61.2.2 Real itmAssetProbability () const

Probability of being in the money in the asset martingale measure. It is a risk-neutral probability, not the real world probability.

7.62 BlackKarasinski Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/blackkarasinski.hpp>`

Inheritance diagram for BlackKarasinski:

7.62.1 Detailed Description

Standard Black-Karasinski model class.

This class implements the standard Black-Karasinski model defined by

$$d \ln r_t = (\theta(t) - \alpha \ln r_t)dt + \sigma dW_t,$$

where *alpha* and *sigma* are constants.

Public Member Functions

- **BlackKarasinski** (const Handle< YieldTermStructure > &termStructure, Real a=0.1, Real sigma=0.1)
- boost::shared_ptr< ShortRateDynamics > dynamics () const

  returns the short-rate dynamics

- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Return by default a trinomial recombining tree.

Classes

- class Dynamics

  Short-rate dynamics in the Black-Karasinski model.

7.63 BlackKarasinski::Dynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/blackkarasinski.hpp>`

7.63.1 Detailed Description

Short-rate dynamics in the Black-Karasinski model.

The short-rate is here

$$r_t = e^{\varphi(t)+x_t}$$

where $\varphi(t)$ is the deterministic time-dependent parameter (which can not be determined analytically) used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

Public Member Functions

- **Dynamics** (const Parameter &fitting, Real alpha, Real sigma)
- Real **variable** (Time t, Rate r) const
- Real **shortRate** (Time t, Real x) const

7.64 BlackModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/blackmodel.hpp>`

Inheritance diagram for BlackModel:

7.64.1 Detailed Description

Black-model for vanilla interest-rate derivatives.

Public Member Functions

- **BlackModel** (const Handle< Quote > &volatility, const Handle< YieldTermStructure > &termStructure)
- void update ()
- Volatility **volatility** () const
- const Handle< YieldTermStructure > & **termStructure** () const

Static Public Member Functions

- static Real formula (Real f, Real k, Real v, Real w)

  General Black formula.

- static Real itmProbability (Real f, Real k, Real v, Real w)

  In-the-money cash probability.

7.64.2 Member Function Documentation

7.64.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.64.2.2 Real formula (Real *f*, Real *k*, Real *v*, Real *w*) [static]

General Black formula.

Returns

$$Black(f,k,v,w) = fw\Phi(wd_1(f,k,v)) - kw\Phi(wd_2(f,k,v)),$$

where

$$d_1(f,k,v) = \frac{\ln(f/k) + v^2/2}{v}$$

and

$$d_2(f,k,v) = d_1(f,k,v) - v.$$

7.64.2.3 Real itmProbability (Real *f*, Real *k*, Real *v*, Real *w*) [static]

In-the-money cash probability.

Returns

$$P(f,k,v,w) = \Phi(w d_2(f,k,v)),$$

where

$$d_1(f,k,v) = \frac{\ln(f/k) + v^2/2}{v}$$

and

$$d_2(f,k,v) = d_1(f,k,v) - v.$$

7.65 BlackScholesLattice Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/bsmlattice.hpp>`

Inheritance diagram for BlackScholesLattice:

7.65.1 Detailed Description

template<class T> class QuantLib::BlackScholesLattice< T >

Simple binomial lattice approximating the Black-Scholes model.

Public Member Functions

- **BlackScholesLattice** (const boost::shared_ptr< T > &tree, Rate riskFreeRate, Time end, Size steps)
- Size **size** (Size i) const
- DiscountFactor **discount** (Size, Size) const
- void **stepback** (Size i, const Array &values, Array &newValues) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

7.66 BlackScholesProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/blackscholesprocess.hpp>`

Inheritance diagram for BlackScholesProcess:

7.66.1 Detailed Description

Black-Scholes stochastic process.

This class describes the stochastic process governed by

$$dS(t,S) = (r(t) - q(t) - \frac{\sigma(t,S)^2}{2})dt + \sigma dW_t.$$

Public Member Functions

- **BlackScholesProcess** (const Handle< Quote > &x0, const Handle< YieldTermStructure > &dividendTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< BlackVolTermStructure > &blackVolTS, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new EulerDiscretization))
- Time time (const Date &) const

StochasticProcess1D interface

- Real x0 () const
  returns the initial value of the state variable
- Real drift (Time t, Real x) const
- Real diffusion (Time t, Real x) const
- Real apply (Real x0, Real dx) const

Observer interface

- void update ()

Inspectors

- const boost::shared_ptr< Quote > & **stateVariable** () const
- const boost::shared_ptr< YieldTermStructure > & **dividendYield** () const
- const boost::shared_ptr< YieldTermStructure > & **riskFreeRate** () const
- const boost::shared_ptr< BlackVolTermStructure > & **blackVolatility** () const
- const boost::shared_ptr< LocalVolTermStructure > & **localVolatility** () const

7.66.2 Member Function Documentation

7.66.2.1 Real drift (Time *t*, Real *x*) const [virtual]

Todo
revise extrapolation

Implements StochasticProcess1D.

7.66.2.2 Real diffusion (Time *t*, Real *x*) const [virtual]

Todo
revise extrapolation

Implements StochasticProcess1D.

7.66.2.3 Real apply (Real *x0*, Real *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from StochasticProcess1D.

7.66.2.4 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from StochasticProcess.

7.66.2.5 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from StochasticProcess.

7.67 BlackSwaptionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/blackswaptionengine.hpp>`

Inheritance diagram for BlackSwaptionEngine:

7.67.1 Detailed Description

Black-formula swaption engine.

Public Member Functions

- **BlackSwaptionEngine** (const boost::shared_ptr< BlackModel > &model)
- void **calculate** () const

7.68 BlackVarianceCurve Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackvariancecurve.hpp>`

Inheritance diagram for BlackVarianceCurve:

7.68.1 Detailed Description

Black volatility curve modelled as variance curve.

This class calculates time-dependent Black volatilities using as input a vector of (ATM) Black volatilities observed in the market.

The calculation is performed interpolating on the variance curve. Linear interpolation is used as default; this can be changed by the setInterpolation() method.

For strike dependence, see BlackVarianceSurface.

Todo

check time extrapolation

Public Member Functions

- **BlackVarianceCurve** (const Date &referenceDate, const std::vector< Date > &dates, const std::vector< Volatility > &blackVolCurve, const DayCounter &dayCounter)

BlackVolTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Modifiers

- template<class Interpolator> void **setInterpolation** (const Interpolator &i=Interpolator())

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- virtual Real blackVarianceImpl (Time t, Real) const
  Black variance calculation.

7.69 BlackVarianceSurface Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackvariancesurface.hpp>`

Inheritance diagram for BlackVarianceSurface:

7.69.1 Detailed Description

Black volatility surface modelled as variance surface.

This class calculates time/strike dependent Black volatilities using as input a matrix of Black volatilities observed in the market.

The calculation is performed interpolating on the variance surface. Bilinear interpolation is used as default; this can be changed by the setInterpolation() method.

Todo
check time extrapolation

Public Types

- enum **Extrapolation** { **ConstantExtrapolation**, **InterpolatorDefaultExtrapolation** }

Public Member Functions

- **BlackVarianceSurface** (const Date &referenceDate, const std::vector< Date > &dates, const std::vector< Real > &strikes, const Matrix &blackVolMatrix, const DayCounter &dayCounter, Extrapolation lowerExtrapolation=InterpolatorDefaultExtrapolation, Extrapolation upperExtrapolation=InterpolatorDefaultExtrapolation)

BlackVolTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Modifiers

- template<class Interpolator> void **setInterpolation** (const Interpolator &i=Interpolator())

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- virtual Real blackVarianceImpl (Time t, Real strike) const

  Black variance calculation.

7.70 BlackVarianceTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/voltermstructure.hpp>`

Inheritance diagram for BlackVarianceTermStructure:

7.70.1 Detailed Description

Black variance term structure.

This abstract class acts as an adapter to VolTermStructure allowing the programmer to implement only the `blackVarianceImpl(Time, Real, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- BlackVarianceTermStructure ()
  default constructor
- BlackVarianceTermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- BlackVarianceTermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- Volatility blackVolImpl (Time maturity, Real strike) const

7.70.2 Constructor & Destructor Documentation

7.70.2.1 BlackVarianceTermStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.70.3 Member Function Documentation

7.70.3.1 Volatility blackVolImpl (Time *maturity*, Real *strike*) const [protected, virtual]

Returns the volatility for the given strike and date calculating it from the variance.

Implements BlackVolTermStructure.

7.71 BlackVolatilityTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/voltermstructure.hpp>`

Inheritance diagram for BlackVolatilityTermStructure:

7.71.1 Detailed Description

Black-volatility term structure.

This abstract class acts as an adapter to BlackVolTermStructure allowing the programmer to implement only the `blackVolImpl(Time, Real, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- BlackVolatilityTermStructure ()
  default constructor
- BlackVolatilityTermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- BlackVolatilityTermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- Real blackVarianceImpl (Time maturity, Real strike) const

7.71.2 Constructor & Destructor Documentation

7.71.2.1 BlackVolatilityTermStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.71.3 Member Function Documentation

7.71.3.1 Real blackVarianceImpl (Time *maturity*, Real *strike*) const [protected, virtual]

Returns the variance for the given strike and date calculating it from the volatility.

Implements BlackVolTermStructure.

7.72 BlackVolTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/voltermstructure.hpp>`

Inheritance diagram for BlackVolTermStructure:

7.72.1 Detailed Description

Black-volatility term structure.

This abstract class defines the interface of concrete Black-volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- BlackVolTermStructure ()
  default constructor
- BlackVolTermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- BlackVolTermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Black Volatility

- Volatility blackVol (const Date &maturity, Real strike, bool extrapolate=false) const
  present (a.k.a spot) volatility
- Volatility blackVol (Time maturity, Real strike, bool extrapolate=false) const
  present (a.k.a spot) volatility
- Real blackVariance (const Date &maturity, Real strike, bool extrapolate=false) const
  present (a.k.a spot) variance
- Real blackVariance (Time maturity, Real strike, bool extrapolate=false) const
  present (a.k.a spot) variance
- Volatility blackForwardVol (const Date &date1, const Date &date2, Real strike, bool extrapolate=false) const
  future (a.k.a. forward) volatility
- Volatility blackForwardVol (Time time1, Time time2, Real strike, bool extrapolate=false) const
  future (a.k.a. forward) volatility

- Real blackForwardVariance (const Date &date1, const Date &date2, Real strike, bool extrapolate=false) const
  future (a.k.a. forward) variance
- Real blackForwardVariance (Time time1, Time time2, Real strike, bool extrapolate=false) const
  future (a.k.a. forward) variance

Limits

- virtual Date maxDate () const =0
  the latest date for which the term structure can return vols
- Time maxTime () const
  the latest time for which the term structure can return vols
- virtual Real minStrike () const =0
  the minimum strike for which the term structure can return vols
- virtual Real maxStrike () const =0
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual volatility calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual Real blackVarianceImpl (Time t, Real strike) const =0
  Black variance calculation.
- virtual Volatility blackVolImpl (Time t, Real strike) const =0
  Black volatility calculation.

7.72.2 Constructor & Destructor Documentation

7.72.2.1 BlackVolTermStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.73 Bombay Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/bombay.hpp>`

Inheritance diagram for Bombay:

7.73.1 Detailed Description

Bombay calendar

Holidays (data from <http://www.nse-india.com/>):

- Saturdays
- Sundays
- Republic Day, January 26th
- Good Friday
- Ambedkar Jayanti, April 14th
- Independence Day, August 15th
- Gandhi Jayanti, October 2nd
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2005 only:)

- Bakri Id
- Moharram
- Holi
- Maharashtra Day
- Ganesh Chaturthi
- Dasara
- Laxmi Puja
- Bhaubeej
- Ramzan Id
- Guru Nanak Jayanti

7.74 Bond Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/bond.hpp>`

Inheritance diagram for Bond:

7.74.1 Detailed Description

Base bond class.

Derived classes must fill the unitialized data members.

Warning:
Most methods assume that the cashflows are stored sorted by date

Test

- price/yield calculations are cross-checked for consistency.
- price/yield calculations are checked against known good values.

Public Member Functions

Inspectors

- Date **settlementDate** () const
- const std::vector< boost::shared_ptr< CashFlow > > & **cashflows** () const
- const boost::shared_ptr< CashFlow > & **redemption** () const
- const Calendar & **calendar** () const
- BusinessDayConvention **businessDayConvention** () const
- const DayCounter & **dayCounter** () const
- Frequency **frequency** () const
- boost::shared_ptr< YieldTermStructure > **discountCurve** () const

Calculations

- Real cleanPrice () const
  theoretical clean price
- Real dirtyPrice () const
  theoretical dirty price
- Rate yield (Compounding compounding, Real accuracy=1.0e-8, Size maxEvaluations=100) const
  theoretical bond yield
- Real cleanPrice (Rate yield, Compounding compounding, Date settlementDate=Date()) const
  clean price given a yield and settlement date
- Real dirtyPrice (Rate yield, Compounding compounding, Date settlementDate=Date()) const
  dirty price given a yield and settlement date
- Rate yield (Real cleanPrice, Compounding compounding, Date settlementDate=Date(), Real accuracy=1.0e-8, Size maxEvaluations=100) const

yield given a (clean) price and settlement date

- Real accruedAmount (Date d=Date()) const
  accrued amount at a given date
- bool isExpired () const
  returns whether the instrument is still tradable.

Protected Member Functions

- **Bond** (const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention businessDayConvention, Integer settlementDays, const Handle< YieldTermStructure > &discountCurve=Handle< YieldTermStructure >())
- void performCalculations () const

Protected Attributes

- Integer **settlementDays_**
- Calendar **calendar_**
- BusinessDayConvention **businessDayConvention_**
- DayCounter **dayCount_**
- Date **issueDate_**
- Date **datedDate_**
- Date **maturityDate_**
- Frequency **frequency_**
- std::vector< boost::shared_ptr< CashFlow > > **cashFlows_**
- boost::shared_ptr< CashFlow > **redemption_**
- Handle< YieldTermStructure > **discountCurve_**

7.74.2 Member Function Documentation

7.74.2.1 Real cleanPrice () const

theoretical clean price

The default bond settlement is used for calculation.

7.74.2.2 Real dirtyPrice () const

theoretical dirty price

The default bond settlement is used for calculation.

7.74.2.3 Rate yield (Compounding *compounding*, Real *accuracy* = `1.0e-8`, Size *maxEvaluations* = `100`) const

theoretical bond yield

The default bond settlement and theoretical price are used for calculation.

7.74.2.4 Real cleanPrice (Rate *yield*, Compounding *compounding*, Date *settlementDate* = Date()) const

clean price given a yield and settlement date

The default bond settlement is used if no date is given.

7.74.2.5 Real dirtyPrice (Rate *yield*, Compounding *compounding*, Date *settlementDate* = Date()) const

dirty price given a yield and settlement date

The default bond settlement is used if no date is given.

7.74.2.6 Rate yield (Real *cleanPrice*, Compounding *compounding*, Date *settlementDate* = Date(), Real *accuracy* = 1.0e-8, Size *maxEvaluations* = 100) const

yield given a (clean) price and settlement date

The default bond settlement is used if no date is given.

7.74.2.7 Real accruedAmount (Date *d* = Date()) const

accrued amount at a given date

The default bond settlement is used if no date is given.

7.74.2.8 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from Instrument.

7.75 BoundaryCondition Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/boundarycondition.hpp>`

7.75.1 Detailed Description

template<class Operator> class QuantLib::BoundaryCondition< Operator >

Abstract boundary condition class for finite difference problems.

Public Types

- typedef Operator **operator_type**
- typedef Operator::array_type **array_type**
- enum Side { **None**, **Upper**, **Lower** }

Public Member Functions

- virtual void applyBeforeApplying (operator_type &) const =0
- virtual void applyAfterApplying (array_type &) const =0
- virtual void applyBeforeSolving (operator_type &, array_type &rhs) const =0
- virtual void applyAfterSolving (array_type &) const =0
- virtual void setTime (Time t)=0

7.75.2 Member Enumeration Documentation

7.75.2.1 enum Side

Todo

Generalize for n-dimensional conditions

7.75.3 Member Function Documentation

7.75.3.1 virtual void applyBeforeApplying (operator_type &) const [pure virtual]

This method modifies an operator L before it is applied to an array u so that $v = Lu$ will satisfy the given condition.

Implemented in NeumannBC, and DirichletBC.

7.75.3.2 virtual void applyAfterApplying (array_type &) const [pure virtual]

This method modifies an array u so that it satisfies the given condition.

Implemented in NeumannBC, and DirichletBC.

7.75.3.3 virtual void applyBeforeSolving (operator_type &, array_type & *rhs*) const [pure virtual]

This method modifies an operator L before the linear system $Lu' = u$ is solved so that u' will satisfy the given condition.

Implemented in NeumannBC, and DirichletBC.

7.75.3.4 virtual void applyAfterSolving (array_type &) const [pure virtual]

This method modifies an array u so that it satisfies the given condition.

Implemented in NeumannBC, and DirichletBC.

7.75.3.5 virtual void setTime (Time *t*) [pure virtual]

This method sets the current time for time-dependent boundary conditions.

Implemented in NeumannBC, and DirichletBC.

7.76 BoundaryConstraint Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

Inheritance diagram for BoundaryConstraint:

7.76.1 Detailed Description

Constraint imposing all arguments to be in [low,high]

Public Member Functions

- **BoundaryConstraint** (Real low, Real high)

7.77 BoxMullerGaussianRng Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/boxmullergaussianrng.hpp>`

7.77.1 Detailed Description

template<class RNG> class QuantLib::BoxMullerGaussianRng< RNG >

Gaussian random number generator.

It uses the well-known Box-Muller transformation to return a normal distributed Gaussian deviate with average 0.0 and standard deviation of 1.0, from a uniform deviate in (0,1) supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef Sample< Real > **sample_type**
- typedef RNG **urng_type**

Public Member Functions

- **BoxMullerGaussianRng** (const RNG &uniformGenerator)
- sample_type next () const

  returns a sample from a Gaussian distribution

7.78 BPSBasketCalculator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/basispointsensitivity.hpp>`

Inheritance diagram for BPSBasketCalculator:

7.78.1 Detailed Description

Bug

this class must still be checked. It is not guaranteed to yield the right results.

Public Member Functions

- **BPSBasketCalculator** (const Handle< YieldTermStructure > &ts, Integer basis)
- const TimeBasket & **result** () const

Visitor interface

- virtual void **visit** (Coupon &)
- virtual void **visit** (FixedRateCoupon &)
- virtual void **visit** (CashFlow &)

7.79 BPSCalculator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/basispointsensitivity.hpp>`

Inheritance diagram for BPSCalculator:

7.79.1 Detailed Description

basis point sensitivity (BPS) calculator

Instances of this class accumulate the BPS of each cash flow they visit, returning the sum through their result() method.

Public Member Functions

- **BPSCalculator** (const Handle< YieldTermStructure > &ts)
- Real **result** () const

Visitor interface

- virtual void **visit** (Coupon &)
- virtual void **visit** (CashFlow &)

7.80 Bratislava Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/bratislava.hpp>`

Inheritance diagram for Bratislava:

7.80.1 Detailed Description

Bratislava calendar

Holidays (see http://www.bsse.sk/):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- May Day, May 1st
- Liberation of the Republic, May 8th
- SS. Cyril and Methodius, July 5th
- Slovak National Uprising, August 29th
- Constitution of the Slovak Republic, September 1st
- Our Lady of the Seven Sorrows, September 15th
- All Saints Day, November 1st
- Freedom and Democracy of the Slovak Republic, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

7.81 Brent Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/brent.hpp>`

Inheritance diagram for Brent:

7.81.1 Detailed Description

Brent 1-D solver

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.82 Bridge Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/bridge.hpp>`

7.82.1 Detailed Description

template<class T, class T_impl> class QuantLib::Bridge< T, T_impl >

The Bridge pattern made explicit.

The typical use of this class is:

```
class FooImpl;
class Foo : public Bridge<Foo,FooImpl> {
    ...
};
```

which makes it possible to pass instances of class Foo by value while retaining polymorphic behavior.

Public Types

- typedef T_impl **Impl**

Public Member Functions

- bool **isNull** () const

Protected Member Functions

- **Bridge** (const boost::shared_ptr< Impl > &impl=boost::shared_ptr< Impl >())

Protected Attributes

- boost::shared_ptr< Impl > **impl_**

7.83 BRLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for BRLCurrency:

7.83.1 Detailed Description

Brazilian real.

The ISO three-letter code is BRL; the numeric code is 986. It is divided in 100 centavos.

7.84 BrownianBridge Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/brownianbridge.hpp>`

7.84.1 Detailed Description

template<class GSG> class QuantLib::BrownianBridge< GSG >

Builds Wiener process paths using Gaussian variates.

For more details: "Monte Carlo Methods in Finance" by P. Jäckel, section 10.8.3

Note:
this class does not work if the diffusion term of the underlying stochastic process is asset-dependent.

Public Types

- typedef Sample< std::vector< Real > > **sample_type**

Public Member Functions

- BrownianBridge (const GSG &generator)
  normalised (unit time, unit variance) Wiener process paths
- BrownianBridge (Time length, Size timeSteps, const GSG &generator)
  unit variance Wiener process paths
- BrownianBridge (const TimeGrid &timeGrid, const GSG &generator)
  unit variance Wiener process paths
- BrownianBridge (const std::vector< Real > &sigma, const TimeGrid &timeGrid, const GSG &generator)
  general Wiener process paths
- **BrownianBridge** (const boost::shared_ptr< BlackVolTermStructure > &, const TimeGrid &timeGrid, const GSG &generator)
- **BrownianBridge** (const boost::shared_ptr< StochasticProcess1D > &, const TimeGrid &timeGrid, const GSG &generator)

inspectors

- const sample_type & **next** () const
- const sample_type & **last** () const
- Size **size** () const
- const TimeGrid & **timeGrid** () const

7.85 BSMOperator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/bsmoperator.hpp>`

Inheritance diagram for BSMOperator:

7.85.1 Detailed Description

Black-Scholes-Merton differential operator.

Public Member Functions

- **BSMOperator** (Size size, Real dx, Rate r, Rate q, Volatility sigma)
- **BSMOperator** (const Array &grid, const boost::shared_ptr< BlackScholesProcess > &, Time residualTime)

7.86 BSMTermOperator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/bsmtermoperator.hpp>`

Inheritance diagram for BSMTermOperator:

7.86.1 Detailed Description

Black-Scholes-Merton differential operator.

Test
coefficients are tested against constant BSM operator

Public Member Functions

- **BSMTermOperator** (const Array &grid, const boost::shared_ptr< BlackScholesProcess > &, Time residualTime=0.0)

7.87 Budapest Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/budapest.hpp>`

Inheritance diagram for Budapest:

7.87.1 Detailed Description

Budapest calendar

Holidays:

- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

7.88 BYRCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for BYRCurrency:

7.88.1 Detailed Description

Belarussian ruble.

The ISO three-letter code is BYR; the numeric code is 974. It has no subdivisions.

7.89 CADCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for CADCurrency:

7.89.1 Detailed Description

Canadian dollar.

The ISO three-letter code is CAD; the numeric code is 124. It is divided into 100 cents.

7.90 CADLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/cadlibor.hpp>`

Inheritance diagram for CADLibor:

7.90.1 Detailed Description

CAD LIBOR rate

Canadian Dollar LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Warning:
This is the rate fixed in London by BBA. Use CDOR if you're interested in the Canadian fixing by IDA.

Public Member Functions

- **CADLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.91 Calendar Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/calendar.hpp>`

Inheritance diagram for Calendar:

7.91.1 Detailed Description

calendar class

This class provides methods for determining whether a date is a business day or a holiday for a given market, and for incrementing/decrementing a date of a given number of business days.

The Bridge pattern is used to provide the base behavior of the calendar, namely, to determine whether a date is a business day.

A calendar should be defined for specific exchange holiday schedule or for general country holiday schedule. Legacy city holiday schedule calendars will be moved to the exchange/country convention.

Test

the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Public Member Functions

- Calendar ()

Calendar interface

- std::string name () const
  Returns the name of the calendar.
- bool isBusinessDay (const Date &d) const
- bool isHoliday (const Date &d) const
- bool isEndOfMonth (const Date &d) const
- void addHoliday (const Date &)
- void removeHoliday (const Date &)
- Date adjust (const Date &, BusinessDayConvention convention=Following, const Date &origin=Date()) const
- Date advance (const Date &, Integer n, TimeUnit unit, BusinessDayConvention convention=Following) const
- Date advance (const Date &date, const Period &period, BusinessDayConvention convention=Following) const

Protected Member Functions

- Calendar (const boost::shared_ptr< CalendarImpl > &impl)

Related Functions

(Note that these are not member functions.)

- bool operator== (const Calendar &, const Calendar &)
- bool **operator!=** (const Calendar &, const Calendar &)

Classes

- class WesternImpl

  partial calendar implementation

7.91.2 Constructor & Destructor Documentation

7.91.2.1 Calendar ()

This default constructor returns a calendar with a null implementation, which is therefore unusable except as a placeholder.

7.91.2.2 Calendar (const boost::shared_ptr< CalendarImpl > & *impl*) [protected]

This protected constructor will only be invoked by derived classes which define a given Calendar implementation

7.91.3 Member Function Documentation

7.91.3.1 std::string name () const

Returns the name of the calendar.

Warning:
This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

7.91.3.2 bool isBusinessDay (const Date & *d*) const

Returns `true` iff the date is a business day for the given market.

7.91.3.3 bool isHoliday (const Date & *d*) const

Returns `true` iff the date is a holiday for the given market.

7.91.3.4 bool isEndOfMonth (const Date & *d*) const

Returns `true` iff the date is last business day for the month in given market.

7.91.3.5 void addHoliday (const Date &)

Adds a date to the set of holidays for the given calendar.

7.91.3.6 void removeHoliday (const Date &)

Removes a date from the set of holidays for the given calendar.

7.91.3.7 Date adjust (const Date &, BusinessDayConvention *convention* = `Following`, const Date & *origin* = `Date()`) const

Adjusts a non-business day to the appropriate near business day with respect to the given convention.

7.91.3.8 Date advance (const Date &, Integer *n*, TimeUnit *unit*, BusinessDayConvention *convention* = `Following`) const

Advances the given date of the given number of business days and returns the result.

Note:
The input date is not modified.

7.91.3.9 Date advance (const Date & *date*, const Period & *period*, BusinessDayConvention *convention* = `Following`) const

Advances the given date as specified by the given period and returns the result.

Note:
The input date is not modified.

7.91.4 Friends And Related Function Documentation

7.91.4.1 bool operator== (const Calendar &, const Calendar &) `[related]`

Returns `true` iff the two calendars belong to the same derived class.

7.92 Calendar::WesternImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/calendar.hpp>`

Inheritance diagram for Calendar::WesternImpl:

7.92.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Easter Monday for a given year.

Static Protected Member Functions

- static Day easterMonday (Year y)
  expressed relative to first day of year

7.93 CalendarImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/calendar.hpp>`

Inheritance diagram for CalendarImpl:

7.93.1 Detailed Description

abstract base class for calendar implementations

Public Member Functions

- virtual std::string **name** () const =0
- virtual bool **isBusinessDay** (const Date &) const =0

Public Attributes

- std::set< Date > **addedHolidays**
- std::set< Date > **removedHolidays**

7.94 CalibrationHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/calibrationhelper.hpp>`

Inheritance diagram for CalibrationHelper:

7.94.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

- **CalibrationHelper** (const Handle< Quote > &volatility, const Handle< YieldTermStructure > &termStructure)
- void update ()
- Real marketValue () const

  returns the actual price of the instrument (from volatility)

- virtual Real modelValue () const =0

  returns the price of the instrument according to the model

- virtual Real calibrationError ()

  returns the error resulting from the model valuation

- virtual void **addTimesTo** (std::list< Time > &times) const =0
- Volatility impliedVolatility (Real targetValue, Real accuracy, Size maxEvaluations, Volatility minVol, Volatility maxVol) const

  Black volatility implied by the model.

- virtual Real blackPrice (Volatility volatility) const =0

  Black price given a volatility.

- void **setPricingEngine** (const boost::shared_ptr< PricingEngine > &engine)

Protected Attributes

- Real **marketValue_**
- Handle< Quote > **volatility_**
- Handle< YieldTermStructure > **termStructure_**
- boost::shared_ptr< BlackModel > **blackModel_**
- boost::shared_ptr< PricingEngine > **engine_**

7.94.2 Member Function Documentation

7.94.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.95 Cap Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for Cap:

7.95.1 Detailed Description

Concrete cap class.

Public Member Functions

- **Cap** (const std::vector< boost::shared_ptr< CashFlow > > &floatingLeg, const std::vector< Rate > &exerciseRates, const Handle< YieldTermStructure > &termStructure, const boost::shared_ptr< PricingEngine > &engine)

7.96 CapFloor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for CapFloor:

7.96.1 Detailed Description

Base class for cap-like instruments.

Test

- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

Public Types

- enum **Type** { **Cap**, **Floor**, **Collar** }

Public Member Functions

- **CapFloor** (Type type, const std::vector< boost::shared_ptr< CashFlow > > &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const Handle< YieldTermStructure > &termStructure, const boost::shared_ptr< PricingEngine > &engine)
- void setupArguments (Arguments *) const
- Volatility impliedVolatility (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=QL_MIN_VOLATILITY, Volatility maxVol=QL_MAX_VOLATILITY) const

  implied term volatility

Instrument interface

- bool isExpired () const

  returns whether the instrument is still tradable.

Inspectors

- Type **type** () const
- const std::vector< boost::shared_ptr< CashFlow > > & **leg** () const
- const std::vector< Rate > & **capRates** () const
- const std::vector< Rate > & **floorRates** () const

Classes

- class arguments

  Arguments for cap/floor calculation

- class results

  Results from cap/floor calculation

7.96.2 Member Function Documentation

7.96.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from Instrument.

7.97 CapFloor::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for CapFloor::arguments:

7.97.1 Detailed Description

Arguments for cap/floor calculation

Public Member Functions

- void **validate** () const

Public Attributes

- CapFloor::Type **type**
- std::vector< Time > **startTimes**
- std::vector< Time > **fixingTimes**
- std::vector< Time > **endTimes**
- std::vector< Time > **accrualTimes**
- std::vector< Rate > **capRates**
- std::vector< Rate > **floorRates**
- std::vector< Rate > **forwards**
- std::vector< Real > **nominals**

7.98 CapFloor::results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for CapFloor::results:

7.98.1 Detailed Description

Results from cap/floor calculation

7.99 CapletConstantVolatility Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/capletconstantvol.hpp>`

Inheritance diagram for CapletConstantVolatility:

7.99.1 Detailed Description

Constant caplet volatility, no time-strike dependence.

Public Member Functions

- **CapletConstantVolatility** (const Date &referenceDate, Volatility volatility, const DayCounter &dayCounter)
- **CapletConstantVolatility** (const Date &referenceDate, const Handle< Quote > &volatility, const DayCounter &dayCounter)
- **CapletConstantVolatility** (Integer settlementDays, const Calendar &, Volatility volatility, const DayCounter &dayCounter)
- **CapletConstantVolatility** (Integer settlementDays, const Calendar &, const Handle< Quote > &volatility, const DayCounter &dayCounter)
- Date maxDate () const

  the latest date for which the term structure can return vols

- Time maxTime () const

  the latest time for which the term structure can return vols

- Real minStrike () const

  the minimum strike for which the term structure can return vols

- Real maxStrike () const

  the maximum strike for which the term structure can return vols

TermStructure interface

- DayCounter dayCounter () const

  the day counter used for date/time conversion

Protected Member Functions

CapletVolatilityStructure interface

- Volatility volatilityImpl (Time t, Rate) const

  implements the actual volatility calculation in derived classes

7.100 CapletLiborMarketModelProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/capletlmmprocess.hpp>`

Inheritance diagram for CapletLiborMarketModelProcess:

7.100.1 Detailed Description

caplet libor-market-model process

stochastic process of a (cap) libor market model using the rolling forward measure incl. predictor-corrector step

References: Glasserman, Paul, 2004, Monte Carlo Methods in Financial Engineering, Springer, Section 3.7

Antoon Pelsser, 2000, Efficient Methods for Valuing Interest Rate Derivatives, Springer, 8

Hull, John, White, Alan, 1999, Forward Rate Volatilities, Swap Rate Volatilities and the Implementation of the Libor Market Model (<http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf>)

Test

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

Public Member Functions

- CapletLiborMarketModelProcess (Size fixings, const boost::shared_ptr< Xibor > &underlyingIndex, const boost::shared_ptr< CapletVolatilityStructure > &capletVol, const Matrix &volatilityComponents=Matrix())
- Size size () const

  returns the number of dimensions of the stochastic process

- Size factors () const

  returns the number of independent factors of the process

- Disposable< Array > initialValues () const

  returns the initial values of the state variables

- Disposable< Array > drift (Time t, const Array &x) const

  returns the drift part of the equation, i.e., $\mu(t, x_t)$

- Disposable< Matrix > diffusion (Time t, const Array &x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

- Disposable< Array > apply (const Array &x0, const Array &dx) const
- Disposable< Array > evolve (Time t0, const Array &x0, Time dt, const Array &dw) const
- std::vector< Time > **fixingTimes** () const
- Time **accrualPeriod** (Size i) const
- Volatility lambda (Size i, Size j=0) const
- DiscountFactor discountBond (const std::vector< Rate > &rates, Size j) const

  discount factor until the j-th fixing period

Protected Member Functions

- Size **nextResetDate** (Time t) const

7.100.2 Constructor & Destructor Documentation

7.100.2.1 CapletLiborMarketModelProcess (Size *fixings*, const boost::shared_ptr< Xibor > & *underlyingIndex*, const boost::shared_ptr< CapletVolatilityStructure > & *capletVol*, const Matrix & *volatilityComponents* = `Matrix()`)

Parameters:

fixings number of rate fixing

underlyingIndex underlying Libor index

capletVol cap volatility term structure. Used to bootstrap volatilities Λ_i of F_i .

volatilityComponents $\lambda_{i,q}/\Lambda_i$, the ratio of the q -th component of the volatility of the forward rate to the total volatility of the forward rate. The number of columns of this matrix defines the number of factors of the model.

7.100.3 Member Function Documentation

7.100.3.1 Disposable<Array> apply (const Array & *x0*, const Array & *dx*) const `[virtual]`

applies a change to the asset value. By default, it returns x + Δx .

Reimplemented from StochasticProcess.

7.100.3.2 Disposable<Array> evolve (Time *t0*, const Array & *x0*, Time *dt*, const Array & *dw*) const `[virtual]`

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented from StochasticProcess.

7.100.3.3 Volatility lambda (Size *i*, Size *j* = `0`) const

volatility matrix $\lambda_{i,j}$, i-th fixing, j-th volatility factor, see equation 20 in Hull White paper

7.101 CapletVolatilityStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/capvolstructures.hpp>`

Inheritance diagram for CapletVolatilityStructure:

7.101.1 Detailed Description

Caplet/floorlet forward-volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- CapletVolatilityStructure ()
  default constructor
- CapletVolatilityStructure (const Date &referenceDate)
  initialize with a fixed reference date
- CapletVolatilityStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Volatility

- Volatility volatility (const Date &start, Rate strike, bool extrapolate=false) const
  returns the volatility for a given start date and strike rate
- Volatility volatility (Time t, Rate strike, bool extrapolate=false) const
  returns the volatility for a given start time and strike rate

Limits

- virtual Date maxDate () const =0
  the latest date for which the term structure can return vols
- virtual Time maxTime () const
  the latest time for which the term structure can return vols
- virtual Real minStrike () const =0
  the minimum strike for which the term structure can return vols
- virtual Real maxStrike () const =0
  the maximum strike for which the term structure can return vols

Protected Member Functions

- virtual Volatility volatilityImpl (Time length, Rate strike) const =0

  implements the actual volatility calculation in derived classes

7.101.2 Constructor & Destructor Documentation

7.101.2.1 CapletVolatilityStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.102 CapVolatilityStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/capvolstructures.hpp>`

Inheritance diagram for CapVolatilityStructure:

7.102.1 Detailed Description

Cap/floor term-volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- CapVolatilityStructure ()
  default constructor
- CapVolatilityStructure (const Date &referenceDate)
  initialize with a fixed reference date
- CapVolatilityStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Volatility

- Volatility **volatility** (const Date &end, Rate strike, bool extrapolate=false) const
- Volatility volatility (const Period &length, Rate strike, bool extrapolate=false) const
  returns the volatility for a given cap/floor length and strike rate
- Volatility volatility (Time t, Rate strike, bool extrapolate=false) const
  returns the volatility for a given end time and strike rate

Limits

- virtual Date maxDate () const =0
  the latest date for which the term structure can return vols
- virtual Time maxTime () const
  the latest time for which the term structure can return vols
- virtual Real minStrike () const =0
  the minimum strike for which the term structure can return vols
- virtual Real maxStrike () const =0
  the maximum strike for which the term structure can return vols

Protected Member Functions

- virtual Volatility volatilityImpl (Time length, Rate strike) const =0

  implements the actual volatility calculation in derived classes

7.102.2 Constructor & Destructor Documentation

7.102.2.1 CapVolatilityStructure ()

default constructor

Warning:

term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.103 CapVolatilityVector Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/capflatvolvector.hpp>`

Inheritance diagram for CapVolatilityVector:

7.103.1 Detailed Description

Cap/floor at-the-money term-volatility vector.

This class provides the at-the-money volatility for a given cap by interpolating a volatility vector whose elements are the market volatilities of a set of caps/floors with given length.

Todo

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Public Member Functions

- **CapVolatilityVector** (const Date &settlementDate, const std::vector< Period > &lengths, const std::vector< Volatility > &volatilities, const DayCounter &dayCounter)
- **CapVolatilityVector** (Integer settlementDays, const Calendar &calendar, const std::vector< Period > &lengths, const std::vector< Volatility > &volatilities, const DayCounter &dayCounter)
- DayCounter dayCounter () const

  the day counter used for date/time conversion

- Date maxDate () const

  the latest date for which the term structure can return vols

- Time maxTime () const

  the latest time for which the term structure can return vols

- Real minStrike () const

  the minimum strike for which the term structure can return vols

- Real maxStrike () const

  the maximum strike for which the term structure can return vols

- void update ()

7.103.2 Member Function Documentation

7.103.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from TermStructure.

7.104 CashFlow Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/cashflow.hpp>`

Inheritance diagram for CashFlow:

7.104.1 Detailed Description

Base class for cash flows.

This class is purely virtual and acts as a base class for the actual cash flow implementations.

Public Member Functions

CashFlow interface

- virtual Real amount () const =0
  returns the amount of the cash flow
- virtual Date date () const =0
  returns the date at which the cash flow is settled

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.104.2 Member Function Documentation

7.104.2.1 virtual Real amount () const [pure virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implemented in FixedRateCoupon, IndexedCoupon, ParCoupon, Short< ParCoupon >, and SimpleCashFlow.

7.105 Cashflows Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/analysis.hpp>`

7.105.1 Detailed Description

cashflows analysis functions

Todo
add tests

Static Public Member Functions

- static Real npv (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &)

  NPV of the cash flows.

- static Real npv (const std::vector< boost::shared_ptr< CashFlow > > &, const InterestRate &, Date settlementDate=Date())

  NPV of the cash flows.

- static Rate irr (const std::vector< boost::shared_ptr< CashFlow > > &, Real marketPrice, const DayCounter &dayCounter, Compounding compounding, Frequency frequency=NoFrequency, Date settlementDate=Date(), Real tolerance=1.0e-10, Size maxIterations=10000, Rate guess=0.05)

  Internal rate of return.

- static Real convexity (const std::vector< boost::shared_ptr< CashFlow > > &, const InterestRate &, Date settlementDate=Date())

  Cash-flow convexity.

- static Time duration (const std::vector< boost::shared_ptr< CashFlow > > &, Real marketPrice, const InterestRate &, Duration::Type type=Duration::Simple, Date settlementDate=Date())

  Cash-flow duration.

7.105.2 Member Function Documentation

7.105.2.1 static Real npv (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &) [static]

NPV of the cash flows.

THe NPV is the sum of the cash flows, each discounted according to the given term structure.

7.105.2.2 static Real npv (const std::vector< boost::shared_ptr< CashFlow > > &, const InterestRate &, Date *settlementDate* = Date()) [static]

NPV of the cash flows.

The NPV is the sum of the cash flows, each discounted according to the given constant interest rate. The result is affected by the choice of the interest-rate compounding and the relative frequency and day counter.

7.105.2.3 static Rate irr (const std::vector< boost::shared_ptr< CashFlow > > &, Real *marketPrice*, const DayCounter & *dayCounter*, Compounding *compounding*, Frequency *frequency* = NoFrequency, Date *settlementDate* = Date(), Real *tolerance* = 1.0e-10, Size *maxIterations* = 10000, Rate *guess* = 0.05) [static]

Internal rate of return.

The IRR is the interest rate at which the NPV of the cash flows equals the given market price. The function verifies the theoretical existance of an IRR and numerically establishes the IRR to the desired precision.

7.105.2.4 static Real convexity (const std::vector< boost::shared_ptr< CashFlow > > &, const InterestRate &, Date *settlementDate* = Date()) [static]

Cash-flow convexity.

The convexity is defined as

$$C = \sum t^2 c_t P_t$$

where c_t is the amount of the cash flow and P_t is the discount at time t as implied by the given interest rate.

7.105.2.5 static Time duration (const std::vector< boost::shared_ptr< CashFlow > > &, Real *marketPrice*, const InterestRate &, Duration::Type *type* = Duration::Simple, Date *settlementDate* = Date()) [static]

Cash-flow duration.

The simple duration is defined as

$$D_{\text{simple}} = \frac{\sum t c_t P_t}{\sum c_t P_t}$$

where c_t is the amount of the cash flow and P_t is the discount at time t as implied by the given interest rate.

The modified duration is

$$D_{\text{modified}} = \frac{1}{y} D_{\text{simple}}$$

where y is the IRR.

Finally, the Macaulay duration is

$$D_{\text{Macaulay}} = \frac{\sum t c_t P'_t}{\sum c_t P'_t}$$

where $P'_t = e^{-yt}$ and y is the IRR.

7.106 CashOrNothingPayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for CashOrNothingPayoff:

7.106.1 Detailed Description

Binary cash-or-nothing payoff.

Public Member Functions

- **CashOrNothingPayoff** (Option::Type type, Real strike, Real cashPayoff)
- Real **operator()** (Real price) const
- Real **cashPayoff** () const

7.107 Cdor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/cdor.hpp>`

Inheritance diagram for Cdor:

7.107.1 Detailed Description

CDOR rate

Canadian Dollar Offered Rate fixed by IDA.

Warning:
This is the rate fixed in Canada by IDA. Use CADLibor if you're interested in the London fixing by BBA.

Todo
check settlement days and day-count convention.

Public Member Functions

- **Cdor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.108 CeilingTruncation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for CeilingTruncation:

7.108.1 Detailed Description

Ceiling truncation.

Public Member Functions

- **CeilingTruncation** (Integer precision, Integer digit=5)

7.109 CHFCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for CHFCurrency:

7.109.1 Detailed Description

Swiss franc.

The ISO three-letter code is CHF; the numeric code is 756. It is divided into 100 cents.

7.110 CHFLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/chflibor.hpp>`

Inheritance diagram for CHFLibor:

7.110.1 Detailed Description

CHF LIBOR rate

Swiss Franc LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Warning:
This is the rate fixed in London by BBA. Use ZIBOR if you're interested in the Zurich fixing.

Public Member Functions

- **CHFLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.111 CLGaussianRng Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/centrallimitgaussianrng.hpp>`

7.111.1 Detailed Description

template<class RNG> class QuantLib::CLGaussianRng< RNG >

Gaussian random number generator.

It uses the well-known fact that the sum of 12 uniform deviate in (-.5,.5) is approximately a Gaussian deviate with average 0 and standard deviation 1. The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef Sample< Real > **sample_type**
- typedef RNG **urng_type**

Public Member Functions

- **CLGaussianRng** (const RNG &uniformGenerator)
- sample_type next () const

  returns a sample from a Gaussian distribution

7.112 CliquetOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/cliquetoption.hpp>`

Inheritance diagram for CliquetOption:

7.112.1 Detailed Description

cliquet (Ratchet) option

A cliquet option, also known as Ratchet option, is a series of forward-starting (a.k.a. deferred strike) options where the strike for each forward start option is set equal to a fixed percentage of the spot price at the beginning of each period.

Todo

- add local/global caps/floors
- add accrued coupon and last fixing

Public Member Functions

- **CliquetOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< PercentageStrikePayoff > &, const boost::shared_ptr< EuropeanExercise > &maturity, const std::vector< Date > &resetDates, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Classes

- class arguments

  Arguments for cliquet option calculation

- class engine

  Cliquet engine base class.

7.112.2 Member Function Documentation

7.112.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.113 CliquetOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/cliquetoption.hpp>`

7.113.1 Detailed Description

Arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Real **accruedCoupon**
- Real **lastFixing**
- Real **localCap**
- Real **localFloor**
- Real **globalCap**
- Real **globalFloor**
- std::vector< Date > **resetDates**

7.114 CliquetOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/cliquetoption.hpp>`

Inheritance diagram for CliquetOption::engine:

7.114.1 Detailed Description

Cliquet engine base class.

7.115 ClosestRounding Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for ClosestRounding:

7.115.1 Detailed Description

Closest rounding.

Public Member Functions

- **ClosestRounding** (Integer precision, Integer digit=5)

7.116 CLPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for CLPCurrency:

7.116.1 Detailed Description

Chilean peso.

The ISO three-letter code is CLP; the numeric code is 152. It is divided in 100 centavos.

7.117 CNYCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for CNYCurrency:

7.117.1 Detailed Description

Chinese yuan.

The ISO three-letter code is CNY; the numeric code is 156. It is divided in 100 fen.

7.118 Collar Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for Collar:

7.118.1 Detailed Description

Concrete collar class.

Public Member Functions

- **Collar** (const std::vector< boost::shared_ptr< CashFlow > > &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const Handle< YieldTermStructure > &termStructure, const boost::shared_ptr< PricingEngine > &engine)

7.119 Composite Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/composite.hpp>`

7.119.1 Detailed Description

template<class T> class QuantLib::Composite< T >

Composite pattern.

The typical use of this class is:

```
class CompositeFoo : public Composite<Foo> {
    ...
};
```

which causes CompositeFoo to inherit from Foo and provides it with methods for adding components. Of course, any abstract Foo interface must still be implemented.

Protected Types

- typedef std::list< boost::shared_ptr< T > >::iterator **iterator**
- typedef std::list< boost::shared_ptr< T > >::const_iterator **const_iterator**

Protected Member Functions

- void **add** (const boost::shared_ptr< T > &c)

Protected Attributes

- std::list< boost::shared_ptr< T > > **components_**

7.120 CompositeConstraint Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

Inheritance diagram for CompositeConstraint:

7.120.1 Detailed Description

Constraint enforcing both given sub-constraints

Public Member Functions

- **CompositeConstraint** (const Constraint &c1, const Constraint &c2)

7.121 CompositeQuote Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/quote.hpp>`

Inheritance diagram for CompositeQuote:

7.121.1 Detailed Description

template<class BinaryFunction> class QuantLib::CompositeQuote< BinaryFunction >

market element whose value depends on two other market element

Test

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **CompositeQuote** (const Handle< Quote > &element1, const Handle< Quote > &element2, const BinaryFunction &f)

Quote interface

- Real value () const

  returns the current value

Observer interface

- void update ()

7.121.2 Member Function Documentation

7.121.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.122 CompoundForward Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/compoundforward.hpp>`

Inheritance diagram for CompoundForward:

7.122.1 Detailed Description

compound-forward structure

Test

- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Bug

swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

Public Member Functions

- **CompoundForward** (const Date &referenceDate, const std::vector< Date > &dates, const std::vector< Rate > &forwards, const Calendar &calendar, const BusinessDayConvention conv, const Integer compounding, const DayCounter &dayCounter)
- Calendar calendar () const

  the calendar used for reference date calculation

- BusinessDayConvention **businessDayConvention** () const
- DayCounter dayCounter () const

  the day counter used for date/time conversion

- Integer **compounding** () const
- Date maxDate () const

  the latest date for which the curve can return rates

- Time maxTime () const

  the latest time for which the curve can return rates

- const std::vector< Time > & **times** () const
- const std::vector< Date > & **dates** () const
- const std::vector< Rate > & **forwards** () const
- boost::shared_ptr< ExtendedDiscountCurve > **discountCurve** () const
- Rate **compoundForward** (const Date &d1, Integer f, bool extrapolate=false) const
- Rate **compoundForward** (Time t1, Integer f, bool extrapolate=false) const

Protected Member Functions

- void **calibrateNodes** () const
- boost::shared_ptr< YieldTermStructure > **bootstrap** () const
- Rate zeroYieldImpl (Time) const
- DiscountFactor discountImpl (Time) const
- Size **referenceNode** (Time) const
- Rate forwardImpl (Time) const

  instantaneous forward-rate calculation

- Rate **compoundForwardImpl** (Time, Integer) const

7.122.2 Member Function Documentation

7.122.2.1 Rate zeroYieldImpl (Time) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented from ForwardRateStructure.

7.122.2.2 DiscountFactor discountImpl (Time) const [protected, virtual]

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Reimplemented from ForwardRateStructure.

7.123 ConjugateGradient Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/conjugategradient.hpp>`

Inheritance diagram for ConjugateGradient:

7.123.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

User has to provide line-search method and optimization end criteria

search direction $d_i = -f'(x_i) + c_i * d_{i-1}$ where $c_i = \|f'(x_i)\|^2 / \|f'(x_{i-1})\|^2$ and $d_1 = -f'(x_1)$

Public Member Functions

- ConjugateGradient ()
  default constructor
- **ConjugateGradient** (const boost::shared_ptr< LineSearch > &lineSearch)
- virtual ~ConjugateGradient ()
  destructor
- virtual void minimize (const Problem &P) const
  minimize the optimization problem P

7.124 ConstantParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

Inheritance diagram for ConstantParameter:

7.124.1 Detailed Description

Standard constant parameter $a(t) = a$.

Public Member Functions

- **ConstantParameter** (const Constraint &constraint)
- **ConstantParameter** (Real value, const Constraint &constraint)

7.125 Constraint Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

Inheritance diagram for Constraint:

7.125.1 Detailed Description

Base constraint class.

Public Member Functions

- bool **test** (const Array &p) const
- Real **update** (Array &p, const Array &direction, Real beta)
- **Constraint** (const boost::shared_ptr< ConstraintImpl > &impl=boost::shared_ptr< ConstraintImpl >())

7.126 ConstraintImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

7.126.1 Detailed Description

Base class for constraint implementations.

Public Member Functions

- virtual bool test (const Array &params) const =0

  Tests if params satisfy the constraint.

7.127 ContinuousAveragingAsianOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

Inheritance diagram for ContinuousAveragingAsianOption:

7.127.1 Detailed Description

Continuous-averaging Asian option.

Todo

add running average

Public Member Functions

- **ContinuousAveragingAsianOption** (Average::Type averageType, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Protected Attributes

- Average::Type **averageType_**

Classes

- class arguments

  Extra arguments for single-asset continuous-average Asian option.

- class engine

  Continuous-averaging Asian engine base class.

7.127.2 Member Function Documentation

7.127.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.128 ContinuousAveragingAsianOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

7.128.1 Detailed Description

Extra arguments for single-asset continuous-average Asian option.

Public Member Functions

- void **validate** () const

Public Attributes

- Average::Type **averageType**

7.129 ContinuousAveragingAsianOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

Inheritance diagram for ContinuousAveragingAsianOption::engine:

7.129.1 Detailed Description

Continuous-averaging Asian engine base class.

7.130 ConvergenceStatistics Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/convergencestatistics.hpp>`

7.130.1 Detailed Description

template<class T, class U = DoublingConvergenceSteps> class QuantLib::ConvergenceStatistics< T, U >

statistics class with convergence table

This class decorates another statistics class adding a convergence table calculation. The table tracks the convergence of the mean.

It is possible to specify the number of samples at which the mean should be stored by mean of the second template parameter; the default is to store 2^{n-1} samples at the n -th step. Any passed class must implement the following interface:

```
Size initialSamples() const;
Size nextSamples(Size currentSamples) const;
```

as well as a copy constructor.

Test

results are tested against known good values.

Public Member Functions

- **ConvergenceStatistics** (const U &rule=U())
- void **add** (Real value, Real weight=1.0)
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
- void **reset** ()
- const std::vector< std::pair< Size, Real > > & **convergenceTable** () const

7.131 COPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for COPCurrency:

7.131.1 Detailed Description

Colombian peso.

The ISO three-letter code is COP; the numeric code is 170. It is divided in 100 centavos.

7.132 Copenhagen Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/copenhagen.hpp>`

Inheritance diagram for Copenhagen:

7.132.1 Detailed Description

Copenhagen calendar

Holidays:

- Saturdays
- Sundays
- Maunday Thursday
- Good Friday
- Easter Monday
- General Prayer Day, 25 days after Easter Monday
- Ascension
- Whit (Pentecost) Monday
- New Year's Day, January 1st
- Constitution Day, June 5th
- Christmas, December 25th
- Boxing Day, December 26th

7.133 CostFunction Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/costfunction.hpp>`

Inheritance diagram for CostFunction:

7.133.1 Detailed Description

Cost function abstract class for optimization problem.

Public Member Functions

- virtual Real value (const Array &x) const =0
  method to overload to compute the cost functon value in x
- virtual void gradient (Array &grad, const Array &x) const
  method to overload to compute grad_f, the first derivative of
- virtual Real valueAndGradient (Array &grad, const Array &x) const
  method to overload to compute grad_f, the first derivative of
- virtual Real finiteDifferenceEpsilon () const
  Default epsilon for finite difference method :.

7.134 Coupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/coupon.hpp>`

Inheritance diagram for Coupon:

7.134.1 Detailed Description

coupon accruing over a fixed period

This class implements part of the CashFlow interface but it is still abstract and provides derived classes with methods for accrual period calculations.

Public Member Functions

- Coupon (Real nominal, const Date &paymentDate, const Date &accrualStartDate, const Date &accrualEndDate, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date())

Partial CashFlow interface

- Date date () const
  returns the date at which the cash flow is settled

Inspectors

- Real **nominal** () const
- const Date & accrualStartDate () const
  start of the accrual period
- const Date & accrualEndDate () const
  end of the accrual period
- Time accrualPeriod () const
  accrual period as fraction of year
- Integer accrualDays () const
  accrual period in days
- virtual Rate rate () const =0
  accrued rate
- virtual DayCounter dayCounter () const =0
  day counter for accrual calculation
- virtual Real accruedAmount (const Date &) const =0
  accrued amount at the given date

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Attributes

- Real **nominal_**
- Date **paymentDate_**
- Date **accrualStartDate_**
- Date **accrualEndDate_**
- Date **refPeriodStart_**
- Date **refPeriodEnd_**

7.134.2 Constructor & Destructor Documentation

7.134.2.1 Coupon (Real *nominal*, const Date & *paymentDate*, const Date & *accrualStartDate*, const Date & *accrualEndDate*, const Date & *refPeriodStart* = `Date()`, const Date & *refPeriodEnd* = `Date()`)

Warning:
the coupon does not adjust the payment date which must already be a business day.

7.135 CovarianceDecomposition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/getcovariance.hpp>`

7.135.1 Detailed Description

Extracts the correlation matrix and the vector of volatilities out of the input covariance matrix.

Note that only the lower symmetric part of the covariance matrix is used.

Precondition:
The covariance matrix must be symmetric.

Test
cross checked with getCovariance

Public Member Functions

- CovarianceDecomposition (const Matrix &covarianceMatrix, Real tolerance=1.0e-12)
- const Array & variances () const
- const Array & standardDeviations () const
- const Matrix & correlationMatrix () const

7.135.2 Constructor & Destructor Documentation

7.135.2.1 CovarianceDecomposition (const Matrix & *covarianceMatrix*, Real *tolerance* = 1.0e-12)

Precondition:
covarianceMatrix must be symmetric

7.135.3 Member Function Documentation

7.135.3.1 const Array& variances () const

returns the variances Array

7.135.3.2 const Array& standardDeviations () const

returns the standard deviations Array

7.135.3.3 const Matrix& correlationMatrix () const

returns the correlation matrix

7.136 CoxIngersollRoss Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/coxingersollross.hpp>`

Inheritance diagram for CoxIngersollRoss:

7.136.1 Detailed Description

Cox-Ingersoll-Ross model class.

This class implements the Cox-Ingersoll-Ross model defined by

$$dr_t = k(\theta - r_t)dt + \sqrt{r_t}\sigma dW_t.$$

Bug

this class was not tested enough to guarantee its functionality.

Public Member Functions

- **CoxIngersollRoss** (Rate r0=0.05, Real theta=0.1, Real k=0.1, Real sigma=0.1)
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- virtual boost::shared_ptr< ShortRateDynamics > dynamics () const

  returns the short-rate dynamics

- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Return by default a trinomial recombining tree.

Protected Member Functions

- Real **A** (Time t, Time T) const
- Real **B** (Time t, Time T) const
- Real **theta** () const
- Real **k** () const
- Real **sigma** () const
- Real **x0** () const

Classes

- class Dynamics

  Dynamics of the short-rate under the Cox-Ingersoll-Ross model

7.137 CoxIngersollRoss::Dynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>`

Inheritance diagram for CoxIngersollRoss::Dynamics:

7.137.1 Detailed Description

Dynamics of the short-rate under the Cox-Ingersoll-Ross model

The state variable y_t will here be the square-root of the short-rate. It satisfies the following stochastic equation

$$dy_t = \left[(\frac{k\theta}{2} + \frac{\sigma^2}{8}) \frac{1}{y_t} - \frac{k}{2} y_t \right] d_t + \frac{\sigma}{2} dW_t$$

.

Public Member Functions

- **Dynamics** (Real theta, Real k, Real sigma, Real x0)
- virtual Real **variable** (Time, Rate r) const
- virtual Real **shortRate** (Time, Real y) const

7.138 CoxRossRubinstein Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for CoxRossRubinstein:

7.138.1 Detailed Description

Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

Public Member Functions

- **CoxRossRubinstein** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)

7.139 CrankNicolson Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/cranknicolson.hpp>`

Inheritance diagram for CrankNicolson:

7.139.1 Detailed Description

template<class Operator> class QuantLib::CrankNicolson< Operator >

Crank-Nicolson scheme for finite difference methods.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type applyTo(const array_type&);
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:
The differential operator must be linear for this evolver to work.

Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **CrankNicolson** (const operator_type &L, const bc_set &bcs)

7.140 Cubic Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp>`

7.140.1 Detailed Description

cubic-spline interpolation factory and traits

Public Types

- enum { **global** = 1 }

Public Member Functions

- **Cubic** (CubicSpline::BoundaryCondition leftCondition=CubicSpline::SecondDerivative, Real leftConditionValue=0.0, CubicSpline::BoundaryCondition rightCondition=CubicSpline::SecondDerivative, Real rightConditionValue=0.0, bool monotonicityConstraint=false)
- template<class I1, class I2> Interpolation **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

7.141 CubicSpline Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp>`

Inheritance diagram for CubicSpline:

7.141.1 Detailed Description

Cubic spline interpolation between discrete points.

It implements different type of end conditions: not-a-knot, first derivative value, second derivative value.

It also implements Hyman's monotonicity constraint filter which ensures that the interpolating spline remains monotonic at the expense of the second derivative of the curve which will no longer be continuous where the filter has been applied. If the interpolating spline is already monotonic, the Hyman filter leaves it unchanged.

See R. L. Dougherty, A. Edelman, and J. M. Hyman, "Nonnegativity-, Monotonicity-, or Convexity-Preserving Cubic and Quintic Hermite Interpolation" Mathematics Of Computation, v. 52, n. 186, April 1989, pp. 471-494.

Test
the correctness of the returned values is tested by reproducing results available in literature.

Public Types

- enum BoundaryCondition {
  NotAKnot, FirstDerivative, SecondDerivative, Periodic,
  Lagrange }

Public Member Functions

- template<class I1, class I2> CubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, CubicSpline::BoundaryCondition leftCondition, Real leftConditionValue, CubicSpline::BoundaryCondition rightCondition, Real rightConditionValue, bool monotonicityConstraint)
- const std::vector< Real > & **aCoefficients** () const
- const std::vector< Real > & **bCoefficients** () const
- const std::vector< Real > & **cCoefficients** () const

7.141.2 Member Enumeration Documentation

7.141.2.1 enum BoundaryCondition

Enumerator:

NotAKnot Make second(-last) point an inactive knot.

FirstDerivative Match value of end-slope.

SecondDerivative Match value of second derivative at end.

Periodic Match first and second derivative at either end.

Lagrange Match end-slope to the slope of the cubic that matches the first four data at the respective end

7.141.3 Constructor & Destructor Documentation

7.141.3.1 CubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, CubicSpline::BoundaryCondition *leftCondition*, Real *leftConditionValue*, CubicSpline::BoundaryCondition *rightCondition*, Real *rightConditionValue*, bool *monotonicityConstraint*)

Precondition:
the x values must be sorted.

7.142 CumulativeBinomialDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/binomialdistribution.hpp>`

7.142.1 Detailed Description

Cumulative binomial distribution function.

Given an integer k it provides the cumulative probability of observing kk<=k: formula here ...

Public Member Functions

- **CumulativeBinomialDistribution** (Real p, BigNatural n)
- Real **operator()** (BigNatural k) const

7.143 CumulativeNormalDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/normaldistribution.hpp>`

7.143.1 Detailed Description

Cumulative normal distribution function.

Given x it provides an approximation to the integral of the gaussian normal distribution: formula here ...

For this implementation see M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Dover Publications, New York (1972)

Public Member Functions

- **CumulativeNormalDistribution** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const
- Real **derivative** (Real x) const

7.144 CumulativePoissonDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/poissondistribution.hpp>`

7.144.1 Detailed Description

Cumulative Poisson distribution function.

This function provides an approximation of the integral of the Poisson distribution.

For this implementation see "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

Test
the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **CumulativePoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

7.145 CuriouslyRecurringTemplate Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/curiouslyrecurring.hpp>`

Inheritance diagram for CuriouslyRecurringTemplate:

7.145.1 Detailed Description

template<class Impl> class QuantLib::CuriouslyRecurringTemplate< Impl >

Support for the curiously recurring template pattern.

See James O. Coplien. A Curiously Recurring Template Pattern. In Stanley B. Lippman, editor, C++ Gems, 135-144. Cambridge University Press, New York, New York, 1996.

Protected Member Functions

- Impl & **impl** ()
- const Impl & **impl** () const

7.146 Currency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/currency.hpp>`

Inheritance diagram for Currency:

7.146.1 Detailed Description

Currency specification

Public Member Functions

- Currency ()
  default constructor

Inspectors

- const std::string & name () const
  currency name, e.g, "U.S. Dollar"
- const std::string & code () const
  ISO 4217 three-letter code, e.g, "USD".
- Integer numericCode () const
  ISO 4217 numeric code, e.g, "840".
- const std::string & symbol () const
  symbol, e.g, "$"
- const std::string & fractionSymbol () const
  fraction symbol, e.g, "162"
- Integer fractionsPerUnit () const
  number of fractionary parts in a unit, e.g, 100
- const Rounding & rounding () const
  rounding convention
- std::string format () const
  output format

other info

- bool isValid () const
  is this a usable instance?
- const Currency & triangulationCurrency () const
  currency used for triangulated exchange when required

Protected Attributes

- boost::shared_ptr< Data > **data_**

Related Functions

(Note that these are not member functions.)

- bool **operator==** (const Currency &, const Currency &)
- bool **operator!=** (const Currency &, const Currency &)
- std::ostream & **operator**<< (std::ostream &, const Currency &)

7.146.2 Constructor & Destructor Documentation

7.146.2.1 Currency ()

default constructor

Instances built via this constructor have undefined behavior. Such instances can only act as placeholders and must be reassigned to a valid currency before being used.

7.146.3 Member Function Documentation

7.146.3.1 std::string format () const

output format

The format will be fed three positional parameters, namely, value, code, and symbol, in this order.

7.147 CYPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for CYPCurrency:

7.147.1 Detailed Description

Cyprus pound.

The ISO three-letter code is CYP; the numeric code is 196. It is divided in 100 cents.

7.148 CZKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for CZKCurrency:

7.148.1 Detailed Description

Czech koruna.

The ISO three-letter code is CZK; the numeric code is 203. It is divided in 100 haleru.

7.149 Date Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/date.hpp>`

7.149.1 Detailed Description

Concrete date class.

This class provides methods to inspect dates as well as methods and operators which implement a limited date algebra (increasing and decreasing dates, and calculating their difference).

Test
self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Public Member Functions

constructors

- Date ()
  Default constructor returning a null date.
- Date (BigInteger serialNumber)
  Constructor taking a serial number as given by Applix or Excel.
- Date (Day d, Month m, Year y)
  More traditional constructor.

inspectors

- Weekday **weekday** () const
- Day **dayOfMonth** () const
- Day dayOfYear () const
  One-based (Jan 1st = 1).
- Month **month** () const
- Year **year** () const
- BigInteger **serialNumber** () const

date algebra

- Date & operator+= (BigInteger days)
  increments date by the given number of days
- Date & operator+= (const Period &)
  increments date by the given period
- Date & operator-= (BigInteger days)
  decrement date by the given number of days
- Date & operator-= (const Period &)

decrements date by the given period

- Date & operator++ ()
  1-day pre-increment
- Date operator++ (int)
  1-day post-increment
- Date & operator– ()
  1-day pre-decrement
- Date operator– (int)
  1-day post-decrement
- Date operator+ (BigInteger days) const
  returns a new date incremented by the given number of days
- Date operator+ (const Period &) const
  returns a new date incremented by the given period
- Date operator- (BigInteger days) const
  returns a new date decremented by the given number of days
- Date operator- (const Period &) const
  returns a new date decremented by the given period

Static Public Member Functions

static methods

- static Date todaysDate ()
  today's date.
- static Date minDate ()
  earliest allowed date
- static Date maxDate ()
  latest allowed date
- static bool isLeap (Year y)
  whether the given year is a leap one
- static Date endOfMonth (const Date &d)
  last day of the month to which the given date belongs
- static bool isEOM (const Date &d)
  whether a date is the last day of its month
- static Date nextWeekday (const Date &d, Weekday)
  next given weekday following or equal to the given date
- static Date nthWeekday (Size n, Weekday, Month m, Year y)

n-th given weekday in the given month and year

- static bool isIMMdate (const Date &d)
  whether or not the given date is an IMM date
- static Date nextIMMdate (const Date &d)
  next IMM date following (or equal to) the given date

Related Functions

(Note that these are not member functions.)

- BigInteger operator- (const Date &, const Date &)
  Difference in days between dates.
- bool **operator==** (const Date &, const Date &)
- bool **operator!=** (const Date &, const Date &)
- bool **operator**< (const Date &, const Date &)
- bool **operator**<= (const Date &, const Date &)
- bool **operator**> (const Date &, const Date &)
- bool **operator**>= (const Date &, const Date &)
- std::ostream & **operator**<< (std::ostream &, const Date &)

7.149.2 Member Function Documentation

7.149.2.1 static Date nextWeekday (const Date & *d*, Weekday) [static]

next given weekday following or equal to the given date

E.g., the Friday following January 15th, 20 (a Tuesday) was January 18th, 2002.

see http://www.cpearson.com/excel/DateTimeWS.htm

7.149.2.2 static Date nthWeekday (Size *n*, Weekday, Month *m*, Year *y*) [static]

n-th given weekday in the given month and year

E.g., the 4th Thursday of March, 1998 was March 26th, 1998.

see http://www.cpearson.com/excel/DateTimeWS.htm

7.149.2.3 static Date nextIMMdate (const Date & *d*) [static]

next IMM date following (or equal to) the given date

returns the 1st delivery date for next contract listed in the International Money Market section of the Chicago Mercantile Exchange.

Warning:
The result date is following or equal to the original date

7.150 DayCounter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/daycounter.hpp>`

Inheritance diagram for DayCounter:

7.150.1 Detailed Description

day counter class

This class provides methods for determining the length of a time period according to given market convention, both as a number of days and as a year fraction.

The Bridge pattern is used to provide the base behavior of the day counter.

Public Member Functions

- DayCounter ()

DayCounter interface

- std::string name () const
  Returns the name of the day counter.
- BigInteger dayCount (const Date &, const Date &) const
  Returns the number of days between two dates.
- Time yearFraction (const Date &, const Date &, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date()) const
  Returns the period between two dates as a fraction of year.

Protected Member Functions

- DayCounter (const boost::shared_ptr< DayCounterImpl > &impl)

Related Functions

(Note that these are not member functions.)

- bool operator== (const DayCounter &, const DayCounter &)
- bool **operator!=** (const DayCounter &, const DayCounter &)

7.150.2 Constructor & Destructor Documentation

7.150.2.1 DayCounter ()

This default constructor returns a day counter with a null implementation, which is therefore unusable except as a placeholder.

7.150.2.2 DayCounter (const boost::shared_ptr< DayCounterImpl > & *impl*) [protected]

This protected constructor will only be invoked by derived classes which define a given DayCounter implementation

7.150.3 Member Function Documentation

7.150.3.1 std::string name () const

Returns the name of the day counter.

Warning:

This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

7.150.4 Friends And Related Function Documentation

7.150.4.1 bool operator== (const DayCounter &, const DayCounter &) [related]

Returns `true` iff the two day counters belong to the same derived class.

7.151 DayCounterImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/daycounter.hpp>`

7.151.1 Detailed Description

abstract base class for day counter implementations

Public Member Functions

- virtual std::string **name** () const =0
- virtual BigInteger dayCount (const Date &d1, const Date &d2) const

  to be overloaded by more complex day counters

- virtual Time **yearFraction** (const Date &, const Date &, const Date &refPeriodStart, const Date &refPeriodEnd) const =0

7.152 DEMCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for DEMCurrency:

7.152.1 Detailed Description

Deutsche mark.

The ISO three-letter code was DEM; the numeric code was 276. It was divided into 100 pfennig.

7.153 DepositRateHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp>`

Inheritance diagram for DepositRateHelper:

7.153.1 Detailed Description

Deposit rate.

Warning:
This class assumes that the reference date does not change between calls of setTermStructure().

Public Member Functions

- **DepositRateHelper** (const Handle< Quote > &rate, Integer n, TimeUnit units, Integer settlementDays, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- **DepositRateHelper** (Rate rate, Integer n, TimeUnit units, Integer settlementDays, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void setTermStructure (YieldTermStructure *)

  sets the term structure to be used for pricing

- Date latestDate () const

  latest relevant date

7.153.2 Member Function Documentation

7.153.2.1 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:
Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from RateHelper.

7.153.2.2 Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements RateHelper.

7.154 DerivedQuote Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/quote.hpp>`

Inheritance diagram for DerivedQuote:

7.154.1 Detailed Description

template<class UnaryFunction> class QuantLib::DerivedQuote< UnaryFunction >

market element whose value depends on another market element

Test

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **DerivedQuote** (const Handle< Quote > &element, const UnaryFunction &f)

Market element interface

- Real value () const
  returns the current value

Observer interface

- void update ()

7.154.2 Member Function Documentation

7.154.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.155 DirichletBC Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/boundarycondition.hpp>`

Inheritance diagram for DirichletBC:

7.155.1 Detailed Description

Neumann boundary condition (i.e., constant value).

Todo

generalize to time-dependent conditions.

Public Member Functions

- **DirichletBC** (Real value, Side side)
- void applyBeforeApplying (TridiagonalOperator &) const
- void applyAfterApplying (Array &) const
- void applyBeforeSolving (TridiagonalOperator &, Array &rhs) const
- void applyAfterSolving (Array &) const
- void setTime (Time)

7.155.2 Member Function Documentation

7.155.2.1 void applyBeforeApplying (TridiagonalOperator &) const [virtual]

This method modifies an operator L before it is applied to an array u so that $v = Lu$ will satisfy the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.155.2.2 void applyAfterApplying (Array &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.155.2.3 void applyBeforeSolving (TridiagonalOperator &, Array & *rhs*) const [virtual]

This method modifies an operator L before the linear system $Lu' = u$ is solved so that u' will satisfy the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.155.2.4 void applyAfterSolving (Array &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.155.2.5 void setTime (Time) [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements BoundaryCondition< TridiagonalOperator >.

7.156 Discount Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bootstraptraits.hpp>`

7.156.1 Detailed Description

Discount-curve traits.

Static Public Member Functions

- static DiscountFactor **initialValue** ()
- static DiscountFactor **initialGuess** ()
- static DiscountFactor **guess** (const YieldTermStructure *c, const Date &d)
- static DiscountFactor **minValueAfter** (Size, const std::vector< Real > &)
- static DiscountFactor **maxValueAfter** (Size i, const std::vector< Real > &data)
- static void **updateGuess** (std::vector< DiscountFactor > &data, DiscountFactor discount, Size i)

7.157 DiscrepancyStatistics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/discrepancystatistics.hpp>`

Inheritance diagram for DiscrepancyStatistics:

7.157.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from SequenceStatistics<Statistics> and adds L^2 discrepancy calculation

Public Member Functions

- **DiscrepancyStatistics** (Size dimension)
- template<class Sequence> void **add** (const Sequence &sample, Real weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, Real weight=1.0)
- void **reset** (Size dimension=0)

1-dimensional inspectors

- Real **discrepancy** () const

7.158 DiscreteAveragingAsianOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

Inheritance diagram for DiscreteAveragingAsianOption:

7.158.1 Detailed Description

Discrete-averaging Asian option.

Public Member Functions

- **DiscreteAveragingAsianOption** (Average::Type averageType, Real runningAccumulator, Size pastFixings, std::vector< Date > fixingDates, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Protected Attributes

- Average::Type **averageType_**
- Real **runningAccumulator_**
- Size **pastFixings_**
- std::vector< Date > **fixingDates_**

Classes

- class arguments

  Extra arguments for single-asset discrete-average Asian option.

- class engine

  Discrete-averaging Asian engine base class.

7.158.2 Member Function Documentation

7.158.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.159 DiscreteAveragingAsianOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

7.159.1 Detailed Description

Extra arguments for single-asset discrete-average Asian option.

Public Member Functions

- void **validate** () const

Public Attributes

- Average::Type **averageType**
- Real **runningAccumulator**
- Size **pastFixings**
- std::vector< Date > **fixingDates**

7.160 DiscreteAveragingAsianOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hpp>`

Inheritance diagram for DiscreteAveragingAsianOption::engine:

7.160.1 Detailed Description

Discrete-averaging Asian engine base class.

7.161 DiscreteGeometricASO Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/discretegeometricaso.hpp>`

Inheritance diagram for DiscreteGeometricASO:

7.161.1 Detailed Description

Discrete geometric average-strike Asian option (European style).

This class implements a discrete geometric average strike asian option, with european exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag65-97

Todo

add analytical greeks

Public Member Functions

- **DiscreteGeometricASO** (Option::Type type, Real underlying, Spread dividendYield, Rate riskFreeRate, const std::vector< Time > &times, Volatility volatility)
- Real **value** () const
- Real **delta** () const
- Real **gamma** () const
- Real **theta** () const
- boost::shared_ptr< SingleAssetOption > **clone** () const

7.162 DiscretizedAsset Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/discretizedasset.hpp>`

Inheritance diagram for DiscretizedAsset:

7.162.1 Detailed Description

Discretized asset class used by numerical methods.

Public Member Functions

inspectors

- Time **time** () const
- Time & **time** ()
- const Array & **values** () const
- Array & **values** ()
- const boost::shared_ptr< NumericalMethod > & **method** () const

High-level interface

Users of discretized assets should use these methods in order to initialize, evolve and take the present value of the assets. They call the corresponding methods in the NumericalMethod interface, to which we refer for documentation.

- void **initialize** (const boost::shared_ptr< NumericalMethod > &, Time t)
- void **rollback** (Time to)
- void **partialRollback** (Time to)
- Real **presentValue** ()

Low-level interface

These methods (that developers should override when deriving from DiscretizedAsset) are to be used by numerical methods and not directly by users, with the exception of adjustValues(), preAdjustValues() and postAdjustValues() that can be used together with partialRollback().

- virtual void reset (Size size)=0
- void preAdjustValues ()
- void postAdjustValues ()
- void adjustValues ()
- virtual std::vector< Time > mandatoryTimes () const =0

Protected Member Functions

- bool isOnTime (Time t) const
- virtual void preAdjustValuesImpl ()
- virtual void postAdjustValuesImpl ()

Protected Attributes

- Time **time_**
- Time **latestPreAdjustment_**
- Time **latestPostAdjustment_**
- Array **values_**

7.162.2 Member Function Documentation

7.162.2.1 virtual void reset (Size *size*) [pure virtual]

This method should initialize the asset values to an Array of the given size and with values depending on the particular asset.

Implemented in DiscretizedDiscountBond, and DiscretizedOption.

7.162.2.2 void preAdjustValues ()

This method will be invoked after rollback and before any other asset (i.e., an option on this one) has any chance to look at the values. For instance, payments happening at times already spanned by the rollback will be added here.

This method is not virtual; derived classes must override the protected preAdjustValuesImpl() method instead.

7.162.2.3 void postAdjustValues ()

This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

This method is not virtual; derived classes must override the protected postAdjustValuesImpl() method instead.

7.162.2.4 void adjustValues ()

This method performs both pre- and post-adjustment

7.162.2.5 virtual std::vector<Time> mandatoryTimes () const [pure virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:
The returned values are not guaranteed to be sorted.

Implemented in DiscretizedDiscountBond, and DiscretizedOption.

7.162.2.6 bool isOnTime (Time *t*) const [protected]

This method checks whether the asset was rolled at the given time.

7.162.2.7 virtual void preAdjustValuesImpl () [protected, virtual]

This method performs the actual pre-adjustment

7.162.2.8 virtual void postAdjustValuesImpl () [protected, virtual]

This method performs the actual post-adjustment

Reimplemented in DiscretizedOption.

7.163 DiscretizedDiscountBond Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/discretizedasset.hpp>`

Inheritance diagram for DiscretizedDiscountBond:

7.163.1 Detailed Description

Useful discretized discount bond asset.

Public Member Functions

- void reset (Size size)
- std::vector< Time > mandatoryTimes () const

7.163.2 Member Function Documentation

7.163.2.1 void reset (Size *size*) [virtual]

This method should initialize the asset values to an Array of the given size and with values depending on the particular asset.

Implements DiscretizedAsset.

7.163.2.2 std::vector<Time> mandatoryTimes () const [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:
The returned values are not guaranteed to be sorted.

Implements DiscretizedAsset.

7.164 DiscretizedOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/discretizedasset.hpp>`

Inheritance diagram for DiscretizedOption:

7.164.1 Detailed Description

Discretized option on a given asset.

Warning:
it is advised that derived classes take care of creating and initializing themselves an instance of the underlying.

Public Member Functions

- **DiscretizedOption** (const boost::shared_ptr< DiscretizedAsset > &underlying, Exercise::Type exerciseType, const std::vector< Time > &exerciseTimes)
- void reset (Size size)
- std::vector< Time > mandatoryTimes () const

Protected Member Functions

- void postAdjustValuesImpl ()
- void **applyExerciseCondition** ()

Protected Attributes

- boost::shared_ptr< DiscretizedAsset > **underlying_**
- Exercise::Type **exerciseType_**
- std::vector< Time > **exerciseTimes_**

7.164.2 Member Function Documentation

7.164.2.1 void reset (Size *size*) [virtual]

This method should initialize the asset values to an Array of the given size and with values depending on the particular asset.

Implements DiscretizedAsset.

7.164.2.2 std::vector< Time > mandatoryTimes () const [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:
The returned values are not guaranteed to be sorted.

Implements DiscretizedAsset.

7.164.2.3 void postAdjustValuesImpl () [protected, virtual]

This method performs the actual post-adjustment

Reimplemented from DiscretizedAsset.

7.165 Disposable Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/disposable.hpp>`

7.165.1 Detailed Description

template<class T> class QuantLib::Disposable< T >

generic disposable object with move semantics

This class can be used for returning a value by copy. It relies on the returned object exposing a `swap(T&)` method through which the copy constructor and assignment operator are implemented, thus resulting in actual move semantics. Typical use of this class is along the following lines:

```
Disposable<Foo> bar(Integer i) {
    Foo f(i*2);
    return f;
}
```

Warning:

In order to avoid copies in code such as shown above, the conversion from `T` to `Disposable`<T> is destructive, i.e., it does **not** preserve the state of the original object. Therefore, it is necessary for the developer to avoid code such as

```
Disposable<Foo> bar(Foo& f) {
    return f;
}
```

which would likely render the passed object unusable. The correct way to obtain the desired behavior would be:

```
Disposable<Foo> bar(Foo& f) {
    Foo temp = f;
    return temp;
}
```

Public Member Functions

- **Disposable** (T &t)
- **Disposable** (const Disposable< T > &t)
- Disposable< T > & **operator=** (const Disposable< T > &t)

7.166 DividendVanillaOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/dividendvanillaoption.hpp>`

Inheritance diagram for DividendVanillaOption:

7.166.1 Detailed Description

Single-asset vanilla option (no barriers) with discrete dividends.

Public Member Functions

- **DividendVanillaOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const std::vector< Date > &dividendDates, const std::vector< Real > &dividends, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())

Protected Member Functions

- void setupArguments (Arguments *) const

Classes

- class arguments

  Arguments for dividend vanilla option calculation

- class engine

  Dividend vanilla option engine base class.

7.166.2 Member Function Documentation

7.166.2.1 void setupArguments (Arguments *) const [protected, virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.167 DividendVanillaOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/dividendvanillaoption.hpp>`

7.167.1 Detailed Description

Arguments for dividend vanilla option calculation

Public Member Functions

- void **validate** () const

7.168 DividendVanillaOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/dividendvanillaoption.hpp>`

Inheritance diagram for DividendVanillaOption::engine:

7.168.1 Detailed Description

Dividend vanilla option engine base class.

7.169 DKKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for DKKCurrency:

7.169.1 Detailed Description

Danish krone.

The ISO three-letter code is DKK; the numeric code is 208. It is divided in 100 øre.

7.170 DKKLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/dkklibor.hpp>`

Inheritance diagram for DKKLibor:

7.170.1 Detailed Description

DKK LIBOR rate

Danish Krona LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Public Member Functions

- **DKKLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.171 DMinus Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/dminus.hpp>`

Inheritance diagram for DMinus:

7.171.1 Detailed Description

D_- matricial representation

The differential operator D_- discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_i - u_{i-1}}{h} = D_- u_i$$

Public Member Functions

- **DMinus** (Size gridPoints, Real h)

7.172 DownRounding Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for DownRounding:

7.172.1 Detailed Description

Down-rounding.

Public Member Functions

- **DownRounding** (Integer precision, Integer digit=5)

7.173 DPlus Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/dplus.hpp>`

Inheritance diagram for DPlus:

7.173.1 Detailed Description

D_+ matricial representation

The differential operator D_+ discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_i}{h} = D_+ u_i$$

Public Member Functions

- **DPlus** (Size gridPoints, Real h)

7.174 DPlusDMinus Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/dplusdminus.hpp>`

Inheritance diagram for DPlusDMinus:

7.174.1 Detailed Description

D_+D_- matricial representation

The differential operator D_+D_- discretizes the second derivative with the second-order formula

$$\frac{\partial^2 u_i}{\partial x^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = D_+D_-u_i$$

Test

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **DPlusDMinus** (Size gridPoints, Real h)

7.175 DriftTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/drifttermstructure.hpp>`

Inheritance diagram for DriftTermStructure:

7.175.1 Detailed Description

Drift term structure.

Drift term structure for modelling the common drift term: riskFreeRate - dividendYield - 0.5∗vol∗vol

Note:
This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **DriftTermStructure** (const Handle< YieldTermStructure > &riskFreeTS, const Handle< YieldTermStructure > &dividendTS, const Handle< BlackVolTermStructure > &blackVolTS)

YieldTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Calendar calendar () const
  the calendar used for reference date calculation
- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- Date maxDate () const
  the latest date for which the curve can return rates

Protected Member Functions

- Rate zeroYieldImpl (Time) const
  returns the discount factor as seen from the evaluation date

7.176 Duration Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/analysis.hpp>`

7.176.1 Detailed Description

duration type

Public Types

- enum **Type** { **Simple**, **Macaulay**, **Modified** }

7.177 DZero Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/dzero.hpp>`

Inheritance diagram for DZero:

7.177.1 Detailed Description

D_0 matricial representation

The differential operator D_0 discretizes the first derivative with the second-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_{i-1}}{2h} = D_0 u_i$$

Test

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **DZero** (Size gridPoints, Real h)

7.178 EarlyExercise Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp>`

Inheritance diagram for EarlyExercise:

7.178.1 Detailed Description

Early-exercise base class.

The payoff can be at exercise (default case) or at expiry

Todo

derive a plain American Exercise class (no earliestDate, no payoffAtExpiry)

Public Member Functions

- **EarlyExercise** (Type type=Undefined, bool payoffAtExpiry=false)
- bool **payoffAtExpiry** () const

7.179 EEKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for EEKCurrency:

7.179.1 Detailed Description

Estonian kroon.

The ISO three-letter code is EEK; the numeric code is 233. It is divided in 100 senti.

7.180 EndCriteria Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/criteria.hpp>`

7.180.1 Detailed Description

Criteria to end optimization process.

- stationary point
  - stationary gradient
  - maximum number of iterations ....

Public Types

- enum **Type** { **none**, **maxIter**, **statPt**, **statGd** }

Public Member Functions

- EndCriteria ()
  default constructor
- EndCriteria (Size maxIteration, Real epsilon)
  initialization constructor
- void **setPositiveOptimization** ()
- bool **checkIterationNumber** (Size iteration)
- bool **checkStationaryValue** (Real fold, Real fnew)
- bool **checkAccuracyValue** (Real f)
- bool **checkStationaryGradientNorm** (Real normDiff)
- bool **checkAccuracyGradientNorm** (Real norm)
- bool operator() (Size iteration, Real fold, Real normgold, Real fnew, Real normgnew, Real)
  test if the number of iteration is not too big and if we don't
- Type criteria () const
  return the end criteria type

Protected Attributes

- Size maxIteration_
  Maximum number of iterations.
- Real functionEpsilon_
  function and gradient epsilons
- Real **gradientEpsilon_**
- Size maxIterStatPt_

Maximun number of iterations in stationary state.

- Size **statState_**
- Type **endCriteria_**
- bool **positiveOptimization_**

7.181 EqualJumpsBinomialTree Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for EqualJumpsBinomialTree:

7.181.1 Detailed Description

template<class T> class QuantLib::EqualJumpsBinomialTree< T >

Base class for equal jumps binomial tree.

Public Member Functions

- **EqualJumpsBinomialTree** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

Protected Attributes

- Real **dx_**
- Real **pu_**
- Real **pd_**

7.182 EqualProbabilitiesBinomialTree Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for EqualProbabilitiesBinomialTree:

7.182.1 Detailed Description

template<class T> class QuantLib::EqualProbabilitiesBinomialTree< T >

Base class for equal probabilities binomial tree.

Public Member Functions

- **EqualProbabilitiesBinomialTree** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size) const

Protected Attributes

- Real **up_**

7.183 Error Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/errors.hpp>`

7.183.1 Detailed Description

Base error class.

Public Member Functions

- Error (const std::string &file, long line, const std::string &function, const std::string &message="")
- ~Error () throw ()
- const char * what () const throw ()

  returns the error message.

7.183.2 Constructor & Destructor Documentation

7.183.2.1 Error (const std::string & *file*, long *line*, const std::string & *function*, const std::string & *message* = "")

The explicit use of this constructor is not advised. Use the QL_FAIL macro instead.

7.183.2.2 ~Error () throw ()

the automatically generated destructor would not have the throw specifier.

7.184 ErrorFunction Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/errorfunction.hpp>`

7.184.1 Detailed Description

Error function

formula here ... Used to calculate the cumulative normal distribution function

Public Member Functions

- Real **operator()** (Real x) const

7.185 ESPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for ESPCurrency:

7.185.1 Detailed Description

Spanish peseta.

The ISO three-letter code is ESP; the numeric code is 724. It is divided in 100 centimos.

7.186 EulerDiscretization Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/eulerdiscretization.hpp>`

Inheritance diagram for EulerDiscretization:

7.186.1 Detailed Description

Euler discretization for stochastic processes.

Public Member Functions

- Disposable< Array > drift (const StochasticProcess &, Time t0, const Array &x0, Time dt) const
- Real drift (const StochasticProcess1D &, Time t0, Real x0, Time dt) const
- Disposable< Matrix > diffusion (const StochasticProcess &, Time t0, const Array &x0, Time dt) const
- Real diffusion (const StochasticProcess1D &, Time t0, Real x0, Time dt) const
- Disposable< Matrix > covariance (const StochasticProcess &, Time t0, const Array &x0, Time dt) const
- Real variance (const StochasticProcess1D &, Time t0, Real x0, Time dt) const

7.186.2 Member Function Documentation

7.186.2.1 Disposable<Array> drift (const StochasticProcess &, Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

Returns an approximation of the drift defined as $\mu(t_0, \mathbf{x}_0)\Delta t$.

Implements StochasticProcess::discretization.

7.186.2.2 Real drift (const StochasticProcess1D &, Time *t0*, Real *x0*, Time *dt*) const [virtual]

Returns an approximation of the drift defined as $\mu(t_0, x_0)\Delta t$.

Implements StochasticProcess1D::discretization.

7.186.2.3 Disposable<Matrix> diffusion (const StochasticProcess &, Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

Returns an approximation of the diffusion defined as $\sigma(t_0, \mathbf{x}_0)\sqrt{\Delta t}$.

Implements StochasticProcess::discretization.

7.186.2.4 Real diffusion (const StochasticProcess1D &, Time *t0*, Real *x0*, Time *dt*) const [virtual]

Returns an approximation of the diffusion defined as $\sigma(t_0, x_0)\sqrt{\Delta t}$.

Implements StochasticProcess1D::discretization.

7.186.2.5 Disposable<Matrix> covariance (const StochasticProcess &, Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

Returns an approximation of the covariance defined as $\sigma(t_0, \mathbf{x}_0)^2 \Delta t$.

Implements StochasticProcess::discretization.

7.186.2.6 Real variance (const StochasticProcess1D &, Time *t0*, Real *x0*, Time *dt*) const [virtual]

Returns an approximation of the variance defined as $\sigma(t_0, x_0)^2 \Delta t$.

Implements StochasticProcess1D::discretization.

7.187 EURCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for EURCurrency:

7.187.1 Detailed Description

European Euro.

The ISO three-letter code is EUR; the numeric code is 978. It is divided into 100 cents.

7.188 Euribor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/euribor.hpp>`

Inheritance diagram for Euribor:

7.188.1 Detailed Description

Euribor index

Euribor rate fixed by the ECB.

Warning:
This is the rate fixed by the ECB. Use EURLibor if you're interested in the London fixing by BBA.

Public Member Functions

- **Euribor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.189 EURLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/eurlibor.hpp>`

Inheritance diagram for EURLibor:

7.189.1 Detailed Description

EUR LIBOR rate

Euro LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Warning:
This is the rate fixed in London by BBA. Use Euribor if you're interested in the fixing by the ECB.

Public Member Functions

- **EURLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.190 EuropeanExercise Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp>`

Inheritance diagram for EuropeanExercise:

7.190.1 Detailed Description

European exercise.

A European option can only be exercised at one (expiry) date.

Public Member Functions

- **EuropeanExercise** (const Date &date)

7.191 EuropeanOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/europeanoption.hpp>`

Inheritance diagram for EuropeanOption:

7.191.1 Detailed Description

European option on a single asset.

Public Member Functions

- **EuropeanOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())

7.192 ExchangeRate Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exchangerate.hpp>`

7.192.1 Detailed Description

exchange rate between two currencies

Test
application of direct and derived exchange rate is tested against calculations.

Utility methods

- Money exchange (const Money &amount) const
  apply the exchange rate to a cash amount
- static ExchangeRate chain (const ExchangeRate &r1, const ExchangeRate &r2)
  chain two exchange rates

Public Types

- enum Type { Direct, Derived }

Public Member Functions

Constructors

- ExchangeRate (const Currency &source, const Currency &target, Decimal rate)

Inspectors

- const Currency & source () const
  the source currency.
- const Currency & target () const
  the target currency.
- Type type () const
  the type
- Decimal rate () const
  the exchange rate (when available)

7.192.2 Member Enumeration Documentation

7.192.2.1 enum Type

Enumerator:

Direct given directly by the user

Derived derived from exchange rates between other currencies

7.192.3 Constructor & Destructor Documentation

7.192.3.1 ExchangeRate (const Currency & *source*, const Currency & *target*, Decimal *rate*)

the rate *r* is given with the convention that a unit of the source is worth *r* units of the target.

7.193 ExchangeRateManager Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/exchangeratemanager.hpp>`

Inheritance diagram for ExchangeRateManager:

7.193.1 Detailed Description

exchange-rate repository

Test
lookup of direct, triangulated, and derived exchange rates is tested.

Public Member Functions

- void add (const ExchangeRate &, const Date &startDate=Date::minDate(), const Date &endDate=Date::maxDate())

  Add an exchange rate.

- ExchangeRate lookup (const Currency &source, const Currency &target, Date date=Date(), ExchangeRate::Type type=ExchangeRate::Derived) const
- void clear ()

  remove the added exchange rates

Friends

- class **Singleton**< **ExchangeRateManager** >

7.193.2 Member Function Documentation

7.193.2.1 void add (const ExchangeRate &, const Date & *startDate* = `Date::minDate()`, const Date & *endDate* = `Date::maxDate()`)

Add an exchange rate.

The given rate is valid between the given dates.

Note:
If two rates are given between the same currencies and with overlapping date ranges, the latest one added takes precedence during lookup.

7.193.2.2 ExchangeRate lookup (const Currency & *source*, const Currency & *target*, Date *date* = `Date()`, ExchangeRate::Type *type* = `ExchangeRate::Derived`) const

Lookup the exchange rate between two currencies at a given date. If the given type is Direct, only direct exchange rates will be returned if available; if Derived, direct rates are still preferred but derived rates are allowed.

Warning:
if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

7.194 Exercise Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp>`

Inheritance diagram for Exercise:

7.194.1 Detailed Description

Base exercise class.

Public Types

- enum **Type** { **Undefined** = -1, **American**, **Bermudan**, **European** }

Public Member Functions

- **Exercise** (Type type=Undefined)
- bool **isNull** () const
- Type **type** () const
- Date **date** (Size index) const
- const std::vector< Date > & **dates** () const
- Date **lastDate** () const

Protected Attributes

- std::vector< Date > **dates_**
- Type **type_**

7.195 ExplicitEuler Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/expliciteuler.hpp>`

Inheritance diagram for ExplicitEuler:

7.195.1 Detailed Description

template<class Operator> class QuantLib::ExplicitEuler< Operator >

Forward Euler scheme for finite difference methods.

See sect. Finite-differences framework for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type applyTo(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator-(const Operator&, const Operator&);
```

Todo

add Richardson extrapolation

Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_type **bc_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **ExplicitEuler** (const operator_type &L, const std::vector< boost::shared_ptr< bc_type > > &bcs)

7.196 ExtendedCoxIngersollRoss Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss:

7.196.1 Detailed Description

Extended Cox-Ingersoll-Ross model class.

This class implements the extended Cox-Ingersoll-Ross model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sqrt{r_t}\sigma dW_t.$$

Bug

this class was not tested enough to guarantee its functionality.

Public Member Functions

- **ExtendedCoxIngersollRoss** (const Handle< YieldTermStructure > &termStructure, Real theta=0.1, Real k=0.1, Real sigma=0.1, Real x0=0.05)
- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Return by default a trinomial recombining tree.

- boost::shared_ptr< ShortRateDynamics > dynamics () const

  returns the short-rate dynamics

- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const

Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const

Classes

- class Dynamics

  Short-rate dynamics in the extended Cox-Ingersoll-Ross model.

- class FittingParameter

  Analytical term-structure fitting parameter $\varphi(t)$.

7.197 ExtendedCoxIngersollRoss::Dynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:

7.197.1 Detailed Description

Short-rate dynamics in the extended Cox-Ingersoll-Ross model.

The short-rate is here

$$r_t = \varphi(t) + y_t^2$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and y_t is the state variable, the square-root of a standard CIR process.

Public Member Functions

- **Dynamics** (const Parameter &phi, Real theta, Real k, Real sigma, Real x0)
- virtual Real **variable** (Time t, Rate r) const
- virtual Real **shortRate** (Time t, Real y) const

7.198 ExtendedCoxIngersollRoss::FittingParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:

7.198.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) - \frac{2k\theta(e^{th}-1)}{2h+(k+h)(e^{th}-1)} - \frac{4x_0h^2e^{th}}{(2h+(k+h)(e^{th}-1))^1},$$

where $f(t)$ is the instantaneous forward rate at t and $h = \sqrt{k^2+2\sigma^2}$.

Public Member Functions

- **FittingParameter** (const Handle< YieldTermStructure > &termStructure, Real theta, Real k, Real sigma, Real x0)

7.199 ExtendedDiscountCurve Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/extendeddiscountcurve.hpp>`

Inheritance diagram for ExtendedDiscountCurve:

7.199.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuos compounding.

Public Member Functions

- **ExtendedDiscountCurve** (const std::vector< Date > &dates, const std::vector< DiscountFactor > &dfs, const Calendar &calendar, const BusinessDayConvention conv, const DayCounter &dayCounter)
- Calendar calendar () const

  the calendar used for reference date calculation

- BusinessDayConvention **businessDayConvention** () const
- void update ()
- Rate **compoundForward** (const Date &d1, Integer f, bool extrapolate=false) const
- Rate **compoundForward** (Time t1, Integer f, bool extrapolate=false) const

Protected Member Functions

- Rate compoundForwardImpl (Time, Integer) const
- Rate zeroYieldImpl (Time) const
- void **calibrateNodes** () const
- boost::shared_ptr< CompoundForward > **reversebootstrap** (Integer) const
- boost::shared_ptr< CompoundForward > **forwardCurve** (Integer) const

7.199.2 Member Function Documentation

7.199.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from TermStructure.

7.199.2.2 Rate compoundForwardImpl (Time, Integer) const [protected]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

7.199.2.3 Rate zeroYieldImpl (Time) const [protected]

Returns the zero yield rate for the given date calculating it from the discount.

7.200 Extrapolator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/extrapolation.hpp>`

Inheritance diagram for Extrapolator:

7.200.1 Detailed Description

base class for classes possibly allowing extrapolation

Public Member Functions

modifiers

- void enableExtrapolation ()
  enable extrapolation in subsequent calls
- void disableExtrapolation ()
  disable extrapolation in subsequent calls

inspectors

- bool allowsExtrapolation () const
  tells whether extrapolation is enabled

7.201 Factorial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/factorial.hpp>`

7.201.1 Detailed Description

Factorial numbers calculator

Test
the correctness of the returned value is tested by checking it against numerical calculations.

Static Public Member Functions

- static Real **get** (Natural n)
- static Real **ln** (Natural n)

7.202 FalsePosition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/falseposition.hpp>`

Inheritance diagram for FalsePosition:

7.202.1 Detailed Description

False position 1-D solver.

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.203 FaureRsg Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/faurersg.hpp>`

7.203.1 Detailed Description

Faure low-discrepancy sequence generator.

It is based on existing Fortran and C algorithms to calculate pascal matrix and gray transforms.

1. E. Thiemard Economic generation of low-discrepancy sequences with a b-ary gray code.
2. Algorithms 659, 647. http://www.netlib.org/toms/647, http://www.netlib.org/toms/659

Test
the correctness of the returned values is tested by reproducing known good values.

Public Types

- typedef Sample< Array > **sample_type**

Public Member Functions

- **FaureRsg** (Size dimensionality)
- const std::vector< long int > & **nextIntSequence** () const
- const std::vector< long int > & **lastIntSequence** () const
- const sample_type & **nextSequence** () const
- const sample_type & **lastSequence** () const
- Size **dimension** () const

7.204 FDAmericanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdamericanengine.hpp>`

Inheritance diagram for FDAmericanEngine:

7.204.1 Detailed Description

Finite-differences pricing engine for American one asset options.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **FDAmericanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

7.205 FDBermudanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdbermudanengine.hpp>`

Inheritance diagram for FDBermudanEngine:

7.205.1 Detailed Description

Finite-differences Bermudan engine.

Public Member Functions

- **FDBermudanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- void **calculate** () const

Protected Member Functions

- void **initializeStepCondition** () const
- void **executeIntermediateStep** (Size) const

Protected Attributes

- Real **extraTermInBermudan**

7.206 FDDividendAmericanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fddividendamericanengine.hpp>`

Inheritance diagram for FDDividendAmericanEngine:

7.206.1 Detailed Description

Finite-differences pricing engine for dividend American options.

Test

- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

Bug

method impliedVolatility() utterly fails

Public Member Functions

- **FDDividendAmericanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- void **calculate** () const

Protected Member Functions

- void **initializeStepCondition** () const

7.207 FDDividendEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fddividendengine.hpp>`

Inheritance diagram for FDDividendEngine:

7.207.1 Detailed Description

Base finite-differences pricing engine for dividend options.

Public Member Functions

- **FDDividendEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

7.208 FDDividendEuropeanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fddividendeuropeanengine.hpp>`

Inheritance diagram for FDDividendEuropeanEngine:

7.208.1 Detailed Description

Finite-differences pricing engine for dividend European options.

Test

- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

Public Member Functions

- **FDDividendEuropeanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- void **calculate** () const

7.209 FDDividendShoutEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fddividendshoutengine.hpp>`

Inheritance diagram for FDDividendShoutEngine:

7.209.1 Detailed Description

Finite-differences shout engine with dividends.

Public Member Functions

- **FDDividendShoutEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- void **calculate** () const

Protected Member Functions

- void **initializeStepCondition** () const

7.210 FDEuropeanEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdeuropeanengine.hpp>`

Inheritance diagram for FDEuropeanEngine:

7.210.1 Detailed Description

Pricing engine for European options using finite-differences.

Test
the correctness of the returned value is tested by checking it against analytic results.

Public Member Functions

- **FDEuropeanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

7.211 FDShoutEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdshoutengine.hpp>`

Inheritance diagram for FDShoutEngine:

7.211.1 Detailed Description

Finite-differences pricing engine for shout vanilla options.

Test
the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **FDShoutEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

7.212 FDStepConditionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdstepconditionengine.hpp>`

Inheritance diagram for FDStepConditionEngine:

7.212.1 Detailed Description

Finite-differences pricing engine for American-style vanilla options.

Public Member Functions

- **FDStepConditionEngine** (Size timeSteps, Size gridPoints, bool timeDependent=false)

Protected Member Functions

- virtual void **initializeStepCondition** () const =0
- void **calculate** (OneAssetOption::results *result) const

Protected Attributes

- boost::shared_ptr< StandardStepCondition > **stepCondition_**
- Array **prices_**
- TridiagonalOperator **controlOperator_**
- std::vector< boost::shared_ptr< bc_type > > **controlBCs_**
- Array **controlPrices_**

7.213 FDVanillaEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdvanillaengine.hpp>`

Inheritance diagram for FDVanillaEngine:

7.213.1 Detailed Description

Finite-differences pricing engine for BSM one asset options.

The name is a misnomer as this is a base class for any finite difference scheme. It's main job is to handle grid layout.

Public Member Functions

- **FDVanillaEngine** (Size timeSteps, Size gridPoints, bool timeDependent=false)
- const Array & **grid** () const

Protected Types

- typedef BoundaryCondition< TridiagonalOperator > **bc_type**

Protected Member Functions

- virtual void **setupArguments** (const OneAssetOption::arguments *args) const
- virtual void **setGridLimits** () const
- virtual void **setGridLimits** (Real, Time) const
- virtual void **initializeGrid** () const
- virtual void **initializeInitialCondition** () const
- virtual void **initializeOperator** () const
- virtual Time **getResidualTime** () const

Protected Attributes

- Size **timeSteps_**
- Size **gridPoints_**
- bool **timeDependent_**
- boost::shared_ptr< BlackScholesProcess > **process_**
- Real **requiredGridValue_**
- Date **exerciseDate_**
- Array **grid_**
- boost::shared_ptr< Payoff > **payoff_**
- TridiagonalOperator **finiteDifferenceOperator_**
- Array **intrinsicValues_**
- std::vector< boost::shared_ptr< bc_type > > **BCs_**
- Real **sMin_**
- Real **center_**
- Real **sMax_**

7.214 FIMCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for FIMCurrency:

7.214.1 Detailed Description

Finnish markka.

The ISO three-letter code is FIM; the numeric code is 246. It is divided in 100 penniä.

7.215 FiniteDifferenceModel Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/finitedifferencemodel.hpp>`

7.215.1 Detailed Description

template<class Evolver> class QuantLib::FiniteDifferenceModel< Evolver >

Generic finite difference model.

Public Types

- typedef Evolver::traits **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **FiniteDifferenceModel** (const operator_type &L, const bc_set &bcs, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- const Evolver & **evolver** () const
- void rollback (array_type &a, Time from, Time to, Size steps)
- void rollback (array_type &a, Time from, Time to, Size steps, const condition_type &condition)

7.215.2 Member Function Documentation

7.215.2.1 void rollback (array_type & *a*, Time *from*, Time *to*, Size *steps*)

solves the problem between the given times.

Warning:
being this a rollback, `from` must be a later time than `to`.

7.215.2.2 void rollback (array_type & *a*, Time *from*, Time *to*, Size *steps*, const condition_type & *condition*)

solves the problem between the given times, applying a condition at every step.

Warning:
being this a rollback, `from` must be a later time than `to`.

7.216 FixedCouponBond Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/fixedcouponbond.hpp>`

Inheritance diagram for FixedCouponBond:

7.216.1 Detailed Description

fixed-coupon bond

Test
calculations are tested by checking results against cached values.

Public Member Functions

- **FixedCouponBond** (const Date &issueDate, const Date &datedDate, const Date &maturityDate, Integer settlementDays, const std::vector< Rate > &coupons, Frequency couponFrequency, const DayCounter &dayCounter, const Calendar &calendar, BusinessDayConvention convention=Following, Real redemption=100.0, const Handle< YieldTermStructure > &discountCurve=Handle< YieldTermStructure >(), const Date &stub=Date(), bool fromEnd=true, bool longFinal=false)

7.217 FixedCouponBondHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bondhelpers.hpp>`

Inheritance diagram for FixedCouponBondHelper:

7.217.1 Detailed Description

fixed-coupon bond helper

Warning:
This class assumes that the reference date does not change between calls of setTermStructure().

Public Member Functions

- **FixedCouponBondHelper** (const Handle< Quote > &cleanPrice, const Date &issueDate, const Date &datedDate, const Date &maturityDate, Integer settlementDays, const std::vector< Rate > &coupons, Frequency frequency, const DayCounter &dayCounter, const Calendar &calendar, BusinessDayConvention convention=Following, Real redemption=100.0, const Date &stub=Date(), bool fromEnd=true)
- Real **impliedQuote** () const
- Date latestDate () const

  latest relevant date

- void setTermStructure (YieldTermStructure *)

  sets the term structure to be used for pricing

Protected Attributes

- Date **issueDate_**
- Date **datedDate_**
- Date **maturityDate_**
- Integer **settlementDays_**
- std::vector< Rate > **coupons_**
- Frequency **frequency_**
- DayCounter **dayCounter_**
- Calendar **calendar_**
- BusinessDayConvention **businessDayConvention_**
- Real **redemption_**
- Date **stub_**
- bool **fromEnd_**
- Date **settlement_**
- Date **latestDate_**
- boost::shared_ptr< FixedCouponBond > **bond_**
- Handle< YieldTermStructure > **termStructureHandle_**

7.217.2 Member Function Documentation

7.217.2.1 Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements RateHelper.

7.217.2.2 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from RateHelper.

7.218 FixedRateCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/fixedratecoupon.hpp>`

Inheritance diagram for FixedRateCoupon:

7.218.1 Detailed Description

Coupon paying a fixed interest rate

Public Member Functions

- **FixedRateCoupon** (Real nominal, const Date &paymentDate, Rate rate, const DayCounter &dayCounter, const Date &startDate, const Date &endDate, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date())

CashFlow interface

- Real amount () const
  returns the amount of the cash flow

Coupon interface

- Rate rate () const
  accrued rate
- DayCounter dayCounter () const
  day counter for accrual calculation
- Real accruedAmount (const Date &) const
  accrued amount at the given date

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.218.2 Member Function Documentation

7.218.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements CashFlow.

7.219 FlatForward Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/flatforward.hpp>`

Inheritance diagram for FlatForward:

7.219.1 Detailed Description

Flat interest-rate curve.

Public Member Functions

- **FlatForward** (const Date &referenceDate, const Handle< Quote > &forward, const DayCounter &dayCounter, Compounding compounding=Continuous, Frequency frequency=Annual)
- **FlatForward** (const Date &referenceDate, Rate forward, const DayCounter &dayCounter, Compounding compounding=Continuous, Frequency frequency=Annual)
- **FlatForward** (Integer settlementDays, const Calendar &calendar, const Handle< Quote > &forward, const DayCounter &dayCounter, Compounding compounding=Continuous, Frequency frequency=Annual)
- **FlatForward** (Integer settlementDays, const Calendar &calendar, Rate forward, const DayCounter &dayCounter, Compounding compounding=Continuous, Frequency frequency=Annual)
- DayCounter dayCounter () const

  the day counter used for date/time conversion

- Compounding **compounding** () const
- Frequency **compoundingFrequency** () const
- Date maxDate () const

  the latest date for which the curve can return rates

- void update ()

7.219.2 Member Function Documentation

7.219.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from TermStructure.

7.220 FloatingRateBond Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/floatingratebond.hpp>`

Inheritance diagram for FloatingRateBond:

7.220.1 Detailed Description

floating-rate bond

Test
calculations are tested by checking results against cached values.

Public Member Functions

- **FloatingRateBond** (const Date &issueDate, const Date &datedDate, const Date &maturityDate, Integer settlementDays, const boost::shared_ptr< Xibor > &index, Integer fixingDays, const std::vector< Spread > &spreads, Frequency couponFrequency, const DayCounter &dayCounter, const Calendar &calendar, BusinessDayConvention convention=Following, Real redemption=100.0, const Handle< YieldTermStructure > &discountCurve=Handle< YieldTermStructure >(), const Date &stub=Date(), bool fromEnd=true)

7.221 FloatingRateCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/floatingratecoupon.hpp>`

Inheritance diagram for FloatingRateCoupon:

7.221.1 Detailed Description

Coupon paying a variable rate

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **FloatingRateCoupon** (Real nominal, const Date &paymentDate, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date())

Coupon interface

- Rate rate () const
  accrued rate
- Real accruedAmount (const Date &) const
  accrued amount at the given date

Inspectors

- Integer fixingDays () const
  fixing days
- virtual Spread spread () const
  spread paid over the fixing of the underlying index
- virtual Rate indexFixing () const =0
  fixing of the underlying index
- virtual Date fixingDate () const =0
  fixing date

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- virtual Rate convexityAdjustment (Rate fixing) const
  convexity adjustment for the given index fixing

Protected Attributes

- Integer **fixingDays_**
- Spread **spread_**

7.222 Floor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp>`

Inheritance diagram for Floor:

7.222.1 Detailed Description

Concrete floor class.

Public Member Functions

- **Floor** (const std::vector< boost::shared_ptr< CashFlow > > &floatingLeg, const std::vector< Rate > &exerciseRates, const Handle< YieldTermStructure > &termStructure, const boost::shared_ptr< PricingEngine > &engine)

7.223 FloorTruncation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for FloorTruncation:

7.223.1 Detailed Description

Floor truncation.

Public Member Functions

- **FloorTruncation** (Integer precision, Integer digit=5)

7.224 ForwardEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Forward/forwardengine.hpp>`

Inheritance diagram for ForwardEngine:

7.224.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::ForwardEngine< ArgumentsType, ResultsType >

Forward engine base class.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **ForwardEngine** (const boost::shared_ptr< GenericEngine< ArgumentsType, ResultsType > > &)
- void **setOriginalArguments** () const
- void **calculate** () const
- void **getOriginalResults** () const

Protected Attributes

- boost::shared_ptr< GenericEngine< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

7.225 ForwardFlat Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/forwardflatinterpolation.hpp>`

7.225.1 Detailed Description

Forward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

Public Member Functions

- template<class I1, class I2> Interpolation **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

7.226 ForwardFlatInterpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/forwardflatinterpolation.hpp>`

Inheritance diagram for ForwardFlatInterpolation:

7.226.1 Detailed Description

Forward-flat interpolation between discrete points.

Public Member Functions

- template<class I1, class I2> ForwardFlatInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.226.2 Constructor & Destructor Documentation

7.226.2.1 ForwardFlatInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:
the *x* values must be sorted.

7.227 ForwardOptionArguments Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Forward/forwardengine.hpp>`

7.227.1 Detailed Description

template<class ArgumentsType> class QuantLib::ForwardOptionArguments< ArgumentsType >

Arguments for forward (strike-resetting) option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Real **moneyness**
- Date **resetDate**

7.228 ForwardPerformanceEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Forward/forwardperformanceengine.hpp>`

Inheritance diagram for ForwardPerformanceEngine:

7.228.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::ForwardPerformanceEngine< ArgumentsType, ResultsType >

Forward performance engine.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **ForwardPerformanceEngine** (const boost::shared_ptr< GenericEngine< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

7.229 ForwardRate Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bootstraptraits.hpp>`

7.229.1 Detailed Description

Forward-curve traits.

Static Public Member Functions

- static Rate **initialValue** ()
- static Rate **initialGuess** ()
- static Rate **guess** (const YieldTermStructure *c, const Date &d)
- static Rate **minValueAfter** (Size, const std::vector< Real > &)
- static Rate **maxValueAfter** (Size, const std::vector< Real > &)
- static void **updateGuess** (std::vector< Rate > &data, Rate forward, Size i)

7.230 ForwardRateStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardstructure.hpp>`

Inheritance diagram for ForwardRateStructure:

7.230.1 Detailed Description

Forward rate term structure.

This abstract class acts as an adapter to TermStructure allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes. Zero yields and discounts are calculated from forwards.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- **ForwardRateStructure** ()
- **ForwardRateStructure** (const Date &referenceDate)
- **ForwardRateStructure** (Integer settlementDays, const Calendar &)

Protected Member Functions

YieldTermStructure implementation

- DiscountFactor discountImpl (Time) const
- virtual Rate forwardImpl (Time) const =0

  instantaneous forward-rate calculation

- virtual Rate zeroYieldImpl (Time) const

7.230.2 Member Function Documentation

7.230.2.1 DiscountFactor discountImpl (Time) const [protected, virtual]

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Implements YieldTermStructure.

Reimplemented in CompoundForward.

7.230.2.2 Rate zeroYieldImpl (Time) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:

This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented in CompoundForward, InterpolatedForwardCurve, and ForwardSpreaded-TermStructure.

7.231 ForwardSpreadedTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardspreadedtermstructure.hpp`

Inheritance diagram for ForwardSpreadedTermStructure:

7.231.1 Detailed Description

Term structure with added spread on the instantaneous forward rate.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Test

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Public Member Functions

- **ForwardSpreadedTermStructure** (const Handle< YieldTermStructure > &, const Handle< Quote > &spread)

YieldTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Calendar calendar () const
  the calendar used for reference date calculation
- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- Date maxDate () const
  the latest date for which the curve can return rates
- Time maxTime () const
  the latest time for which the curve can return rates

Protected Member Functions

- Rate forwardImpl (Time) const
  returns the spreaded forward rate
- Rate zeroYieldImpl (Time) const
  returns the spreaded zero yield rate

7.231.2 Member Function Documentation

7.231.2.1 Rate zeroYieldImpl (Time) const [protected, virtual]

returns the spreaded zero yield rate

Warning:
This method must disappear should the spread become a curve

Reimplemented from ForwardRateStructure.

7.232 ForwardVanillaOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/forwardvanillaoption.hpp>`

Inheritance diagram for ForwardVanillaOption:

7.232.1 Detailed Description

Forward version of a vanilla option.

Public Types

- typedef ForwardOptionArguments< VanillaOption::arguments > **arguments**
- typedef VanillaOption::results **results**
- typedef ForwardEngine< VanillaOption::arguments, VanillaOption::results > **engine**

Public Member Functions

- **ForwardVanillaOption** (Real moneyness, Date resetDate, const boost::shared_ptr< StochasticProcess > &stochProc, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine)
- void setupArguments (Arguments *) const

Protected Member Functions

- void performCalculations () const

7.232.2 Member Function Documentation

7.232.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

7.232.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from OneAssetStrikedOption.

7.233 FraRateHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp>`

Inheritance diagram for FraRateHelper:

7.233.1 Detailed Description

Forward rate agreement.

Warning:
This class assumes that the reference date does not change between calls of setTermStructure().

Todo
convexity adjustment should be implemented.

Public Member Functions

- **FraRateHelper** (const Handle< Quote > &rate, Integer monthsToStart, Integer monthsToEnd, Integer settlementDays, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- **FraRateHelper** (Rate rate, Integer monthsToStart, Integer monthsToEnd, Integer settlementDays, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void setTermStructure (YieldTermStructure *)

  sets the term structure to be used for pricing

- Date latestDate () const

  latest relevant date

7.233.2 Member Function Documentation

7.233.2.1 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:
Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from RateHelper.

7.233.2.2 Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements RateHelper.

7.234 FRFCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for FRFCurrency:

7.234.1 Detailed Description

French franc.

The ISO three-letter code is FRF; the numeric code is 250. It is divided in 100 centimes.

7.235 FuturesRateHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp>`

Inheritance diagram for FuturesRateHelper:

7.235.1 Detailed Description

Interest-rate futures.

Warning:
This class assumes that the reference date does not change between calls of setTermStructure().

Public Member Functions

- **FuturesRateHelper** (const Handle< Quote > &price, const Date &immDate, Integer nMonths, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- **FuturesRateHelper** (const Handle< Quote > &price, const Date &immDate, const Date &matDate, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- **FuturesRateHelper** (Real price, const Date &immDate, Integer nMonths, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- Date latestDate () const

  latest relevant date

7.235.2 Member Function Documentation

7.235.2.1 Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements RateHelper.

7.236 G2 Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/g2.hpp>`

Inheritance diagram for G2:

7.236.1 Detailed Description

Two-additive-factor gaussian model class.

This class implements a two-additive-factor model defined by

$$dr_t = \varphi(t) + x_t + y_t$$

where x_t and y_t are defined by

$$dx_t = -ax_t dt + \sigma dW_t^1, x_0 = 0$$

$$dy_t = -by_t dt + \sigma dW_t^2, y_0 = 0$$

and $dW_t^1 dW_t^2 = \rho dt$.

Bug

This class was not tested enough to guarantee its functionality.

Public Member Functions

- **G2** (const Handle< YieldTermStructure > &termStructure, Real a=0.1, Real sigma=0.01, Real b=0.1, Real eta=0.01, Real rho=-0.75)
- boost::shared_ptr< ShortRateDynamics > dynamics () const

  Returns the short-rate dynamics.

- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- Real **swaption** (const Swaption::arguments &arguments, Real range, Size intervals) const
- DiscountFactor discount (Time t) const

  Implied discount curve.

Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const
- Real **B** (Real x, Time t) const

Friends

- class **SwaptionPricingFunction**

Classes

- class FittingParameter

  Analytical term-structure fitting parameter $\varphi(t)$.

7.237 G2::FittingParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/g2.hpp>`

Inheritance diagram for G2::FittingParameter:

7.237.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2}(\frac{\sigma(1-e^{-at})}{a})^2 + \frac{1}{2}(\frac{\eta(1-e^{-bt})}{b})^2 + \rho\frac{\sigma(1-e^{-at})}{a}\frac{\eta(1-e^{-bt})}{b},$$

where $f(t)$ is the instantaneous forward rate at t .

Public Member Functions

- **FittingParameter** (const Handle< YieldTermStructure > &termStructure, Real a, Real sigma, Real b, Real eta, Real rho)

7.238 G2SwaptionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/g2swaptionengine.hpp>`

Inheritance diagram for G2SwaptionEngine:

7.238.1 Detailed Description

Swaption priced by means of the Black formula

Public Member Functions

- **G2SwaptionEngine** (const boost::shared_ptr< G2 > &mod, Real range, Size intervals)
- void **calculate** () const

7.239 GammaFunction Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gammadistribution.hpp>`

7.239.1 Detailed Description

Gamma function class.

This is a function defined by

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

The implementation of the algorithm was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

Test
the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- Real **logValue** (Real x) const

7.240 GapPayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for GapPayoff:

7.240.1 Detailed Description

Binary gap payoff.

Public Member Functions

- **GapPayoff** (Option::Type type, Real strike, Real strikePayoff)
- Real **operator()** (Real price) const
- Real **strikePayoff** () const

7.241 GaussChebyshev2thIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussChebyshev2thIntegration:

7.241.1 Detailed Description

Gauss-Chebyshev integration second kind.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

The weighting function is

$$w(x) = (1 - x^2)^{1/2}$$

Public Member Functions

- **GaussChebyshev2thIntegration** (Size n)

7.242 GaussChebyshevIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussChebyshevIntegration:

7.242.1 Detailed Description

Gauss-Chebyshev integration.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

The weighting function is

$$w(x) = (1 - x^2)^{-1/2}$$

Public Member Functions

- **GaussChebyshevIntegration** (Size n)

7.243 GaussGegenbauerIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussGegenbauerIntegration:

7.243.1 Detailed Description

Gauss-Gegenbauer integration.

This class performs a 1-dimensional Gauss-Gegenbauer integration.

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

The weighting function is

$$w(x) = (1-x^2)^{\lambda-1/2}$$

Public Member Functions

- **GaussGegenbauerIntegration** (Size n, Real lambda)

7.244 GaussHermiteIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussHermiteIntegration:

7.244.1 Detailed Description

generalized Gauss-Hermite integration

This class performs a 1-dimensional Gauss-Hermite integration.

$$\int_{-\inf}^{\inf} f(x)\mathrm{d}x$$

The weighting function is

$$w(x;\mu) = |x|^{2\mu} \exp -x * x$$

and

$$\mu > -0.5$$

Public Member Functions

- **GaussHermiteIntegration** (Size n, Real mu=0.0)

7.245 GaussHermitePolynomial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpolynomial.hpp>`

Inheritance diagram for GaussHermitePolynomial:

7.245.1 Detailed Description

Gauss-Hermite polynomial.

Public Member Functions

- **GaussHermitePolynomial** (Real mu=0.0)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

7.246 GaussHyperbolicIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussHyperbolicIntegration:

7.246.1 Detailed Description

Gauss-Hyperbolic integration.

This class performs a 1-dimensional Gauss-Hyperbolic integration.

$$\int_{-\inf}^{\inf} f(x)\mathrm{d}x$$

The weighting function is

$$w(x) = 1/cosh(x)$$

Public Member Functions

- **GaussHyperbolicIntegration** (Size n)

7.247 GaussHyperbolicPolynomial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpolynomial.hpp>`

Inheritance diagram for GaussHyperbolicPolynomial:

7.247.1 Detailed Description

Gauss hyperbolic polynomial.

Public Member Functions

- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

7.248 GaussianOrthogonalPolynomial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpolynomial.hpp>`

Inheritance diagram for GaussianOrthogonalPolynomial:

7.248.1 Detailed Description

orthogonal polynomial for Gaussian quadratures

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

The polynomials are defined by the three-term recurrence relation

$$P_{k+1}(x) = (x - \alpha_k)P_k(x) - \beta_k P_{k-1}(x)$$

and

$$\mu_0 = \int w(x)dx$$

Public Member Functions

- virtual Real **mu_0** () const =0
- virtual Real **alpha** (Size i) const =0
- virtual Real **beta** (Size i) const =0
- virtual Real **w** (Real x) const =0

7.249 GaussianQuadrature Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussianQuadrature:

7.249.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures method.

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

Test
the correctness of the result is tested by checking it against known good values.

Public Member Functions

- **GaussianQuadrature** (Size n, const GaussianOrthogonalPolynomial &p)
- template<class F> Real **operator()** (const F &f) const
- Size **order** () const

7.250 GaussianStatistics Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianstatistics.hpp>`

7.250.1 Detailed Description

template<class Stat> class QuantLib::GaussianStatistics< Stat >

Statistics tool for gaussian-assumption risk measures.

It can calculate gaussian assumption risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the template class

Public Member Functions

- **GaussianStatistics** (const Stat &s)

Gaussian risk measures

- Real gaussianDownsideVariance () const
- Real gaussianDownsideDeviation () const
- Real gaussianRegret (Real target) const
- Real gaussianPercentile (Real percentile) const
- Real gaussianTopPercentile (Real percentile) const
- Real gaussianPotentialUpside (Real percentile) const
  gaussian-assumption Potential-Upside at a given percentile
- Real gaussianValueAtRisk (Real percentile) const
  gaussian-assumption Value-At-Risk at a given percentile
- Real gaussianExpectedShortfall (Real percentile) const
  gaussian-assumption Expected Shortfall at a given percentile
- Real gaussianShortfall (Real target) const
  gaussian-assumption Shortfall (observations below target)
- Real gaussianAverageShortfall (Real target) const
  gaussian-assumption Average Shortfall (averaged shortfallness)

7.250.2 Member Function Documentation

7.250.2.1 Real gaussianDownsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^{N} \theta \times x_i^2}{\sum_{i=1}^{N} w_i}$$

, where $\theta = 0$ if x > 0 and θ =1 if x <0

7.250.2.2 Real gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.250.2.3 Real gaussianRegret (Real *target*) const

returns the variance of observations below target

$$\frac{\sum w_i(min(0, x_i - target))^2}{\sum w_i}.$$

See Dembo, Freeman "The Rules Of Risk", Wiley (2001)

7.250.2.4 Real gaussianPercentile (Real *percentile*) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-u^2/2)du$$

7.250.2.5 Real gaussianTopPercentile (Real *percentile*) const

Precondition:
percentile must be in range (0-100%) extremes excluded

7.250.2.6 Real gaussianPotentialUpside (Real *percentile*) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:
percentile must be in range [90-100%)

7.250.2.7 Real gaussianValueAtRisk (Real *percentile*) const

gaussian-assumption Value-At-Risk at a given percentile

Precondition:
percentile must be in range [90-100%)

7.250.2.8 Real gaussianExpectedShortfall (Real *percentile*) const

gaussian-assumption Expected Shortfall at a given percentile

Assuming a gaussian distribution it returns the expected loss in case that the loss exceeded a VaR threshold,

$$\mathrm{E}\left[x \mid x < \mathrm{VaR}(p)\right],$$

that is the average of observations below the given percentile p . Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

7.251 GaussJacobiIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussJacobiIntegration:

7.251.1 Detailed Description

Gauss-Jacobi integration.

This class performs a 1-dimensional Gauss-Jacobi integration.

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

The weighting function is

$$w(x;\alpha,\beta) = (1-x)^{\alpha}(1+x)^{\beta}$$

Public Member Functions

- **GaussJacobiIntegration** (Size n, Real alpha, Real beta)

7.252 GaussJacobiPolynomial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpolynomial.hpp>`

Inheritance diagram for GaussJacobiPolynomial:

7.252.1 Detailed Description

Gauss-Jacobi polynomial.

Public Member Functions

- **GaussJacobiPolynomial** (Real alpha, Real beta)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

7.253 GaussLaguerreIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussLaguerreIntegration:

7.253.1 Detailed Description

generalized Gauss-Laguerre integration

This class performs a 1-dimensional Gauss-Laguerre integration.

$$\int_0^{\text{inf}} f(x)\mathrm{d}x$$

The weighting function is

$$w(x;s) = x^s \exp -x$$

and

$$s > -1$$

Public Member Functions

- **GaussLaguerreIntegration** (Size n, Real s=0.0)

7.254 GaussLaguerrePolynomial Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpolynomial.hpp>`

Inheritance diagram for GaussLaguerrePolynomial:

7.254.1 Detailed Description

Gauss-Laguerre polynomial.

Public Member Functions

- **GaussLaguerrePolynomial** (Real s=0.0)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

7.255 GaussLegendreIntegration Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

Inheritance diagram for GaussLegendreIntegration:

7.255.1 Detailed Description

Gauss-Legendre integration.

This class performs a 1-dimensional Gauss-Legendre integration.

$$\int_{-1}^{1} f(x)\mathrm{d}x$$

The weighting function is

$$w(x) = 1$$

Public Member Functions

- **GaussLegendreIntegration** (Size n)

7.256 GBPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for GBPCurrency:

7.256.1 Detailed Description

British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

7.257 GBPLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/gbplibor.hpp>`

Inheritance diagram for GBPLibor:

7.257.1 Detailed Description

GBP LIBOR rate

Pound Sterling LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Public Member Functions

- **GBPLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual365Fixed())

7.258 GeneralStatistics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/generalstatistics.hpp>`

7.258.1 Detailed Description

Statistics tool.

This class accumulates a set of data and returns their statistics (e.g: mean, variance, skewness, kurtosis, error estimation, percentile, etc.) based on the empirical distribution (no gaussian assumption)

It doesn't suffer the numerical instability problem of IncrementalStatistics. The downside is that it stores all samples, thus increasing the memory requirements.

Public Member Functions

Inspectors

- Size samples () const
  number of samples collected
- const std::vector< std::pair< Real, Real > > & data () const
  collected data
- Real weightSum () const
  sum of data weights
- Real mean () const
- Real variance () const
- Real standardDeviation () const
- Real errorEstimate () const
- Real skewness () const
- Real kurtosis () const
- Real min () const
- Real max () const
- template<class Func, class Predicate> std::pair< Real, Size > expectationValue (const Func &f, const Predicate &inRange) const
- Real percentile (Real y) const
- Real topPercentile (Real y) const

Modifiers

- void add (Real value, Real weight=1.0)
  adds a datum to the set, possibly with a weight
- template<class DataIterator> void addSequence (DataIterator begin, DataIterator end)
  adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)
  adds a sequence of data to the set, each with its weight
- void reset ()

resets the data to a null set

- void sort () const
  sort the data set in increasing order

7.258.2 Member Function Documentation

7.258.2.1 Real mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.258.2.2 Real variance () const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \left\langle (x - \langle x \rangle)^2 \right\rangle.$$

7.258.2.3 Real standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

7.258.2.4 Real errorEstimate () const

returns the error estimate on the mean value, defined as $\epsilon = \sigma / \sqrt{N}$.

7.258.2.5 Real skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\left\langle (x - \langle x \rangle)^3 \right\rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

7.258.2.6 Real kurtosis () const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\left\langle (x - \langle x \rangle)^4 \right\rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

7.258.2.7 Real min () const

returns the minimum sample value

7.258.2.8 Real max () const

returns the maximum sample value

7.258.2.9 std::pair<Real,Size> expectationValue (const Func & *f*, const Predicate & *inRange*) const

Expectation value of a function f on a given range $\mathcal{R}$, i.e.,

$$\mathrm{E}\left[f \mid \mathcal{R}\right] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

The range is passed as a boolean function returning `true` if the argument belongs to the range or `false` otherwise.

The function returns a pair made of the result and the number of observations in the given range.

7.258.2.10 Real percentile (Real *y*) const

y -th percentile, defined as the value $\bar{x}$ such that

$$y = \frac{\sum_{x_i < \bar{x}} w_i}{\sum_i w_i}$$

Precondition:
y must be in the range $(0 - 1]$.

7.258.2.11 Real topPercentile (Real *y*) const

y -th top percentile, defined as the value $\bar{x}$ such that

$$y = \frac{\sum_{x_i > \bar{x}} w_i}{\sum_i w_i}$$

Precondition:
y must be in the range $(0 - 1]$.

7.258.2.12 void add (Real *value*, Real *weight* = 1.0)

adds a datum to the set, possibly with a weight

Precondition:
weights must be positive or null

7.259 GenericEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/pricingengine.hpp>`

Inheritance diagram for GenericEngine:

7.259.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::GenericEngine< ArgumentsType, ResultsType >

template base class for option pricing engines

Derived engines only need to implement the `calculate()` method.

Public Member Functions

- Arguments * **arguments** () const
- const Results * **results** () const
- void **reset** () const

Protected Attributes

- ArgumentsType **arguments_**
- ResultsType **results_**

7.260 GenericModelEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/genericmodelengine.hpp>`

Inheritance diagram for GenericModelEngine:

7.260.1 Detailed Description

template<class ModelType, class ArgumentsType, class ResultsType> class QuantLib::GenericModelEngine< ModelType, ArgumentsType, ResultsType >

Base class for some pricing engine on a particular model.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **GenericModelEngine** (const boost::shared_ptr< ModelType > &model)
- void **setModel** (const boost::shared_ptr< ModelType > &model)
- virtual void update ()

Protected Attributes

- boost::shared_ptr< ModelType > **model_**

7.260.2 Member Function Documentation

7.260.2.1 virtual void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

Reimplemented in LatticeShortRateModelEngine, LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >, and LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >.

7.261 GenericRiskStatistics Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/riskstatistics.hpp>`

7.261.1 Detailed Description

template<class S> class QuantLib::GenericRiskStatistics< S >

empirical-distribution risk measures

This class wraps a somewhat generic statistic tool and adds a number of risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the data distribution as reported by the underlying tool.

Todo

add historical annualized volatility

Public Member Functions

- Real semiVariance () const
- Real semiDeviation () const
- Real downsideVariance () const
- Real downsideDeviation () const
- Real regret (Real target) const
- Real potentialUpside (Real percentile) const

  potential upside (the reciprocal of VAR) at a given percentile

- Real valueAtRisk (Real percentile) const

  value-at-risk at a given percentile

- Real expectedShortfall (Real percentile) const

  expected shortfall at a given percentile

- Real shortfall (Real target) const
- Real averageShortfall (Real target) const

7.261.2 Member Function Documentation

7.261.2.1 Real semiVariance () const

returns the variance of observations below the mean,

$$\frac{N}{N-1}\mathrm{E}\left[(x-\langle x\rangle)^2 \mid x<\langle x\rangle\right].$$

See Markowitz (1959).

7.261.2.2 Real semiDeviation () const

returns the semi deviation, defined as the square root of the semi variance.

7.261.2.3 Real downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1}\mathrm{E}\left[x^2 \mid x < 0\right].$$

7.261.2.4 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.261.2.5 Real regret (Real *target*) const

returns the variance of observations below target,

$$\frac{N}{N-1}\mathrm{E}\left[(x-t)^2 \mid x < t\right].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

7.261.2.6 Real potentialUpside (Real *centile*) const

potential upside (the reciprocal of VAR) at a given percentile

Precondition:
percentile must be in range [90-100%)

7.261.2.7 Real valueAtRisk (Real *centile*) const

value-at-risk at a given percentile

Precondition:
percentile must be in range [90-100%)

7.261.2.8 Real expectedShortfall (Real *percentile*) const

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$\mathrm{E}\left[x \mid x < \mathrm{VaR}(p)\right],$$

that is the average of observations below the given percentile p . Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

7.261.2.9 Real shortfall (Real *target*) const

probability of missing the given target, defined as

$$\mathrm{E}\left[\Theta \mid (-\infty, \infty)\right]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

7.261.2.10 Real averageShortfall (Real *target*) const

averaged shortfallness, defined as

$$\mathrm{E}\left[t - x \mid x < t\right]$$

7.262 GeometricBrownianMotionProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/geometricbrownianprocess.hpp>`

Inheritance diagram for GeometricBrownianMotionProcess:

7.262.1 Detailed Description

Geometric brownian-motion process.

This class describes the stochastic process governed by

$$dS(t,S) = \mu S dt + \sigma S dW_t.$$

Public Member Functions

- **GeometricBrownianMotionProcess** (double initialValue, double mue, double sigma)
- Real x0 () const

  returns the initial value of the state variable

- Real drift (Time t, Real x) const

  returns the drift part of the equation, i.e. $\mu(t, x_t)$

- Real diffusion (Time t, Real x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

Protected Attributes

- double **initialValue_**
- double **mue_**
- double **sigma_**

7.263 Germany Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/germany.hpp>`

Inheritance diagram for Germany:

7.263.1 Detailed Description

German calendars.

Public holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Ascension Thursday
- Whit Monday
- Corpus Christi
- Labour Day, May 1st
- National Day, October 3rd
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31st

Holidays for the Frankfurt Stock exchange (data from `http://deutsche-boerse.com/`):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Xetra exchange (data from http://deutsche-boerse.com/):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Eurex exchange (data from http://www.eurexchange.com/index.html):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Test
the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum Market { Settlement, FrankfurtStockExchange, Xetra, Eurex }
  German calendars.

Public Member Functions

- **Germany** (Market market=FrankfurtStockExchange)

7.263.2 Member Enumeration Documentation

7.263.2.1 enum Market

German calendars.

Enumerator:

Settlement generic settlement calendar

FrankfurtStockExchange Frankfurt stock-exchange.

Xetra Xetra.

Eurex Eurex.

7.264 GRDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for GRDCurrency:

7.264.1 Detailed Description

Greek drachma.

The ISO three-letter code is GRD; the numeric code is 300. It is divided in 100 lepta.

7.265 Greeks Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/option.hpp>`

Inheritance diagram for Greeks:

7.265.1 Detailed Description

additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- Real **delta**
- Real **gamma**
- Real **theta**
- Real **vega**
- Real **rho**
- Real **dividendRho**

7.266 HaltonRsg Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/haltonrsg.hpp>`

7.266.1 Detailed Description

Halton low-discrepancy sequence generator.

Halton algorithm for low-discrepancy sequence. For more details see chapter 8, paragraph 2 of "Monte Carlo Methods in Finance", by Peter Jäckel

Test

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- typedef Sample< Array > **sample_type**

Public Member Functions

- **HaltonRsg** (Size dimensionality, unsigned long seed=0, bool randomStart=true, bool randomShift=false)
- const sample_type & **nextSequence** () const
- const sample_type & **lastSequence** () const
- Size **dimension** () const

7.267 Handle Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/handle.hpp>`

7.267.1 Detailed Description

template<class Type> class QuantLib::Handle< Type >

Globally accessible relinkable pointer.

An instance of this class can be relinked to another shared pointer: such change will be propagated to all the copies of the instance.

Precondition:
Class "Type" must inherit from Observable

Public Member Functions

- Handle (const boost::shared_ptr< Type > &h=boost::shared_ptr< Type >(), bool registerAsObserver=true)
- void linkTo (const boost::shared_ptr< Type > &, bool registerAsObserver=true)
- const boost::shared_ptr< Type > & currentLink () const

  dereferencing

- const boost::shared_ptr< Type > & **operator** → () const
- bool empty () const

  Checks if the contained shared pointer points to anything.

7.267.2 Constructor & Destructor Documentation

7.267.2.1 Handle (const boost::shared_ptr< Type > & *h* = `boost::shared_ptr< Type >()`, bool *registerAsObserver* = `true`) `[explicit]`

Warning:
see the documentation of the Link class for issues relatives to `registerAsObserver`.

7.267.3 Member Function Documentation

7.267.3.1 void linkTo (const boost::shared_ptr< Type > &, bool *registerAsObserver* = `true`)

Warning:
see the documentation of the Link class for issues relatives to `registerAsObserver`.

7.268 Helsinki Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/helsinki.hpp>`

Inheritance diagram for Helsinki:

7.268.1 Detailed Description

Helsinki calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension Thursday
- Labour Day, May 1st
- Midsummer Eve (Friday between June 18-24)
- Independence Day, December 6th
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th

7.269 HestonModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>`

Inheritance diagram for HestonModel:

7.269.1 Detailed Description

Heston model for the stochastic volatility of an asset.

References:

Heston, Steven L., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The review of Financial Studies, Volume 6, Issue 2, 327-343.

Test
calibration is tested against known good values.

Public Member Functions

- **HestonModel** (const boost::shared_ptr< HestonProcess > &process)
- Real **theta** () const
- Real **kappa** () const
- Real **sigma** () const
- Real **rho** () const
- Real **v0** () const
- boost::shared_ptr< NumericalMethod > **tree** (const TimeGrid &) const

7.270 HestonModelHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp>`

Inheritance diagram for HestonModelHelper:

7.270.1 Detailed Description

calibration helper for Heston model

Public Member Functions

- **HestonModelHelper** (const Period &maturity, const Calendar &calendar, const Real s0, const Real strikePrice, const Handle< Quote > &volatility, const Handle< YieldTermStructure > &riskFreeRate, const Handle< YieldTermStructure > &dividendYield, bool calibrateVolatility=false)
- void **addTimesTo** (std::list< Time > &) const
- Real modelValue () const

  returns the price of the instrument according to the model

- Real **blackPrice** (Real volatility) const
- Real calibrationError ()

  returns the error resulting from the model valuation

- Time **maturity** () const

7.271 HestonProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/hestonprocess.hpp>`

Inheritance diagram for HestonProcess:

7.271.1 Detailed Description

Square-root stochastic-volatility Heston process.

This class describes the square root stochastic volatility process governed by

$$\begin{aligned} dS(t,S) &= \mu S dt + \sqrt{v} S dW_1 \\ dv(t,S) &= \kappa(\theta - v)dt + \sigma\sqrt{v}dW_2 \\ dW_1 dW_2 &= \rho dt \end{aligned}$$

Public Member Functions

- **HestonProcess** (const Handle< YieldTermStructure > &riskFreeRate, const Handle< YieldTermStructure > &dividendYield, const Handle< Quote > &s0, Real v0, Real kappa, Real theta, Real sigma, Real rho)
- Size size () const

  returns the number of dimensions of the stochastic process

- Disposable< Array > initialValues () const

  returns the initial values of the state variables

- Disposable< Array > drift (Time t, const Array &x) const

  returns the drift part of the equation, i.e., $\mu(t, x_t)$

- Disposable< Matrix > diffusion (Time t, const Array &x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

- Disposable< Array > apply (const Array &x0, const Array &dx) const
- Real **s0** () const
- Real **v0** () const
- Real **rho** () const
- Real **kappa** () const
- Real **theta** () const
- Real **sigma** () const
- const boost::shared_ptr< YieldTermStructure > & **dividendYield** () const
- const boost::shared_ptr< YieldTermStructure > & **riskFreeRate** () const
- Time time (const Date &) const

7.271.2 Member Function Documentation

7.271.2.1 Disposable<Array> apply (const Array & *x0*, const Array & *dx*) const [virtual]

applies a change to the asset value. By default, it returns x + Δx .

Reimplemented from StochasticProcess.

7.271.2.2 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from StochasticProcess.

7.272 History Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/history.hpp>`

7.272.1 Detailed Description

Container for historical data.

This class acts as a generic repository for a set of historical data. Single data can be accessed through their date, while sets of consecutive data can be accessed through iterators.

A history can contain null data, which can either be returned or skipped according to the chosen iterator type.

Example: uses of history iterators

Public Types

- typedef boost::filter_iterator< DataValidator, const_iterator > const_valid_iterator
  bidirectional iterator on non-null history entries
- typedef std::vector< Real >::const_iterator const_data_iterator
  random access iterator on historical data
- typedef boost::filter_iterator< DataValidator, const_data_iterator > const_valid_data_iterator
  bidirectional iterator on non-null historical data

Public Member Functions

- History ()
- template<class Iterator> History (const Date &firstDate, const Date &lastDate, Iterator begin, Iterator end)
- **History** (const Date &firstDate, const std::vector< Real > &values)
- History (const Date &firstDate, const Date &lastDate, const std::vector< Real > &values)
- History (const std::vector< Date > &dates, const std::vector< Real > &values)

Inspectors

- const Date & firstDate () const
  returns the first date for which a historical datum exists
- const Date & lastDate () const
  returns the last date for which a historical datum exists
- Size size () const
  returns the number of historical data including null ones

Historical data access

- Real operator[ ] (const Date &) const

  returns the (possibly null) datum corresponding to the given date

Iterator access

Four different types of iterators are provided, namely, const_iterator, const_valid_iterator, const_data_-iterator, and const_valid_data_iterator.

const_iterator and const_valid_iterator point to an Entry structure, the difference being that the latter only iterates over valid entries - i.e., entries whose data are not null. The same difference exists between const_data_iterator and const_valid_data_iterator which point directly to historical values without reference to the date they are associated to.

- const_iterator **begin** () const
- const_iterator **end** () const
- const_iterator **iterator** (const Date &d) const
- const_valid_iterator **vbegin** () const
- const_valid_iterator **vend** () const
- const_valid_iterator **valid_iterator** (const Date &d) const
- const_data_iterator **dbegin** () const
- const_data_iterator **dend** () const
- const_data_iterator **data_iterator** (const Date &d) const
- const_valid_data_iterator **vdbegin** () const
- const_valid_data_iterator **vdend** () const
- const_valid_data_iterator **valid_data_iterator** (const Date &d) const

Classes

- class const_iterator

  random access iterator on history entries

- class Entry

  single datum in history

7.272.2 Constructor & Destructor Documentation

7.272.2.1 History ()

Default constructor

7.272.2.2 History (const Date & *firstDate*, const Date & *lastDate*, Iterator *begin*, Iterator *end*)

This constructor initializes the history with the given set of values, corresponding to the date range between ***firstDate*** and ***lastDate*** included.

Precondition:

begin-end must equal the number of days from ***firstDate*** to ***lastDate*** included.

7.272.2.3 History (const Date & *firstDate*, const Date & *lastDate*, const std::vector< Real > & *values*)

This constructor initializes the history with the given set of values, corresponding to the date range between ***firstDate*** and ***lastDate*** included.

Precondition:

The size of ***values*** must equal the number of days from ***firstDate*** to ***lastDate*** included.

7.272.2.4 History (const std::vector< Date > & *dates*, const std::vector< Real > & *values*)

This constructor initializes the history with the given set of values, corresponding each to the element with the same index in the given set of dates. The whole date range between ***dates***[0] and ***dates***[N-1] will be automatically filled by inserting null values where a date is missing from the given set.

Precondition:

dates must be sorted.

There can be no pairs (***dates***[i],***values***[i]) and (***dates***[j],***values***[j]) such that `dates[i] == dates[j] && values[i] != values[j]`. Pairs with `dates[i] == dates[j] && values[i] == values[j]` are allowed; the duplicated entries will be discarded.

The size of ***values*** must equal the number of days from ***firstDate*** to ***lastDate*** included.

7.273 History::const_iterator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/history.hpp>`

7.273.1 Detailed Description

random access iterator on history entries

Public Member Functions

- const Entry & **dereference** () const
- bool **equal** (const const_iterator &i) const
- void **increment** ()
- void **decrement** ()
- void **advance** (BigInteger n)
- BigInteger **distance_to** (const const_iterator &i) const

Friends

- class **History**

7.274 History::Entry Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/history.hpp>`

7.274.1 Detailed Description

single datum in history

Public Member Functions

- const Date & **date** () const
- Real **value** () const

Friends

- class **const_iterator**

7.275 HKDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for HKDCurrency:

7.275.1 Detailed Description

Honk Kong dollar.

The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.

7.276 HongKong Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/hongkong.hpp>`

Inheritance diagram for HongKong:

7.276.1 Detailed Description

Hong Kong calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Ching Ming Festival, April 5th
- Good Friday
- Easter Monday
- Labor Day, May 1st
- SAR Establishment Day, July 1st
- National Day, October 1st
- Christmas, December 25th
- Boxing Day, December 26th
- Christmas Holiday, December 27th

Other holidays for which no rule is given (data available for 2004/2005 only:)

- Lunar New Year
- Chinese New Year
- Buddha's birthday
- Tuen NG Festival
- Mid-autumn fest
- Chung Yeung fest

Data from http://www.hkex.com.hk

7.277 HUFCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for HUFCurrency:

7.277.1 Detailed Description

Hungarian forint.

The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

7.278 HullWhite Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/hullwhite.hpp>`

Inheritance diagram for HullWhite:

7.278.1 Detailed Description

Single-factor Hull-White (extended Vasicek) model class.

This class implements the standard single-factor Hull-White model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t$$

where α and σ are constants.

Test
calibration results are tested against cached values

Bug
When the term structure is relinked, the r0 parameter of the underlying Vasicek model is not updated.

Public Member Functions

- **HullWhite** (const Handle< YieldTermStructure > &termStructure, Real a=0.1, Real sigma=0.01)
- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Return by default a trinomial recombining tree.

- boost::shared_ptr< ShortRateDynamics > dynamics () const

  returns the short-rate dynamics

- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const

Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const

Classes

- class Dynamics

  Short-rate dynamics in the Hull-White model.

- class FittingParameter

  Analytical term-structure fitting parameter $\varphi(t)$.

7.279 HullWhite::Dynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/hullwhite.hpp>`

7.279.1 Detailed Description

Short-rate dynamics in the Hull-White model.

The short-rate is here

$$r_t = \varphi(t) + x_t$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

Public Member Functions

- **Dynamics** (const Parameter &fitting, Real a, Real sigma)
- Real **variable** (Time t, Rate r) const
- Real **shortRate** (Time t, Real x) const

7.280 HullWhite::FittingParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/hullwhite.hpp>`

Inheritance diagram for HullWhite::FittingParameter:

7.280.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2}[\frac{\sigma(1 - e^{-at})}{a}]^2,$$

where $f(t)$ is the instantaneous forward rate at t .

Public Member Functions

- **FittingParameter** (const Handle< YieldTermStructure > &termStructure, Real a, Real sigma)

7.281 IEPCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for IEPCurrency:

7.281.1 Detailed Description

Irish punt.

The ISO three-letter code is IEP; the numeric code is 372. It is divided in 100 pence.

7.282 ILSCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for ILSCurrency:

7.282.1 Detailed Description

Israeli shekel.

The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

7.283 IMM Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/date.hpp>`

7.283.1 Detailed Description

Main cycle of the International Money Market (a.k.a. IMM) Months.

Public Types

- enum **Month** { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }

7.284 ImplicitEuler Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/impliciteuler.hpp>`

Inheritance diagram for ImplicitEuler:

7.284.1 Detailed Description

template<class Operator> class QuantLib::ImplicitEuler< Operator >

Backward Euler scheme for finite difference methods.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **ImplicitEuler** (const operator_type &L, const bc_set &bcs)

7.285 ImpliedTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/impliedtermstructure.hpp>`

Inheritance diagram for ImpliedTermStructure:

7.285.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Test

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure is checked.

Public Member Functions

- **ImpliedTermStructure** (const Handle< YieldTermStructure > &, const Date &referenceDate)

YieldTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Calendar calendar () const
  the calendar used for reference date calculation
- Date maxDate () const
  the latest date for which the curve can return rates

Protected Member Functions

- DiscountFactor discountImpl (Time) const
  returns the discount factor as seen from the evaluation date

7.286 ImpliedVolTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/impliedvoltermstructure.hpp>`

Inheritance diagram for ImpliedVolTermStructure:

7.286.1 Detailed Description

Implied vol term structure at a given date in the future.

The given date will be the implied reference date.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Warning:
It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only

Public Member Functions

- **ImpliedVolTermStructure** (const Handle< BlackVolTermStructure > &originalTS, const Date &referenceDate)

BlackVolTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- virtual Real blackVarianceImpl (Time t, Real strike) const
  Black variance calculation.

7.287 InArrearIndexedCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/inarrearindexedcoupon.hpp>`

Inheritance diagram for InArrearIndexedCoupon:

7.287.1 Detailed Description

In-arrear floating-rate coupon.

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Test
The class is tested by comparing the value of an in-arrear swap against a known good value.

Public Member Functions

- **InArrearIndexedCoupon** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Xibor > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())

FloatingRateCoupon interface

- Date fixingDate () const
  fixing date

Modifiers

- void **setCapletVolatility** (const Handle< CapletVolatilityStructure > &)

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- Rate convexityAdjustment (Rate fixing) const
  convexity adjustment for the given index fixing

Protected Attributes

- boost::shared_ptr< Xibor > **xibor_**
- Handle< CapletVolatilityStructure > **capletVolatility_**

7.288 IncrementalStatistics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/incrementalstatistics.hpp>`

7.288.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

Warning:
high moments are numerically unstable for high average/standardDeviation ratios

Public Member Functions

Inspectors

- Size samples () const
  number of samples collected
- Real weightSum () const
  sum of data weights
- Real mean () const
- Real variance () const
- Real standardDeviation () const
- Real downsideVariance () const
- Real downsideDeviation () const
- Real errorEstimate () const
- Real skewness () const
- Real kurtosis () const
- Real min () const
- Real max () const

Modifiers

- void add (Real value, Real weight=1.0)
  adds a datum to the set, possibly with a weight
- template<class DataIterator> void addSequence (DataIterator begin, DataIterator end)
  adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)
  adds a sequence of data to the set, each with its weight
- void reset ()
  resets the data to a null set

Protected Attributes

- Size **sampleNumber_**
- Size **downsideSampleNumber_**
- Real **sampleWeight_**
- Real **downsideSampleWeight_**
- Real **sum_**
- Real **quadraticSum_**
- Real **downsideQuadraticSum_**
- Real **cubicSum_**
- Real **fourthPowerSum_**
- Real **min_**
- Real **max_**

7.288.2 Member Function Documentation

7.288.2.1 Real mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.288.2.2 Real variance () const

returns the variance, defined as

$$\frac{N}{N-1}\left\langle (x - \langle x \rangle)^2 \right\rangle.$$

7.288.2.3 Real standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

7.288.2.4 Real downsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^{N} \theta \times x_i^2}{\sum_{i=1}^{N} w_i}$$

, where θ = 0 if x > 0 and θ =1 if x <0

7.288.2.5 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.288.2.6 Real errorEstimate () const

returns the error estimate ϵ , defined as the square root of the ratio of the variance to the number of samples.

7.288.2.7 Real skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)}\frac{\left\langle (x-\langle x\rangle)^3\right\rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

7.288.2.8 Real kurtosis () const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)}\frac{\left\langle (x-\langle x\rangle)^4\right\rangle}{\sigma^4}-\frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

7.288.2.9 Real min () const

returns the minimum sample value

7.288.2.10 Real max () const

returns the maximum sample value

7.288.2.11 void add (Real *value*, Real *weight* = 1.0)

adds a datum to the set, possibly with a weight

Precondition:
weight must be positive or null

7.288.2.12 void addSequence (DataIterator *begin*, DataIterator *end*, WeightIterator *wbegin*)

adds a sequence of data to the set, each with its weight

Precondition:
weights must be positive or null

7.289 Index Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/index.hpp>`

Inheritance diagram for Index:

7.289.1 Detailed Description

purely virtual base class for indexes

Public Member Functions

- virtual std::string name () const =0
  Returns the name of the index.
- virtual Rate fixing (const Date &fixingDate) const =0
  returns the fixing at the given date

7.289.2 Member Function Documentation

7.289.2.1 virtual std::string name () const [pure virtual]

Returns the name of the index.

Warning:
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implemented in Xibor.

7.289.2.2 virtual Rate fixing (const Date & *fixingDate*) const [pure virtual]

returns the fixing at the given date

Note:
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implemented in Xibor.

7.290 IndexedCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/indexedcoupon.hpp>`

Inheritance diagram for IndexedCoupon:

7.290.1 Detailed Description

Base indexed coupon class.

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **IndexedCoupon** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Index > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())

CashFlow interface

- Real amount () const
  returns the amount of the cash flow

Coupon interface

- DayCounter dayCounter () const
  day counter for accrual calculation

FloatingRateCoupon interface

- Rate indexFixing () const
  fixing of the underlying index

Inspectors

- const boost::shared_ptr< Index > & **index** () const

Observer interface

- void update ()

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.290.2 Member Function Documentation

7.290.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements CashFlow.

7.290.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.291 IndexManager Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/indexmanager.hpp>`

Inheritance diagram for IndexManager:

7.291.1 Detailed Description

global repository for past index fixings

Public Member Functions

- void **setHistory** (const std::string &name, const History &)
- const History & **getHistory** (const std::string &name) const
- bool **hasHistory** (const std::string &name) const
- std::vector< std::string > **histories** () const

Friends

- class **Singleton**< **IndexManager** >

7.292 INRCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for INRCurrency:

7.292.1 Detailed Description

Indian rupee.

The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.

7.293 Instrument Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/instrument.hpp>`

Inheritance diagram for Instrument:

7.293.1 Detailed Description

Abstract instrument class.

This class is purely abstract and defines the interface of concrete instruments which will be derived from this one.

Test
observability of class instances is checked.

Public Member Functions

- virtual void setupArguments (Arguments *) const

Inspectors

- Real NPV () const
  returns the net present value of the instrument.
- Real errorEstimate () const
  returns the error estimate on the NPV when available.
- virtual bool isExpired () const =0
  returns whether the instrument is still tradable.

Modifiers

- void setPricingEngine (const boost::shared_ptr< PricingEngine > &)
  set the pricing engine to be used.

Protected Member Functions

Calculations

- void calculate () const
- virtual void setupExpired () const
- virtual void performCalculations () const

Protected Attributes

- boost::shared_ptr< PricingEngine > **engine_**

Results

The value of this attribute and any other that derived classes might declare must be set during calculation.

- Real **NPV_**
- Real **errorEstimate_**

7.293.2 Member Function Documentation

7.293.2.1 void setPricingEngine (const boost::shared_ptr< PricingEngine > &)

set the pricing engine to be used.

Warning:
calling this method will have no effects in case the **performCalculation** method was overridden in a derived class.

7.293.2.2 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented in ContinuousAveragingAsianOption, DiscreteAveragingAsianOption, BarrierOption, BasketOption, CapFloor, CliquetOption, DividendVanillaOption, ForwardVanillaOption, MultiAssetOption, OneAssetOption, OneAssetStrikedOption, QuantoForwardVanillaOption, QuantoVanillaOption, SimpleSwap, and Swaption.

7.293.2.3 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:
Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.
Should this method be redefined in derived classes, LazyObject::calculate() should be called in the overriding method.

Reimplemented from LazyObject.

7.293.2.4 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented in MultiAssetOption, OneAssetOption, QuantoVanillaOption, and Swap.

7.293.2.5 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Implements LazyObject.

Reimplemented in BarrierOption, Bond, ForwardVanillaOption, MultiAssetOption, OneAssetOption, OneAssetStrikedOption, QuantoVanillaOption, Stock, and Swap.

7.294 IntegralEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/integralengine.hpp>`

Inheritance diagram for IntegralEngine:

7.294.1 Detailed Description

Pricing engine for European vanilla options using integral approach

Todo
define tolerance for calculate()

Public Member Functions

- void **calculate** () const

7.295 InterestRate Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/interestrate.hpp>`

7.295.1 Detailed Description

Concrete interest rate class.

This class encapsulate the interest rate compounding algebra. It manages day-counting conventions, compounding conventions, conversion between different conventions, discount/compound factor calculations, and implied/equivalent rate calculations.

Test
Converted rates are checked against known good results

Public Member Functions

constructors

- InterestRate ()
  Default constructor returning a null interest rate.
- InterestRate (Rate r, const DayCounter &dc, Compounding comp, Frequency freq=Annual)
  Standard constructor.

conversions

- **operator Rate** () const

inspectors

- Rate **rate** () const
- const DayCounter & **dayCounter** () const
- Compounding **compounding** () const
- Frequency **frequency** () const

discount/compound factor calculations

- DiscountFactor discountFactor (Time t) const
  discount factor implied by the rate compounded at time t.
- DiscountFactor discountFactor (const Date &d1, const Date &d2, const Date &refStart=Date(), const Date &refEnd=Date()) const
  discount factor implied by the rate compounded between two dates
- Real compoundFactor (Time t) const
  compound factor implied by the rate compounded at time t.
- Real compoundFactor (const Date &d1, const Date &d2, const Date &refStart=Date(), const Date &refEnd=Date()) const

compound factor implied by the rate compounded between two dates

equivalent rate calculations

- InterestRate equivalentRate (Time t, Compounding comp, Frequency freq=Annual) const
  equivalent interest rate for a compounding period t.
- InterestRate equivalentRate (Date d1, Date d2, const DayCounter &resultDC, Compounding comp, Frequency freq=Annual) const
  equivalent rate for a compounding period between two dates

Static Public Member Functions

implied rate calculations

- static InterestRate impliedRate (Real compound, Time t, const DayCounter &resultDC, Compounding comp, Frequency freq=Annual)
  implied interest rate for a given compound factor at a given time.
- static InterestRate impliedRate (Real compound, const Date &d1, const Date &d2, const DayCounter &resultDC, Compounding comp, Frequency freq=Annual)
  implied rate for a given compound factor between two dates.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &, const InterestRate &)

7.295.2 Member Function Documentation

7.295.2.1 DiscountFactor discountFactor (Time *t*) const

discount factor implied by the rate compounded at time t.

Warning:
Time must be measured using InterestRate's own day counter.

7.295.2.2 Real compoundFactor (Time *t*) const

compound factor implied by the rate compounded at time t.

returns the compound (a.k.a capitalization) factor implied by the rate compounded at time t.

Warning:
Time must be measured using InterestRate's own day counter.

7.295.2.3 Real compoundFactor (const Date & *d1*, const Date & *d2*, const Date & *refStart* = Date(), const Date & *refEnd* = Date()) const

compound factor implied by the rate compounded between two dates

returns the compound (a.k.a capitalization) factor implied by the rate compounded between two dates.

7.295.2.4 static InterestRate impliedRate (Real *compound*, Time *t*, const DayCounter & *resultDC*, Compounding *comp*, Frequency *freq* = Annual) [static]

implied interest rate for a given compound factor at a given time.

The resulting InterestRate has the day-counter provided as input.

Warning:
Time must be measured using the day-counter provided as input.

7.295.2.5 static InterestRate impliedRate (Real *compound*, const Date & *d1*, const Date & *d2*, const DayCounter & *resultDC*, Compounding *comp*, Frequency *freq* = Annual) [static]

implied rate for a given compound factor between two dates.

The resulting rate is calculated taking the required day-counting rule into account.

7.295.2.6 InterestRate equivalentRate (Time *t*, Compounding *comp*, Frequency *freq* = Annual) const

equivalent interest rate for a compounding period t.

The resulting InterestRate shares the same implicit day-counting rule of the original InterestRate instance.

Warning:
Time must be measured using the InterestRate's own day counter.

7.295.2.7 InterestRate equivalentRate (Date *d1*, Date *d2*, const DayCounter & *resultDC*, Compounding *comp*, Frequency *freq* = Annual) const

equivalent rate for a compounding period between two dates

The resulting rate is calculated taking the required day-counting rule into account.

7.296 InterpolatedDiscountCurve Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/discountcurve.hpp>`

Inheritance diagram for InterpolatedDiscountCurve:

7.296.1 Detailed Description

template<class Interpolator> class QuantLib::InterpolatedDiscountCurve< Interpolator >

Term structure based on interpolation of discount factors.

Public Member Functions

- **InterpolatedDiscountCurve** (const std::vector< Date > &dates, const std::vector< DiscountFactor > &dfs, const DayCounter &dayCounter, const Interpolator &interpolator=Interpolator())

Inspectors

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the curve can return rates
- Time maxTime () const
  the latest time for which the curve can return rates
- const std::vector< Time > & **times** () const
- const std::vector< Date > & **dates** () const
- const std::vector< DiscountFactor > & **discounts** () const

Protected Member Functions

- **InterpolatedDiscountCurve** (const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedDiscountCurve** (const Date &referenceDate, const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedDiscountCurve** (Integer settlementDays, const Calendar &, const DayCounter &, const Interpolator &interpolator=Interpolator())
- DiscountFactor discountImpl (Time) const
  discount calculation

Protected Attributes

- DayCounter **dayCounter_**
- std::vector< Date > **dates_**
- std::vector< Time > **times_**

- std::vector< DiscountFactor > **data_**
- Interpolation **interpolation_**
- Interpolator **interpolator_**

7.297 InterpolatedForwardCurve Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardcurve.hpp>`

Inheritance diagram for InterpolatedForwardCurve:

7.297.1 Detailed Description

template<class Interpolator> class QuantLib::InterpolatedForwardCurve< Interpolator >

Term structure based on interpolation of forward rates.

Inspectors

- DayCounter dayCounter () const

  the day counter used for date/time conversion

- Date maxDate () const

  the latest date for which the curve can return rates

- Time maxTime () const

  the latest time for which the curve can return rates

- const std::vector< Time > & **times** () const
- const std::vector< Date > & **dates** () const
- **InterpolatedForwardCurve** (const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedForwardCurve** (const Date &referenceDate, const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedForwardCurve** (Integer settlementDays, const Calendar &, const DayCounter &, const Interpolator &interpolator=Interpolator())
- Rate forwardImpl (Time t) const

  instantaneous forward-rate calculation

- Rate zeroYieldImpl (Time t) const
- DayCounter **dayCounter_**
- std::vector< Date > **dates_**
- std::vector< Time > **times_**
- std::vector< Rate > **data_**
- Interpolation **interpolation_**
- Interpolator **interpolator_**

Public Member Functions

- **InterpolatedForwardCurve** (const std::vector< Date > &dates, const std::vector< Rate > &forwards, const DayCounter &dayCounter, const Interpolator &interpolator=Interpolator())

7.297.2 Member Function Documentation

7.297.2.1 Rate zeroYieldImpl (Time *t*) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented from ForwardRateStructure.

7.298 InterpolatedZeroCurve Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zerocurve.hpp>`

Inheritance diagram for InterpolatedZeroCurve:

7.298.1 Detailed Description

template<class Interpolator> class QuantLib::InterpolatedZeroCurve< Interpolator >

Term structure based on interpolation of zero yields.

Inspectors

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the curve can return rates
- Time maxTime () const
  the latest time for which the curve can return rates
- const std::vector< Time > & **times** () const
- const std::vector< Date > & **dates** () const
- **InterpolatedZeroCurve** (const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedZeroCurve** (const Date &referenceDate, const DayCounter &, const Interpolator &interpolator=Interpolator())
- **InterpolatedZeroCurve** (Integer settlementDays, const Calendar &, const DayCounter &, const Interpolator &interpolator=Interpolator())
- Rate zeroYieldImpl (Time t) const
  zero-yield calculation
- DayCounter **dayCounter_**
- std::vector< Date > **dates_**
- std::vector< Time > **times_**
- std::vector< Rate > **data_**
- Interpolation **interpolation_**
- Interpolator **interpolator_**

Public Member Functions

- **InterpolatedZeroCurve** (const std::vector< Date > &dates, const std::vector< Rate > &yields, const DayCounter &dayCounter, const Interpolator &interpolator=Interpolator())

7.299 Interpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation.hpp>`

Inheritance diagram for Interpolation:

7.299.1 Detailed Description

base class for 1-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of equal length, representing discretized values of a variable and a function of the former, respectively.

Public Types

- typedef Real **argument_type**
- typedef Real **result_type**

Public Member Functions

- Real **operator()** (Real x, bool allowExtrapolation=false) const
- Real **primitive** (Real x, bool allowExtrapolation=false) const
- Real **derivative** (Real x, bool allowExtrapolation=false) const
- Real **secondDerivative** (Real x, bool allowExtrapolation=false) const
- Real **xMin** () const
- Real **xMax** () const
- bool **isInRange** (Real x) const
- void **update** ()

Protected Member Functions

- void **checkRange** (Real x, bool allowExtrapolation) const

Classes

- class templateImpl

  basic template implementation

7.300 Interpolation2D Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation2D.hpp>`

Inheritance diagram for Interpolation2D:

7.300.1 Detailed Description

base class for 2-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of length N and M , representing the discretized values of the x and y variables, and a $N \times M$ matrix representing the tabulated function values.

Public Types

- typedef Real **first_argument_type**
- typedef Real **second_argument_type**
- typedef Real **result_type**

Public Member Functions

- Real **operator()** (Real x, Real y, bool allowExtrapolation=false) const
- Real **xMin** () const
- Real **xMax** () const
- Real **yMin** () const
- Real **yMax** () const
- bool **isInRange** (Real x, Real y) const
- void **update** ()

Protected Member Functions

- void **checkRange** (Real x, Real y, bool allowExtrapolation) const

Classes

- class templateImpl

  basic template implementation

7.301 Interpolation2D::templateImpl Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation2D.hpp>`

Inheritance diagram for Interpolation2D::templateImpl:

7.301.1 Detailed Description

template<class I1, class I2, class M> class QuantLib::Interpolation2D::templateImpl< I1, I2, M >

basic template implementation

Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- Real **xMin** () const
- Real **xMax** () const
- Real **yMin** () const
- Real **yMax** () const
- bool **isInRange** (Real x, Real y) const

Protected Member Functions

- Size **locateX** (Real x) const
- Size **locateY** (Real y) const

Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**
- I2 **yEnd_**
- const M & **zData_**

7.302 Interpolation2DImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation2D.hpp>`

Inheritance diagram for Interpolation2DImpl:

7.302.1 Detailed Description

abstract base class for 2-D interpolation implementations

Public Member Functions

- virtual void **calculate** ()=0
- virtual Real **xMin** () const =0
- virtual Real **xMax** () const =0
- virtual Real **yMin** () const =0
- virtual Real **yMax** () const =0
- virtual bool **isInRange** (Real x, Real y) const =0
- virtual Real **value** (Real x, Real y) const =0

7.303 Interpolation::templateImpl Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation.hpp>`

Inheritance diagram for Interpolation::templateImpl:

7.303.1 Detailed Description

template<class I1, class I2> class QuantLib::Interpolation::templateImpl< I1, I2 >

basic template implementation

Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)
- Real **xMin** () const
- Real **xMax** () const
- bool **isInRange** (Real x) const

Protected Member Functions

- Size **locate** (Real x) const

Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**

7.304 InterpolationImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation.hpp>`

Inheritance diagram for InterpolationImpl:

7.304.1 Detailed Description

abstract base class for interpolation implementations

Public Member Functions

- virtual void **calculate** ()=0
- virtual Real **xMin** () const =0
- virtual Real **xMax** () const =0
- virtual bool **isInRange** (Real) const =0
- virtual Real **value** (Real) const =0
- virtual Real **primitive** (Real) const =0
- virtual Real **derivative** (Real) const =0
- virtual Real **secondDerivative** (Real) const =0

7.305 InverseCumulativeNormal Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/normaldistribution.hpp>`

7.305.1 Detailed Description

Inverse cumulative normal distribution function.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It use Acklam's approximation: by Peter J. Acklam, University of Oslo, Statistics Division. URL: http://home.online.no/~pjacklam/notes/invnorm/index.html

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Public Member Functions

- **InverseCumulativeNormal** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const

7.306 InverseCumulativePoisson Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/poissondistribution.hpp>`

7.306.1 Detailed Description

Inverse cumulative Poisson distribution function.

Test
the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **InverseCumulativePoisson** (Real lambda=1.0)
- Real **operator()** (Real x) const

7.307 InverseCumulativeRng Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/inversecumulativerng.hpp>`

7.307.1 Detailed Description

template<class RNG, class IC> class QuantLib::InverseCumulativeRng< RNG, IC >

Inverse cumulative random number generator.

It uses a uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();
Real IC::operator() const;
```

Public Types

- typedef Sample< Real > **sample_type**
- typedef RNG **urng_type**

Public Member Functions

- **InverseCumulativeRng** (const RNG &uniformGenerator)
- sample_type next () const

  returns a sample from a Gaussian distribution

7.308 InverseCumulativeRsg Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/inversecumulativersg.hpp>`

7.308.1 Detailed Description

template<class USG, class IC> class QuantLib::InverseCumulativeRsg< USG, IC >

Inverse cumulative random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::nextSequence() const;
Size USG::dimension() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();
Real IC::operator() const;
```

Public Types

- typedef Sample< Array > **sample_type**

Public Member Functions

- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator)
- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator, const IC &inverseCumulative)
- const sample_type & nextSequence () const

  returns next sample from the Gaussian distribution

- const sample_type & **lastSequence** () const
- Size **dimension** () const

7.309 IQDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for IQDCurrency:

7.309.1 Detailed Description

Iraqi dinar.

The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.

7.310 IRRCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for IRRCurrency:

7.310.1 Detailed Description

Iranian rial.

The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

7.311 ISKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for ISKCurrency:

7.311.1 Detailed Description

Iceland krona.

The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

7.312 Istanbul Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/istanbul.hpp>`

Inheritance diagram for Istanbul:

7.312.1 Detailed Description

Istanbul calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- National Holidays (April 23rd, May 19th, August 30th, October 29th
- Local Holidays (Kurban, Ramadan; 2004 to 2009 only)

7.313 Italy Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/italy.hpp>`

Inheritance diagram for Italy:

7.313.1 Detailed Description

Italian calendars.

Public holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Easter Monday
- Liberation Day, April 25th
- Labour Day, May 1st
- Republic Day, June 2nd (since 2000)
- Assumption, August 15th
- All Saint's Day, November 1st
- Immaculate Conception Day, December 8th
- Christmas Day, December 25th
- St. Stephen's Day, December 26th

Holidays for the stock exchange (data from http://www.borsaitalia.it):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Assumption, August 15th
- Christmas' Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th
- New Year's Eve, December 31st

Test
the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum Market { Settlement, Exchange }
  Italian calendars.

Public Member Functions

- **Italy** (Market market=Settlement)

7.313.2 Member Enumeration Documentation

7.313.2.1 enum Market

Italian calendars.

Enumerator:

Settlement generic settlement calendar

Exchange Milan stock-exchange calendar.

7.314 ITLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for ITLCurrency:

7.314.1 Detailed Description

Italian lira.

The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

7.315 JamshidianSwaptionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/jamshidianswaptionengine.hpp>`

Inheritance diagram for JamshidianSwaptionEngine:

7.315.1 Detailed Description

Jamshidian swaption engine.

Public Member Functions

- **JamshidianSwaptionEngine** (const boost::shared_ptr< OneFactorAffineModel > &model)
- void **calculate** () const

Friends

- class **rStarFinder**

7.316 JarrowRudd Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for JarrowRudd:

7.316.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

Public Member Functions

- **JarrowRudd** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)

7.317 Jibar Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/jibar.hpp>`

Inheritance diagram for Jibar:

7.317.1 Detailed Description

JIBAR rate

Johannesburg Interbank Agreed Rate

Todo

check settlement days and day-count convention.

Public Member Functions

- **Jibar** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual365Fixed())

7.318 Johannesburg Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/johannesburg.hpp>`

Inheritance diagram for Johannesburg:

7.318.1 Detailed Description

Johannesburg calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Family Day, Easter Monday
- Human Rights Day, March 21st (possibly moved to Monday)
- Freedom Day, April 27th (possibly moved to Monday)
- Workers Day, May 1st (possibly moved to Monday)
- Youth Day, June 16th (possibly moved to Monday)
- National Women's Day, August 9th (possibly moved to Monday)
- Heritage Day, September 24th (possibly moved to Monday)
- Day of Reconciliation, December 16th (possibly moved to Monday)
- Christmas December 25th
- Day of Goodwill December 26th (possibly moved to Monday)

7.319 JointCalendar Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/jointcalendar.hpp>`

Inheritance diagram for JointCalendar:

7.319.1 Detailed Description

Joint calendar.

Depending on the chosen rule, this calendar has a set of business days given by either the union or the intersection of the sets of business days of the given calendars.

Test
the correctness of the returned results is tested by reproducing the calculations.

Public Member Functions

- **JointCalendar** (const Calendar &, const Calendar &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const Calendar &, const Calendar &, const Calendar &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const Calendar &, const Calendar &, const Calendar &, const Calendar &, JointCalendarRule=JoinHolidays)

7.320 JPYCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for JPYCurrency:

7.320.1 Detailed Description

Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

7.321 JPYLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/jpylibor.hpp>`

Inheritance diagram for JPYLibor:

7.321.1 Detailed Description

JPY LIBOR rate

Japanese Yen LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Warning:
This is the rate fixed in London by BBA. Use TIBOR if you're interested in the Tokio fixing.

Public Member Functions

- **JPYLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.322 JumpDiffusionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/jumpdiffusionengine.hpp>`

Inheritance diagram for JumpDiffusionEngine:

7.322.1 Detailed Description

Jump-diffusion engine for vanilla options.

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **JumpDiffusionEngine** (const boost::shared_ptr< VanillaOption::engine > &, Real relativeAccuracy_=1e-4, Size maxIterations=100)
- void **calculate** () const

7.323 JuQuadraticApproximationEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/juquadraticengine.hpp>`

Inheritance diagram for JuQuadraticApproximationEngine:

7.323.1 Detailed Description

Pricing engine for American options with Ju quadratic approximation

An Approximate Formula for Pricing American Options Journal of Derivatives Winter 1999 Ju, N.

Warning:
Barone-Adesi-Whaley critical commodity price calculation is used, it has not been modified to see whether the method of Ju is faster. Ju does not say how he solves the equation for the critical stock price, e.g. Newton method. He just gives the solution. The method of BAW gives answers to the same accuracy as in Ju (1999)

Test
the correctness of the returned value is tested by reproducing results available in literature.

Bug
test fails for Borland compiler

Public Member Functions

- void **calculate** () const

7.324 KnuthUniformRng Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/knuthuniformrng.hpp>`

7.324.1 Detailed Description

Uniform random number generator.

Random number generator by Knuth. For more details see Knuth, Seminumerical Algorithms, 3rd edition, Section 3.6.

Note:

This is **not** Knuth's original implementation which is available at http://www-cs-faculty.stanford.edu/~knuth/programs.html, but rather a slightly modified version wrapped in a C++ class. Such modifications did not affect the code but only the data structures used, which were converted to their standard C++ equivalents.

Public Types

- typedef Sample< Real > **sample_type**

Public Member Functions

- KnuthUniformRng (long seed=0)
- sample_type next () const

7.324.2 Constructor & Destructor Documentation

7.324.2.1 KnuthUniformRng (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.324.3 Member Function Documentation

7.324.3.1 KnuthUniformRng::sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.325 KronrodIntegral Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/kronrodintegral.hpp>`

7.325.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

References:

Gauss-Kronrod Integration <http://mathcssun1.emporia.edu/~oneilcat/Experiment-Applet3/ExperimentApplet3.html>

NMS - Numerical Analysis Library <http://www.math.iastate.edu/burkardt/f_-src/nms/nms.html>

Test
the correctness of the result is tested by checking it against known good values.

Public Member Functions

- **KronrodIntegral** (Real tolerance, Size maxFunctionEvaluations=Null< Size >())
- template<class F> Real **operator()** (const F &f, Real a, Real b) const
- Size **functionEvaluations** ()

7.326 KRWCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for KRWCurrency:

7.326.1 Detailed Description

South-Korean won.

The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

7.327 KWDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for KWDCurrency:

7.327.1 Detailed Description

Kuwaiti dinar.

The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

7.328 Lattice Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice.hpp>`

Inheritance diagram for Lattice:

7.328.1 Detailed Description

template<class Impl> class QuantLib::Lattice< Impl >

Lattice-method base class.

This class defines a lattice method that is able to rollback (with discount) a discretized asset object. It will usually be based on one or more trees.

Derived classes must implement the following interface:

```
public:
  DiscountFactor discount(Size i, Size index) const;
  Size descendant(Size i, Size index, Size branch) const;
  Real probability(Size i, Size index, Size branch) const;
```

and may implement the following:

```
public:
  void stepback(Size i,
                const Array& values,
                Array& newValues) const;
```

Public Member Functions

- **Lattice** (const TimeGrid &timeGrid, Size n)
- const Array & **statePrices** (Size i) const
- void **stepback** (Size i, const Array &values, Array &newValues) const

NumericalMethod interface

- void initialize (DiscretizedAsset &, Time t) const
  initialize an asset at the given time.
- void rollback (DiscretizedAsset &, Time to) const
- void partialRollback (DiscretizedAsset &, Time to) const
- Real presentValue (DiscretizedAsset &) const
  Computes the present value of an asset using Arrow-Debrew prices.

Protected Member Functions

- void **computeStatePrices** (Size until) const

Protected Attributes

- std::vector< Array > **statePrices_**

7.328.2 Member Function Documentation

7.328.2.1 void rollback (DiscretizedAsset &, Time *to*) const [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements NumericalMethod.

7.328.2.2 void partialRollback (DiscretizedAsset &, Time *to*) const [virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

Warning:

In version 0.3.7 and earlier, this method was called rollAlmostBack method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);
asset->preAdjustValues();
```

Implements NumericalMethod.

7.329 Lattice1D Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice1d.hpp>`

Inheritance diagram for Lattice1D:

7.329.1 Detailed Description

template<class Impl> class QuantLib::Lattice1D< Impl >

One-dimensional lattice.

Derived classes must implement the following interface:

```
Real underlying(Size i, Size index) const;
```

Public Member Functions

- **Lattice1D** (const TimeGrid &timeGrid, Size n)
- Disposable< Array > **grid** (Time t) const
- Real **underlying** (Size i, Size index) const

7.330 Lattice2D Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice2d.hpp>`

Inheritance diagram for Lattice2D:

7.330.1 Detailed Description

template<class Impl, class T = TrinomialTree> class QuantLib::Lattice2D< Impl, T >

Two-dimensional lattice.

This lattice is based on two trinomial trees and primarily used for the G2 short-rate model.

Public Member Functions

- **Lattice2D** (const boost::shared_ptr< T > &tree1, const boost::shared_ptr< T > &tree2, Real correlation)
- Size **size** (Size i) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

Protected Member Functions

- Disposable< Array > **grid** (Time) const

Protected Attributes

- boost::shared_ptr< T > **tree1_**
- boost::shared_ptr< T > **tree2_**

7.331 LatticeShortRateModelEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/latticeshortratemodelengine.hpp>`

Inheritance diagram for LatticeShortRateModelEngine:

7.331.1 Detailed Description

template<class Arguments, class Results> class QuantLib::LatticeShortRateModelEngine< Arguments, Results >

Engine for a short-rate model specialized on a lattice.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **LatticeShortRateModelEngine** (const boost::shared_ptr< ShortRateModel > &model, Size timeSteps)
- **LatticeShortRateModelEngine** (const boost::shared_ptr< ShortRateModel > &model, const TimeGrid &timeGrid)
- void update ()

Protected Attributes

- TimeGrid **timeGrid_**
- Size **timeSteps_**
- boost::shared_ptr< NumericalMethod > **lattice_**

7.331.2 Member Function Documentation

7.331.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from GenericModelEngine< ShortRateModel, Arguments, Results >.

7.332 LazyObject Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/lazyobject.hpp>`

Inheritance diagram for LazyObject:

7.332.1 Detailed Description

Framework for calculation on demand and result caching.

Calculations

These methods do not modify the structure of the object and are therefore declared as `const`. Data members which will be calculated on demand need to be declared as mutable.

- void recalculate ()
- void freeze ()
- void unfreeze ()
- virtual void calculate () const
- virtual void performCalculations () const =0

Public Member Functions

Observer interface

- void update ()

Protected Attributes

- bool **calculated_**
- bool **frozen_**

7.332.2 Member Function Documentation

7.332.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

Reimplemented in AffineTermStructure, and PiecewiseYieldCurve.

7.332.2.2 void recalculate ()

This method force the recalculation of any results which would otherwise be cached. It is not declared as `const` since it needs to call the non-`const` **notifyObservers** method.

Note:
Explicit invocation of this method is **not** necessary if the object registered itself as observer with the structures on which such results depend. It is strongly advised to follow this policy when possible.

7.332.2.3 void freeze ()

This method constrains the object to return the presently cached results on successive invocations, even if arguments upon which they depend should change.

7.332.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

7.332.2.5 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:
Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.
Should this method be redefined in derived classes, LazyObject::calculate() should be called in the overriding method.

Reimplemented in Instrument.

7.332.2.6 virtual void performCalculations () const [protected, pure virtual]

This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in Instrument, BarrierOption, Bond, ForwardVanillaOption, MultiAssetOption, OneAssetOption, OneAssetStrikedOption, QuantoVanillaOption, Stock, and Swap.

7.333 LeastSquareFunction Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/leastsquare.hpp>`

Inheritance diagram for LeastSquareFunction:

7.333.1 Detailed Description

Cost function for least-square problems.

Implements a cost function using the interface provided by the LeastSquareProblem class.

Public Member Functions

- LeastSquareFunction (LeastSquareProblem &lsp)
  Default constructor.
- virtual ~LeastSquareFunction ()
  Destructor.
- virtual Real value (const Array &x) const
  compute value of the least square function
- virtual void gradient (Array &grad_f, const Array &x) const
  compute vector of derivatives of the least square function
- virtual Real valueAndGradient (Array &grad_f, const Array &x) const
  compute value and gradient of the least square function

Protected Attributes

- LeastSquareProblem & lsp_
  least square problem

7.334 LeastSquareProblem Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/leastsquare.hpp>`

7.334.1 Detailed Description

Base class for least square problem.

Public Member Functions

- virtual Size size ()=0
  size of the problem ie size of target vector
- virtual void targetAndValue (const Array &x, Array &target, Array &fct2fit)=0
  compute the target vector and the values of the function to fit
- virtual void targetValueAndGradient (const Array &x, Matrix &grad_fct2fit, Array &target, Array &fct2fit)=0

7.334.2 Member Function Documentation

7.334.2.1 virtual void targetValueAndGradient (const Array & *x*, Matrix & *grad_fct2fit*, Array & *target*, Array & *fct2fit*) [pure virtual]

compute the target vector, the values of the function to fit and the matrix of derivatives

7.335 LecuyerUniformRng Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/lecuyeruniformrng.hpp>`

7.335.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (know as ran2 in Numerical recipes)

For more details see Section 7.1 of Numerical Recipes in C, 2nd Edition, Cambridge University Press (available at http://www.nr.com/)

Public Types

- typedef Sample< Real > **sample_type**

Public Member Functions

- LecuyerUniformRng (long seed=0)
- sample_type next () const

7.335.2 Constructor & Destructor Documentation

7.335.2.1 LecuyerUniformRng (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.335.3 Member Function Documentation

7.335.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.336 LeisenReimer Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for LeisenReimer:

7.336.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

Public Member Functions

- **LeisenReimer** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

Protected Attributes

- Real **up_**
- Real **down_**
- Real **pu_**
- Real **pd_**

7.337 LexicographicalView Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/lexicographicalview.hpp>`

7.337.1 Detailed Description

template<class RandomAccessIterator> class QuantLib::LexicographicalView< RandomAccessIterator >

Lexicographical 2-D view of a contiguous set of data.

This view can be used to easily store a discretized 2-D function in an array to be used in a finite differences calculation.

Public Types

- typedef RandomAccessIterator x_iterator

  iterates over v_{ij} with j fixed.

- typedef boost::reverse_iterator< RandomAccessIterator > reverse_x_iterator

  iterates backwards over v_{ij} with j fixed.

- typedef step_iterator< RandomAccessIterator > y_iterator

  iterates over v_{ij} with i fixed.

- typedef boost::reverse_iterator< y_iterator > reverse_y_iterator

  iterates backwards over v_{ij} with i fixed.

Public Member Functions

- LexicographicalView (const RandomAccessIterator &begin, const RandomAccessIterator &end, Size xSize)

  attaches the view with the given dimension to a sequence

Element access

- y_iterator **operator[ ]** (Size i)

Iterator access

- x_iterator **xbegin** (Size j)
- x_iterator **xend** (Size j)
- reverse_x_iterator **rxbegin** (Size j)
- reverse_x_iterator **rxend** (Size j)
- y_iterator **ybegin** (Size i)
- y_iterator **yend** (Size i)
- reverse_y_iterator **rybegin** (Size i)
- reverse_y_iterator **ryend** (Size i)

Inspectors

- Size xSize () const
  dimension of the array along x
- Size ySize () const
  dimension of the array along y

7.338 Libor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/libor.hpp>`

Inheritance diagram for Libor:

7.338.1 Detailed Description

base class for BBA LIBOR indexes

Public Member Functions

- **Libor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency &currency, const Calendar &localCalendar, const Calendar &currency-Calendar, BusinessDayConvention convention, const DayCounter &dayCounter, const Handle< YieldTermStructure > &h)

Date calculations

see `http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1412`

- Date **valueDate** (const Date &fixingDate) const
- Date **maturityDate** (const Date &valueDate) const

7.339 Linear Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/linearinterpolation.hpp>`

7.339.1 Detailed Description

Linear interpolation factory and traits.

Public Types

- enum { **global** = 0 }

Public Member Functions

- template<class I1, class I2> Interpolation **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

7.340 LinearInterpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/linearinterpolation.hpp>`

Inheritance diagram for LinearInterpolation:

7.340.1 Detailed Description

Linear interpolation between discrete points

Public Member Functions

- template<class I1, class I2> LinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.340.2 Constructor & Destructor Documentation

7.340.2.1 LinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:
the *x* values must be sorted.

7.341 LineSearch Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/linesearch.hpp>`

Inheritance diagram for LineSearch:

7.341.1 Detailed Description

Base class for line search.

Public Member Functions

- LineSearch (Real eps=1e-8)
  Default constructor.
- virtual ~LineSearch ()
  Destructor.
- const Array & lastX ()
  return last x value
- Real lastFunctionValue ()
  return last cost function value
- const Array & lastGradient ()
  return last gradient
- Real lastGradientNorm2 ()
  return square norm of last gradient
- bool **succeed** ()
- virtual Real operator() (const Problem &P, Real t_ini)=0
  Perform line search.
- Real **update** (Array &params, const Array &direction, Real beta, const Constraint &constraint)

Protected Attributes

- Array xtd_
  new x and its gradient
- Array **gradient_**
- Real qt_
  *cost function value and gradient norm corresponding to xtd_*
- Real **qpt_**
- bool succeed_
  flag to know if linesearch succeed

7.342 Link Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/handle.hpp>`

Inheritance diagram for Link:

7.342.1 Detailed Description

template<class Type> class QuantLib::Link< Type >

Relinkable access to a shared pointer.

Precondition:
Class "Type" must inherit from Observable

Public Member Functions

- Link (const boost::shared_ptr< Type > &h=boost::shared_ptr< Type >(), bool registerAsObserver=true)
- void linkTo (const boost::shared_ptr< Type > &, bool registerAsObserver=true)
- bool empty () const
  Checks if the contained shared pointer points to anything.
- const boost::shared_ptr< Type > & currentLink () const
  Returns the contained shared pointer.
- void update ()
  Observer interface.

7.342.2 Constructor & Destructor Documentation

7.342.2.1 Link (const boost::shared_ptr< Type > & *h* = `boost::shared_ptr< Type >()`, bool *registerAsObserver* = `true`) `[explicit]`

Warning:
see the documentation of the linkTo() method for issues relatives to `registerAsObserver`.

7.342.3 Member Function Documentation

7.342.3.1 void linkTo (const boost::shared_ptr< Type > &, bool *registerAsObserver* = `true`)

Warning:
`registerAsObserver` is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be set to `false` when the passed shared pointer was created with `owns = false` (the latter should only happen in a controlled environment, so that the programmer is aware of it). Failure to do so can very likely result in a program crash. If the programmer does want the handle to register as observer of such a shared pointer, it is his responsibility to ensure that the handle gets destroyed before the pointed object does.

7.343 LocalConstantVol Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localconstantvol.hpp>`

Inheritance diagram for LocalConstantVol:

7.343.1 Detailed Description

Constant local volatility, no time-strike dependence.

This class implements the LocalVolatilityTermStructure interface for a constant local volatility (no time/asset dependence). Local volatility and Black volatility are the same when volatility is at most time dependent, so this class is basically a proxy for BlackVolatilityTermStructure.

Public Member Functions

- **LocalConstantVol** (const Date &referenceDate, Volatility volatility, const DayCounter &dayCounter)
- **LocalConstantVol** (const Date &referenceDate, const Handle< Quote > &volatility, const DayCounter &dayCounter)
- **LocalConstantVol** (Integer settlementDays, const Calendar &, Volatility volatility, const DayCounter &dayCounter)
- **LocalConstantVol** (Integer settlementDays, const Calendar &, const Handle< Quote > &volatility, const DayCounter &dayCounter)

LocalVolTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.344 LocalVolCurve Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localvolcurve.hpp>`

Inheritance diagram for LocalVolCurve:

7.344.1 Detailed Description

Local volatility curve derived from a Black curve.

Public Member Functions

- **LocalVolCurve** (const Handle< BlackVarianceCurve > &curve)

LocalVolTermStructure interface

- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- Volatility localVolImpl (Time, Real) const

7.344.2 Member Function Documentation

7.344.2.1 Volatility localVolImpl (Time *t*, Real *dummy*) const [protected, virtual]

The relation

$$\int_0^T \sigma_L^2(t)dt = \sigma_B^2 T$$

holds, where $\sigma_L(t)$ is the local volatility at time t and $\sigma_B(T)$ is the Black volatility for maturity T . From the above, the formula

$$\sigma_L(t) = \sqrt{\frac{\mathrm{d}}{\mathrm{d}t}\sigma_B^2(t)t}$$

can be deduced which is here implemented.

Implements LocalVolTermStructure.

7.345 LocalVolSurface Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localvolsurface.hpp>`

Inheritance diagram for LocalVolSurface:

7.345.1 Detailed Description

Local volatility surface derived from a Black vol surface.

For details about this implementation refer to "Stochastic Volatility and Local Volatility," in "Case Studies and Financial Modelling Course Notes," by Jim Gatheral, Fall Term, 2003

see www.math.nyu.edu/fellows_fin_math/gatheral/Lecture1_Fall02.pdf

Bug
this class is untested, probably unreliable.

Public Member Functions

- **LocalVolSurface** (const Handle< BlackVolTermStructure > &blackTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< YieldTermStructure > &dividendTS, const Handle< Quote > &underlying)
- **LocalVolSurface** (const Handle< BlackVolTermStructure > &blackTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< YieldTermStructure > &dividendTS, Real underlying)

LocalVolTermStructure interface

- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Date maxDate () const
  the latest date for which the term structure can return vols
- Real minStrike () const
  the minimum strike for which the term structure can return vols
- Real maxStrike () const
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

- Volatility localVolImpl (Time, Real) const
  local vol calculation

7.346 LocalVolTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/voltermstructure.hpp>`

Inheritance diagram for LocalVolTermStructure:

7.346.1 Detailed Description

Local-volatility term structure.

This abstract class defines the interface of concrete local-volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- LocalVolTermStructure ()
  default constructor
- LocalVolTermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- LocalVolTermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Local Volatility

- Volatility **localVol** (const Date &d, Real underlyingLevel, bool extrapolate=false) const
- Volatility **localVol** (Time t, Real underlyingLevel, bool extrapolate=false) const

Limits

- virtual Date maxDate () const =0
  the latest date for which the term structure can return vols
- Time maxTime () const
  the latest time for which the term structure can return vols
- virtual Real minStrike () const =0
  the minimum strike for which the term structure can return vols
- virtual Real maxStrike () const =0
  the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual volatility calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual Volatility localVolImpl (Time t, Real strike) const =0

  local vol calculation

7.346.2 Constructor & Destructor Documentation

7.346.2.1 LocalVolTermStructure ()

default constructor

Warning:

term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.347 LogLinear Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/loglinearinterpolation.hpp>`

7.347.1 Detailed Description

log-linear interpolation factory and traits

Public Types

- enum { **global** = 0 }

Public Member Functions

- template<class I1, class I2> Interpolation **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

7.348 LogLinearInterpolation Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/loglinearinterpolation.hpp>`

Inheritance diagram for LogLinearInterpolation:

7.348.1 Detailed Description

log-linear interpolation between discrete points

Todo

implement primitive, derivative, and secondDerivative functions.

Public Member Functions

- template<class I1, class I2> LogLinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.348.2 Constructor & Destructor Documentation

7.348.2.1 LogLinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:

the *x* values must be sorted.

7.349 LTLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for LTLCurrency:

7.349.1 Detailed Description

Lithuanian litas.

The ISO three-letter code is LTL; the numeric code is 440. It is divided in 100 centu.

7.350 LUFCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for LUFCurrency:

7.350.1 Detailed Description

Luxembourg franc.

The ISO three-letter code is LUF; the numeric code is 442. It is divided in 100 centimes.

7.351 LVLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for LVLCurrency:

7.351.1 Detailed Description

Latvian lat.

The ISO three-letter code is LVL; the numeric code is 428. It is divided in 100 santims.

7.352 MakeMCDigitalEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mcdigitalengine.hpp>`

7.352.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCDigitalEngine< RNG, S >

Monte Carlo digital engine factory.

Public Member Functions

- MakeMCDigitalEngine & **withSteps** (Size steps)
- MakeMCDigitalEngine & **withStepsPerYear** (Size steps)
- MakeMCDigitalEngine & **withBrownianBridge** (bool b=true)
- MakeMCDigitalEngine & **withSamples** (Size samples)
- MakeMCDigitalEngine & **withTolerance** (Real tolerance)
- MakeMCDigitalEngine & **withMaxSamples** (Size samples)
- MakeMCDigitalEngine & **withSeed** (BigNatural seed)
- MakeMCDigitalEngine & **withAntitheticVariate** (bool b=true)
- MakeMCDigitalEngine & **withControlVariate** (bool b=true)
- **operator boost::shared_ptr** () const

7.353 MakeMCEuropeanEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mceuropeanengine.hpp>`

7.353.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCEuropeanEngine< RNG, S >

Monte Carlo European engine factory.

Public Member Functions

- MakeMCEuropeanEngine & **withSteps** (Size steps)
- MakeMCEuropeanEngine & **withStepsPerYear** (Size steps)
- MakeMCEuropeanEngine & **withBrownianBridge** (bool b=true)
- MakeMCEuropeanEngine & **withSamples** (Size samples)
- MakeMCEuropeanEngine & **withTolerance** (Real tolerance)
- MakeMCEuropeanEngine & **withMaxSamples** (Size samples)
- MakeMCEuropeanEngine & **withSeed** (BigNatural seed)
- MakeMCEuropeanEngine & **withAntitheticVariate** (bool b=true)
- MakeMCEuropeanEngine & **withControlVariate** (bool b=true)
- **operator boost::shared_ptr** () const

7.354 MakeMCEuropeanHestonEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mceuropeanhestonengine.hpp>`

7.354.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCEuropeanHestonEngine< RNG, S >

Monte Carlo Heston European engine factory.

Public Member Functions

- MakeMCEuropeanHestonEngine & **withSteps** (Size steps)
- MakeMCEuropeanHestonEngine & **withStepsPerYear** (Size steps)
- MakeMCEuropeanHestonEngine & **withSamples** (Size samples)
- MakeMCEuropeanHestonEngine & **withTolerance** (Real tolerance)
- MakeMCEuropeanHestonEngine & **withMaxSamples** (Size samples)
- MakeMCEuropeanHestonEngine & **withSeed** (BigNatural seed)
- MakeMCEuropeanHestonEngine & **withAntitheticVariate** (bool b=true)
- **operator boost::shared_ptr** () const

7.355 MakeSchedule Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/schedule.hpp>`

7.355.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

Public Member Functions

- **MakeSchedule** (const Calendar &calendar, const Date &startDate, const Date &endDate, Frequency frequency, BusinessDayConvention convention)
- MakeSchedule & **withStubDate** (const Date &d)
- MakeSchedule & **backwards** (bool flag=true)
- MakeSchedule & **forwards** (bool flag=true)
- MakeSchedule & **longFinalPeriod** (bool flag=true)
- MakeSchedule & **shortFinalPeriod** (bool flag=true)
- **operator Schedule** ()

7.356 Matrix Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/matrix.hpp>`

7.356.1 Detailed Description

Matrix used in linear algebra.

This class implements the concept of Matrix as used in linear algebra. As such, it is **not** meant to be used as a container.

Public Types

- typedef Real * **iterator**
- typedef const Real * **const_iterator**
- typedef boost::reverse_iterator< iterator > **reverse_iterator**
- typedef boost::reverse_iterator< const_iterator > **const_reverse_iterator**
- typedef Real * **row_iterator**
- typedef const Real * **const_row_iterator**
- typedef boost::reverse_iterator< row_iterator > **reverse_row_iterator**
- typedef boost::reverse_iterator< const_row_iterator > **const_reverse_row_iterator**
- typedef step_iterator< iterator > **column_iterator**
- typedef step_iterator< const_iterator > **const_column_iterator**
- typedef boost::reverse_iterator< column_iterator > **reverse_column_iterator**
- typedef boost::reverse_iterator< const_column_iterator > **const_reverse_column_iterator**

Public Member Functions

Constructors, destructor, and assignment

- Matrix ()
  creates a null matrix
- Matrix (Size rows, Size columns)
  creates a matrix with the given dimensions
- Matrix (Size rows, Size columns, Real value)
  creates the matrix and fills it with `value`
- **Matrix** (const Matrix &)
- **Matrix** (const Disposable< Matrix > &)
- Matrix & **operator=** (const Matrix &)
- Matrix & **operator=** (const Disposable< Matrix > &)

Algebraic operators

- const Matrix & operator+= (const Matrix &)
- const Matrix & **operator-=** (const Matrix &)
- const Matrix & **operator *=** (Real)
- const Matrix & **operator/=** (Real)

Iterator access

- const_iterator **begin** () const
- iterator **begin** ()
- const_iterator **end** () const
- iterator **end** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rend** () const
- reverse_iterator **rend** ()
- const_row_iterator **row_begin** (Size i) const
- row_iterator **row_begin** (Size i)
- const_row_iterator **row_end** (Size i) const
- row_iterator **row_end** (Size i)
- const_reverse_row_iterator **row_rbegin** (Size i) const
- reverse_row_iterator **row_rbegin** (Size i)
- const_reverse_row_iterator **row_rend** (Size i) const
- reverse_row_iterator **row_rend** (Size i)
- const_column_iterator **column_begin** (Size i) const
- column_iterator **column_begin** (Size i)
- const_column_iterator **column_end** (Size i) const
- column_iterator **column_end** (Size i)
- const_reverse_column_iterator **column_rbegin** (Size i) const
- reverse_column_iterator **column_rbegin** (Size i)
- const_reverse_column_iterator **column_rend** (Size i) const
- reverse_column_iterator **column_rend** (Size i)

Element access

- const_row_iterator **operator[ ]** (Size) const
- row_iterator **operator[ ]** (Size)
- Disposable< Array > **diagonal** (void) const

Inspectors

- Size **rows** () const
- Size **columns** () const
- bool **empty** () const

Utilities

- void **swap** (Matrix &)

Related Functions

(Note that these are not member functions.)

- const Disposable< Matrix > **CholeskyDecomposition** (const Matrix &m, bool flexible=false)

- const Disposable< Matrix > **operator+** (const Matrix &, const Matrix &)
- const Disposable< Matrix > **operator-** (const Matrix &, const Matrix &)
- const Disposable< Matrix > **operator** * (const Matrix &, Real)
- const Disposable< Matrix > **operator** * (Real, const Matrix &)
- const Disposable< Matrix > **operator/** (const Matrix &, Real)
- const Disposable< Array > **operator** * (const Array &, const Matrix &)

- const Disposable< Array > **operator** * (const Matrix &, const Array &)
- const Disposable< Matrix > **operator** * (const Matrix &, const Matrix &)
- const Disposable< Matrix > **transpose** (const Matrix &)
- const Disposable< Matrix > **outerProduct** (const Array &v1, const Array &v2)
- template<class Iterator1, class Iterator2> const Disposable< Matrix > **outerProduct** (Iterator1 v1begin, Iterator1 v1end, Iterator2 v2begin, Iterator2 v2end)
- void **swap** (Matrix &, Matrix &)
- std::ostream & **operator**<< (std::ostream &, const Matrix &)
- const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type)

  Returns the pseudo square root of a real symmetric matrix.

- const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)

7.356.2 Member Function Documentation

7.356.2.1 const Matrix & operator+= (const Matrix &)

Precondition:
all matrices involved in an algebraic expression must have the same size.

7.356.3 Friends And Related Function Documentation

7.356.3.1 const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type) [related]

Returns the pseudo square root of a real symmetric matrix.

Given a matrix M , the result S is defined as the matrix such that $SS^T = M$. If the matrix is not positive semi definite, it can return an approximation of the pseudo square root using a (user selected) salvaging algorithm.

For more information see: "The most general methodology to create a valid correlation matrix for risk management and option pricing purposes", by R. Rebonato and P. Jäckel. The Journal of Risk, 2(2), Winter 1999/2000 http://www.rebonato.com/correlationmatrix.pdf

Revised and extended in "Monte Carlo Methods in Finance", by Peter Jäckel, Chapter 6.

Precondition:
the given matrix must be symmetric.

Todo

- implement Hypersphere decomposition:
  1. Jäckel "Monte Carlo Methods in Finance", Chapter 6
  2. Brigo "A Note on Correlation and Rank Reduction"
  3. Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
- implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Test

- the correctness of the results is tested by reproducing known good data.
- the correctness of the results is tested by checking returned values against numerical calculations.

7.356.3.2 const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size *maxRank*, Real *componentRetainedPercentage*, SalvagingAlgorithm::Type) [related]

Precondition:
the given matrix must be symmetric.

7.357 MCAmericanBasketEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Basket/mcamericanbasketengine.hpp>`

Inheritance diagram for MCAmericanBasketEngine:

7.357.1 Detailed Description

least-square Monte Carlo engine

Warning:
This method is intrinsically weak for out-of-the-money options.

Bug
this engine does not yet work for put options. More problems might surface.

Public Member Functions

- **MCAmericanBasketEngine** (Size requiredSamples, Size timeSteps, BigNatural seed=0)
- void **calculate** () const

7.358 MCBarrierEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Barrier/mcbarrierengine.hpp>`

Inheritance diagram for MCBarrierEngine:

7.358.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBarrierEngine< RNG, S >

Pricing engine for barrier options using Monte Carlo simulation.

Uses the Brownian-bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation - D.R. Beaglehole, P.H. Dybvig and G. Zhou Financial Analysts Journal; Jan/Feb 1997; 53, 1. pg. 62-68* and *Simulating path-dependent options: A new approach - M. El Babsiri and G. Noel Journal of Derivatives; Winter 1998; 6, 2; pg. 65-83*

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef McSimulation< SingleVariate< RNG >, S >::path_generator_type **path_generator_type**
- typedef McSimulation< SingleVariate< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< SingleVariate< RNG >, S >::stats_type **stats_type**

Public Member Functions

- **MCBarrierEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, bool isBiased, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- TimeGrid **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

Protected Attributes

- Size **maxTimeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **isBiased_**

- bool **brownianBridge_**
- BigNatural **seed_**

7.359 MCBasketEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Basket/mcbasketengine.hpp>`

Inheritance diagram for MCBasketEngine:

7.359.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBasketEngine< RNG, S >

Pricing engine for basket options using Monte Carlo simulation.

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef McSimulation< MultiVariate< RNG >, S >::path_generator_type **path_generator_type**
- typedef McSimulation< MultiVariate< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< MultiVariate< RNG >, S >::stats_type **stats_type**

Public Member Functions

- **MCBasketEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- TimeGrid **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

Protected Attributes

- Size **maxTimeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

7.360 McCliquetOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mccliquetoption.hpp>`

Inheritance diagram for McCliquetOption:

7.360.1 Detailed Description

simple example of Monte Carlo pricer

Public Member Functions

- **McCliquetOption** (Option::Type type, Real underlying, Real moneyness, const Handle< YieldTermStructure > &dividendYield, const Handle< YieldTermStructure > &riskFreeRate, const Handle< BlackVolTermStructure > &volatility, const std::vector< Time > &times, Real accruedCoupon, Real lastFixing, Real localCap, Real localFloor, Real globalCap, Real globalFloor, bool redemptionOnly, BigNatural seed=0)

7.361 MCDigitalEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mcdigitalengine.hpp>`

Inheritance diagram for MCDigitalEngine:

7.361.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDigitalEngine< RNG, S >

Pricing engine for digital options using Monte Carlo simulation.

Uses the Brownian Bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation - D.R. Beaglehole, P.H. Dybvig and G. Zhou Financial Analysts Journal; Jan/Feb 1997; 53, 1. pg. 62-68* and *Simulating path-dependent options: A new approach - M. El Babsiri and G. Noel Journal of Derivatives; Winter 1998; 6, 2; pg. 65-83*

Test

the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Public Types

- typedef MCVanillaEngine< RNG, S >::path_generator_type **path_generator_type**
- typedef MCVanillaEngine< RNG, S >::path_pricer_type **path_pricer_type**
- typedef MCVanillaEngine< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDigitalEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.362 MCDiscreteArithmeticAPEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp>`

Inheritance diagram for MCDiscreteArithmeticAPEngine:

7.362.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteArithmeticAPEngine< RNG, S >

Monte Carlo pricing engine for discrete arithmetic average price Asian.

Monte Carlo pricing engine for discrete arithmetic average price Asian options. It can use MCDiscreteGeometricAPEngine (Monte Carlo discrete arithmetic average price engine) and AnalyticDiscreteGeometricAveragePriceAsianEngine (analytic discrete arithmetic average price engine) for control variation.

Test

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef MCDiscreteAveragingAsianEngine< RNG, S >::path_generator_type **path_generator_type**
- typedef MCDiscreteAveragingAsianEngine< RNG, S >::path_pricer_type **path_pricer_type**
- typedef MCDiscreteAveragingAsianEngine< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteArithmeticAPEngine** (Size maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural)

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const
- boost::shared_ptr< path_pricer_type > **controlPathPricer** () const
- boost::shared_ptr< PricingEngine > **controlPricingEngine** () const

7.363 McDiscreteArithmeticASO Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcdiscretearithmeticaso.hpp>`

Inheritance diagram for McDiscreteArithmeticASO:

7.363.1 Detailed Description

example of Monte Carlo pricer using a control variate.

Todo

continous-averaging version

Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, Real underlying, const Handle< YieldTermStructure > &dividendYield, const Handle< YieldTermStructure > &riskFreeRate, const Handle< BlackVolTermStructure > &volatility, const std::vector< Time > &times, bool controlVariate, BigNatural seed=0)

7.364 MCDiscreteAveragingAsianEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Asian/mcdiscreteasianengine.hpp>`

Inheritance diagram for MCDiscreteAveragingAsianEngine:

7.364.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscrete-AveragingAsianEngine< RNG, S >

Pricing engine for discrete average Asians using Monte Carlo simulation.

Warning:
control-variate calculation is disabled under VC++6

Public Types

- typedef McSimulation< SingleVariate< RNG >, S >::path_generator_type **path_generator_-type**
- typedef McSimulation< SingleVariate< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< SingleVariate< RNG >, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteAveragingAsianEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- TimeGrid **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- Real **controlVariateValue** () const

Protected Attributes

- Size **maxTimeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

7.365 MCDiscreteGeometricAPEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>`

Inheritance diagram for MCDiscreteGeometricAPEngine:

7.365.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteGeometricAPEngine< RNG, S >

Monte Carlo pricing engine for discrete geometric average price Asian.

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef MCDiscreteAveragingAsianEngine< RNG, S >::path_generator_type **path_generator_type**
- typedef MCDiscreteAveragingAsianEngine< RNG, S >::path_pricer_type **path_pricer_type**
- typedef MCDiscreteAveragingAsianEngine< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteGeometricAPEngine** (Size maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.366 MCEuropeanEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mceuropeanengine.hpp>`

Inheritance diagram for MCEuropeanEngine:

7.366.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropean-Engine< RNG, S >

European option pricing engine using Monte Carlo simulation.

Test
the correctness of the returned value is tested by checking it against analytic results.

Public Types

- typedef MCVanillaEngine< RNG, S >::path_generator_type **path_generator_type**
- typedef MCVanillaEngine< RNG, S >::path_pricer_type **path_pricer_type**
- typedef MCVanillaEngine< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCEuropeanEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.367 MCEuropeanHestonEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mceuropeanhestonengine.hpp>`

Inheritance diagram for MCEuropeanHestonEngine:

7.367.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropeanHestonEngine< RNG, S >

Monte Carlo Heston-model engine for European options.

Test

the correctness of the returned value is tested by reproducing results available in web/literature

Public Types

- typedef MCHestonEngine< RNG, S >::path_pricer_type **path_pricer_type**

Public Member Functions

- **MCEuropeanHestonEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.368 McEverest Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mceverest.hpp>`

Inheritance diagram for McEverest:

7.368.1 Detailed Description

Everest-type option pricer.

The payoff of an Everest option is simply given by the final price / initial price ratio of the worst performer

Public Member Functions

- **McEverest** (const std::vector< Handle< YieldTermStructure > > &dividendYield, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, Time residualTime, BigNatural seed=0)

7.369 MCHestonEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mchestonengine.hpp>`

Inheritance diagram for MCHestonEngine:

7.369.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCHeston-Engine< RNG, S >

Monte Carlo Heston-model engine.

Public Member Functions

- void **calculate** () const

Protected Types

- typedef McSimulation< MultiVariate< RNG >, S >::path_generator_type **path_generator_-type**
- typedef McSimulation< MultiVariate< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< MultiVariate< RNG >, S >::stats_type **stats_type**

Protected Member Functions

- **MCHestonEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- TimeGrid **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const

Protected Attributes

- Size **timeSteps_**
- Size **timeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

7.370 McHimalaya Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mchimalaya.hpp>`

Inheritance diagram for McHimalaya:

7.370.1 Detailed Description

Himalayan-type option pricer.

The payoff of a Himalaya option is computed in the following way: Given a basket of N assets, and N time periods, at end of each period the option who performed the best is added to the average and then discarded from the basket. At the end of the N periods the option pays the max between the strike and the average of the best performers.

Public Member Functions

- **McHimalaya** (const std::vector< Real > &underlyings, const std::vector< Handle< YieldTermStructure > > &dividendYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, Real strike, const std::vector< Time > &times, BigNatural seed=0)

7.371 McMaxBasket Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcmaxbasket.hpp>`

Inheritance diagram for McMaxBasket:

7.371.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

Public Member Functions

- **McMaxBasket** (const std::vector< Real > &underlyings, const std::vector< Handle< YieldTermStructure > > &dividendYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, Time residualTime, BigNatural seed=0)

7.372 McPagoda Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcpagoda.hpp>`

Inheritance diagram for McPagoda:

7.372.1 Detailed Description

roofed Asian option

Given a certain portfolio of assets at the end of the period it is returned the minimum of a given roof and a certain fraction of the positive portfolio performance. If the performance of the portfolio is below then the payoff is null.

Public Member Functions

- **McPagoda** (const std::vector< Real > &underlyings, Real fraction, Real roof, const std::vector< Handle< YieldTermStructure > > &dividendYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, const std::vector< Time > &times, BigNatural seed=0)

7.373 McPerformanceOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcperformanceoption.hpp>`

Inheritance diagram for McPerformanceOption:

7.373.1 Detailed Description

Performance option computed using Monte Carlo simulation.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is $ max(S/X- 1) $.

Public Member Functions

- **McPerformanceOption** (Option::Type type, Real underlying, Real moneyness, const Handle< YieldTermStructure > &dividendYield, const Handle< YieldTermStructure > &riskFreeRate, const Handle< BlackVolTermStructure > &volatility, const std::vector< Time > &times, BigNatural seed=0)

7.374 McPricer Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcpricer.hpp>`

7.374.1 Detailed Description

template<class MC, class S = Statistics> class QuantLib::McPricer< MC, S >

base class for Monte Carlo pricers

Eventually this class might be linked to the general tree of pricers, in order to have tools like impliedVolatility available. Also, it could, eventually, offer greeks methods. Deriving a class from McPricer gives an easy way to write a Monte Carlo Pricer. See McEuropean as example of one factor pricer, Basket as example of multi factor pricer.

Public Member Functions

- Real value (Real tolerance, Size maxSample=QL_MAX_INTEGER) const
  add samples until the required tolerance is reached
- Real valueWithSamples (Size samples) const
  simulate a fixed number of samples
- Real errorEstimate () const
  error Estimated of the samples simulated so far
- const S & sampleAccumulator (void) const
  access to the sample accumulator for more statistics

Protected Attributes

- boost::shared_ptr< MonteCarloModel< MC, S > > **mcModel_**

Static Protected Attributes

- static const Size **minSample_** = 1023

7.375 McSimulation Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/mcsimulation.hpp>`

7.375.1 Detailed Description

template<class MC, class S = Statistics> class QuantLib::McSimulation< MC, S >

base class for Monte Carlo engines

Eventually this class might offer greeks methods. Deriving a class from McEngine gives an easy way to write a Monte Carlo engine.

See McVanillaEngine as an example of one factor engine.

Public Types

- typedef MonteCarloModel< MC, S >::path_generator_type **path_generator_type**
- typedef MonteCarloModel< MC, S >::path_pricer_type **path_pricer_type**
- typedef MonteCarloModel< MC, S >::stats_type **stats_type**

Public Member Functions

- Real value (Real tolerance, Size maxSample=QL_MAX_INTEGER) const
  add samples until the required absolute tolerance is reached
- Real valueWithSamples (Size samples) const
  simulate a fixed number of samples
- Real errorEstimate () const
  error estimated using the samples simulated so far
- const stats_type & sampleAccumulator (void) const
  access to the sample accumulator for richer statistics
- void calculate (Real requiredTolerance, Size requiredSamples, Size maxSamples) const
  basic calculate method provided to inherited pricing engines

Protected Member Functions

- **McSimulation** (bool antitheticVariate, bool controlVariate)
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const =0
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const =0
- virtual TimeGrid **timeGrid** () const =0
- virtual boost::shared_ptr< path_pricer_type > **controlPathPricer** () const
- virtual boost::shared_ptr< PricingEngine > **controlPricingEngine** () const
- virtual Real **controlVariateValue** () const

Protected Attributes

- boost::shared_ptr< MonteCarloModel< MC, S > > **mcModel_**
- bool **antitheticVariate_**
- bool **controlVariate_**

Static Protected Attributes

- static const Size **minSample_** = 1023

7.375.2 Member Function Documentation

7.375.2.1 void calculate (Real *requiredTolerance*, Size *requiredSamples*, Size *maxSamples*) const

basic calculate method provided to inherited pricing engines

Initialize the one-factor Monte Carlo

7.376 MCVanillaEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mcvanillaengine.hpp>`

Inheritance diagram for MCVanillaEngine:

7.376.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCVanillaEngine< RNG, S >

Pricing engine for vanilla options using Monte Carlo simulation.

Public Member Functions

- void **calculate** () const

Protected Types

- typedef McSimulation< SingleVariate< RNG >, S >::path_generator_type **path_generator_type**
- typedef McSimulation< SingleVariate< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< SingleVariate< RNG >, S >::stats_type **stats_type**

Protected Member Functions

- **MCVanillaEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- TimeGrid **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- Real **controlVariateValue** () const

Protected Attributes

- Size **timeSteps_**
- Size **timeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

7.377 MersenneTwisterUniformRng Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/mt19937uniformrng.hpp>`

7.377.1 Detailed Description

Uniform random number generator.

Mersenne Twister random number generator of period 2**19937-1

For more details see http://www.math.keio.ac.jp/matumoto/emt.html

Test
the correctness of the returned values is tested by checking them against known good results.

Public Types

- typedef Sample< Real > **sample_type**

Public Member Functions

- MersenneTwisterUniformRng (unsigned long seed=0)
- **MersenneTwisterUniformRng** (const std::vector< unsigned long > &seeds)
- sample_type next () const
- unsigned long nextInt32 () const
  return a random number on [0,0xffffffff]-interval

7.377.2 Constructor & Destructor Documentation

7.377.2.1 MersenneTwisterUniformRng (unsigned long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.377.3 Member Function Documentation

7.377.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

7.378 Merton76Process Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/merton76process.hpp>`

Inheritance diagram for Merton76Process:

7.378.1 Detailed Description

Merton-76 jump-diffusion process.

Public Member Functions

- **Merton76Process** (const Handle< Quote > &stateVariable, const Handle< YieldTermStructure > &dividendTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< BlackVolTermStructure > &blackVolTS, const Handle< Quote > &jumpInt, const Handle< Quote > &logJMean, const Handle< Quote > &logJVol, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new EulerDiscretization))
- Time time (const Date &) const

StochasticProcess1D interface

- Real x0 () const
  returns the initial value of the state variable
- Real drift (Time, Real) const
  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real diffusion (Time, Real) const
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Real apply (Real x0, Real dx) const

Inspectors

- const boost::shared_ptr< Quote > & **stateVariable** () const
- const boost::shared_ptr< YieldTermStructure > & **dividendYield** () const
- const boost::shared_ptr< YieldTermStructure > & **riskFreeRate** () const
- const boost::shared_ptr< BlackVolTermStructure > & **blackVolatility** () const
- const boost::shared_ptr< Quote > & **jumpIntensity** () const
- const boost::shared_ptr< Quote > & **logMeanJump** () const
- const boost::shared_ptr< Quote > & **logJumpVolatility** () const

7.378.2 Member Function Documentation

7.378.2.1 Real apply (Real *x0*, Real *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from StochasticProcess1D.

7.378.2.2 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from StochasticProcess.

7.379 MixedScheme Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/mixedscheme.hpp>`

Inheritance diagram for MixedScheme:

7.379.1 Detailed Description

template<class Operator> class QuantLib::MixedScheme< Operator >

Mixed (explicit/implicit) scheme for finite difference methods.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type applyTo(const array_type&);
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:
The differential operator must be linear for this evolver to work.

Todo

- derive variable theta schemes
- introduce multi time-level schemes.

Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **MixedScheme** (const operator_type &L, Real theta, const bc_set &bcs)
- void **step** (array_type &a, Time t)
- void **setStep** (Time dt)

Protected Attributes

- operator_type **L_**
- operator_type **I_**
- operator_type **explicitPart_**
- operator_type **implicitPart_**
- Time **dt_**
- Real **theta_**
- bc_set **bcs_**

7.380 Money Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/money.hpp>`

7.380.1 Detailed Description

amount of cash

Test
money arithmetic is tested with and without currency conversions.

Conversion settings

These parameters are used for combining money amounts in different currencies

- enum ConversionType { NoConversion, BaseCurrencyConversion, AutomatedConversion }
- static ConversionType **conversionType**
- static Currency **baseCurrency**

Public Member Functions

Constructors

- **Money** (const Currency &currency, Decimal value)
- **Money** (Decimal value, const Currency &currency)

Inspectors

- const Currency & **currency** () const
- Decimal **value** () const
- Money **rounded** () const

Money arithmetics

See below for non-member functions and for settings which determine the behavior of the operators.

- Money **operator+** () const
- Money **operator-** () const
- Money & **operator+=** (const Money &)
- Money & **operator-=** (const Money &)
- Money & **operator** *= (Decimal)
- Money & **operator/=** (Decimal)

Related Functions

(Note that these are not member functions.)

- Money **operator+** (const Money &, const Money &)
- Money **operator-** (const Money &, const Money &)
- Money **operator** * (const Money &, Decimal)

- Money **operator** * (Decimal, const Money &)
- Money **operator/** (const Money &, Decimal)
- Decimal **operator/** (const Money &, const Money &)
- bool **operator==** (const Money &, const Money &)
- bool **operator!=** (const Money &, const Money &)
- bool **operator**< (const Money &, const Money &)
- bool **operator<=** (const Money &, const Money &)
- bool **operator**> (const Money &, const Money &)
- bool **operator>=** (const Money &, const Money &)
- bool **close** (const Money &, const Money &, Size n=42)
- bool **close_enough** (const Money &, const Money &, Size n=42)
- std::ostream & **operator**<< (std::ostream &, const Money &)

7.380.2 Member Enumeration Documentation

7.380.2.1 enum ConversionType

Enumerator:

NoConversion do not perform conversions

BaseCurrencyConversion convert both operands to the base currency before converting

AutomatedConversion return the result in the currency of the first operand

7.381 MonotonicCubicSpline Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp>`

Inheritance diagram for MonotonicCubicSpline:

7.381.1 Detailed Description

Cubic spline with monotonicity constraint

Public Member Functions

- template<class I1, class I2> MonotonicCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, CubicSpline::BoundaryCondition leftCondition, Real leftConditionValue, CubicSpline::BoundaryCondition rightCondition, Real rightConditionValue)

7.381.2 Constructor & Destructor Documentation

7.381.2.1 MonotonicCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, CubicSpline::BoundaryCondition *leftCondition*, Real *leftConditionValue*, CubicSpline::BoundaryCondition *rightCondition*, Real *rightConditionValue*)

Precondition:
the x values must be sorted.

7.382 MonteCarloModel Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Monte-Carlo/montecarlomodel.hpp>`

7.382.1 Detailed Description

template<class mc_traits, class stats_traits = Statistics> class QuantLib::MonteCarloModel< mc_traits, stats_traits >

General purpose Monte Carlo model for path samples.

The template arguments of this class correspond to available policies for the particular model to be instantiated—i.e., whether it is single- or multi-asset, or whether it should use pseudo-random or low-discrepancy numbers for path generation. Such decisions are grouped in trait classes so as to be orthogonal—see mctraits.hpp for examples.

The constructor accepts two safe references, i.e. two smart pointers, one to a path generator and the other to a path pricer. In case of control variate technique the user should provide the additional control option, namely the option path pricer and the option value.

Public Types

- typedef mc_traits::rsg_type **rsg_type**
- typedef mc_traits::path_generator_type **path_generator_type**
- typedef mc_traits::path_pricer_type **path_pricer_type**
- typedef path_generator_type::sample_type **sample_type**
- typedef path_pricer_type::result_type **result_type**
- typedef stats_traits **stats_type**

Public Member Functions

- **MonteCarloModel** (const boost::shared_ptr< path_generator_type > &pathGenerator, const boost::shared_ptr< path_pricer_type > &pathPricer, const stats_type &sample-Accumulator, bool antitheticVariate, const boost::shared_ptr< path_pricer_type > &cvPath-Pricer=boost::shared_ptr< path_pricer_type >(), result_type cvOptionValue=result_type())

- void **addSamples** (Size samples)
- const stats_type & **sampleAccumulator** (void) const

7.383 MoreGreeks Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/option.hpp>`

Inheritance diagram for MoreGreeks:

7.383.1 Detailed Description

more additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- Real **itmCashProbability**
- Real **deltaForward**
- Real **elasticity**
- Real **thetaPerDay**
- Real **strikeSensitivity**

7.384 MoroInverseCumulativeNormal Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/normaldistribution.hpp>`

7.384.1 Detailed Description

Moro Inverse cumulative normal distribution class.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It uses Beasly and Springer approximation, with an improved approximation for the tails. See Boris Moro, "The Full Monte", 1995, Risk Magazine.

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Peter J. Acklam's approximation is better and is available as QuantLib::InverseCumulativeNormal

Public Member Functions

- **MoroInverseCumulativeNormal** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const

7.385 MTLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for MTLCurrency:

7.385.1 Detailed Description

Maltese lira.

The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

7.386 MultiAsset Struct Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctraits.hpp>`

7.386.1 Detailed Description

template<class rng_traits = PseudoRandom> struct QuantLib::MultiAsset< rng_traits >

Deprecated
use MultiVariate instead

Public Types

- typedef MultiVariate< rng_traits >::path_type **path_type**
- typedef MultiVariate< rng_traits >::path_pricer_type **path_pricer_type**
- typedef MultiVariate< rng_traits >::rsg_type **rsg_type**
- typedef MultiVariate< rng_traits >::path_generator_type **path_generator_type**

7.387 MultiAssetOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/multiassetoption.hpp>`

Inheritance diagram for MultiAssetOption:

7.387.1 Detailed Description

Base class for options on multiple assets.

Public Member Functions

- **MultiAssetOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< Payoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

Instrument interface

- bool isExpired () const

  returns whether the instrument is still tradable.

greeks

- Real **delta** () const
- Real **gamma** () const
- Real **theta** () const
- Real **vega** () const
- Real **rho** () const
- Real **dividendRho** () const

Protected Member Functions

- void setupExpired () const
- void performCalculations () const

Protected Attributes

- Real **delta_**
- Real **gamma_**
- Real **theta_**
- Real **vega_**
- Real **rho_**
- Real **dividendRho_**
- boost::shared_ptr< StochasticProcess > **stochasticProcess_**

Classes

- class arguments

  Arguments for multi-asset option calculation

- class results

  Results from multi-asset option calculation

7.387.2 Member Function Documentation

7.387.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from Instrument.

Reimplemented in BasketOption.

7.387.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from Instrument.

7.387.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from Instrument.

7.388 MultiAssetOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/multiassetoption.hpp>`

Inheritance diagram for MultiAssetOption::arguments:

7.388.1 Detailed Description

Arguments for multi-asset option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< StochasticProcess > **stochasticProcess**

7.389 MultiAssetOption::results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/multiassetoption.hpp>`

Inheritance diagram for MultiAssetOption::results:

7.389.1 Detailed Description

Results from multi-asset option calculation

Public Member Functions

- void **reset** ()

7.390 MultiCubicSpline Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/multicubicspline.hpp>`

7.390.1 Detailed Description

template<Size i> class QuantLib::MultiCubicSpline< i >

Test

interpolated values are checked against the original function.

Todo

- fix it for Borland compilation
- allow extrapolation as for the other interpolations
- investigate if and how to implement Hyman filters and different boundary conditions

Bug

- cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- it does not compile under Borland

Public Types

- typedef c_splint::argument_type **argument_type**
- typedef c_splint::result_type **result_type**
- typedef c_splint::data_table **data_table**
- typedef c_splint::return_type **return_type**
- typedef c_splint::output_data **output_data**
- typedef c_splint::dimensions **dimensions**
- typedef c_splint::data **data**

Public Member Functions

- **MultiCubicSpline** (const SplineGrid &grid, const data_table &y, const std::vector< bool > &ae=std::vector< bool >(20, false))
- result_type **operator()** (const argument_type &x) const
- void **set_shared_increments** () const
- void **set_shared_coefficients** (const argument_type &x) const

7.391 MultiPath Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/multipath.hpp>`

7.391.1 Detailed Description

Correlated multiple asset paths.

MultiPath contains the list of paths for each asset, i.e., multipath[j] is the path followed by the j-th asset.

Public Member Functions

- **MultiPath** (Size nAsset, const TimeGrid &timeGrid)
- **MultiPath** (const std::vector< Path > &multiPath)

inspectors

- Size **assetNumber** () const
- Size **pathSize** () const

read/write access to components

- const Path & **operator[ ]** (Size j) const
- Path & **operator[ ]** (Size j)

7.392 MultiPathGenerator Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Monte-Carlo/multipathgenerator.hpp>`

7.392.1 Detailed Description

template<class GSG> class QuantLib::MultiPathGenerator< GSG >

Generates a multipath from a random number generator.

RSG is a sample generator which returns a random sequence. It must have the minimal interface:

```
RSG {
    Sample<Array> next();
};
```

Test
the generated paths are checked against cached results

Public Types

- typedef Sample< MultiPath > **sample_type**

Public Member Functions

- **MultiPathGenerator** (const boost::shared_ptr< StochasticProcess > &, const TimeGrid &, GSG generator, bool brownianBridge=false)
- const sample_type & **next** () const
- const sample_type & **antithetic** () const

7.393 MultiVariate Struct Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctraits.hpp>`

7.393.1 Detailed Description

template<class rng_traits = PseudoRandom> struct QuantLib::MultiVariate< rng_traits >

default Monte Carlo traits for multi-variate models

Public Types

- typedef MultiPath **path_type**
- typedef PathPricer< path_type > **path_pricer_type**
- typedef rng_traits::rsg_type **rsg_type**
- typedef MultiPathGenerator< rsg_type > **path_generator_type**

7.394 MXNCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for MXNCurrency:

7.394.1 Detailed Description

Mexican peso.

The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

7.395 NaturalCubicSpline Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp>`

Inheritance diagram for NaturalCubicSpline:

7.395.1 Detailed Description

Cubic spline with null second derivative at end points

Public Member Functions

- template<class I1, class I2> NaturalCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.395.2 Constructor & Destructor Documentation

7.395.2.1 NaturalCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:
the x values must be sorted.

7.396 NaturalMonotonicCubicSpline Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp>`

Inheritance diagram for NaturalMonotonicCubicSpline:

7.396.1 Detailed Description

Natural cubic spline with monotonicity constraint.

Public Member Functions

- template<class I1, class I2> NaturalMonotonicCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.396.2 Constructor & Destructor Documentation

7.396.2.1 NaturalMonotonicCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:
the *x* values must be sorted.

7.397 NeumannBC Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/boundarycondition.hpp>`

Inheritance diagram for NeumannBC:

7.397.1 Detailed Description

Neumann boundary condition (i.e., constant derivative).

Warning:
The value passed must not be the value of the derivative. Instead, it must be comprehensive of the grid step between the first two points–i.e., it must be the difference between f[0] and f[1].

Todo
generalize to time-dependent conditions.

Public Member Functions

- **NeumannBC** (Real value, Side side)
- void applyBeforeApplying (TridiagonalOperator &) const
- void applyAfterApplying (Array &) const
- void applyBeforeSolving (TridiagonalOperator &, Array &rhs) const
- void applyAfterSolving (Array &) const
- void setTime (Time)

7.397.2 Member Function Documentation

7.397.2.1 void applyBeforeApplying (TridiagonalOperator &) const [virtual]

This method modifies an operator L before it is applied to an array u so that $v = Lu$ will satisfy the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.397.2.2 void applyAfterApplying (Array &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.397.2.3 void applyBeforeSolving (TridiagonalOperator &, Array & *rhs*) const [virtual]

This method modifies an operator L before the linear system $Lu' = u$ is solved so that u' will satisfy the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.397.2.4 void applyAfterSolving (Array &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements BoundaryCondition< TridiagonalOperator >.

7.397.2.5 void setTime (Time) [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements BoundaryCondition< TridiagonalOperator >.

7.398 Newton Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/newton.hpp>`

Inheritance diagram for Newton:

7.398.1 Detailed Description

Newton 1-D solver

Note:
This solver requires that the passed function object implement a method `Real derivative(Real)`.

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.399 NewtonSafe Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/newtonsafe.hpp>`

Inheritance diagram for NewtonSafe:

7.399.1 Detailed Description

safe Newton 1-D solver

Note:
This solver requires that the passed function object implement a method `Real derivative(Real)`.

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.400 NLGCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for NLGCurrency:

7.400.1 Detailed Description

Dutch guilder.

The ISO three-letter code is NLG; the numeric code is 528. It is divided in 100 cents.

7.401 NoConstraint Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

Inheritance diagram for NoConstraint:

7.401.1 Detailed Description

No constraint.

7.402 NOKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for NOKCurrency:

7.402.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

7.403 NonLinearLeastSquare Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/leastsquare.hpp>`

7.403.1 Detailed Description

Non-linear least-square method.

Using a given optimization algorithm (default is conjugate gradient),

$$min\{r(x) : xinR^n\}$$

where $r(x) = |f(x)|^2$ is the Euclidean norm of $f(x)$ for some vector-valued function f from R^n to R^m ,

$$f = (f_1, ..., f_m)$$

with $f_i(x) = b_i - \phi(x, t_i)$ where b is the vector of target data and *phi* is a scalar function.

Assuming the differentiability of f , the gradient of r is defined by

$$gradr(x) = f'(x)^t.f(x)$$

Public Member Functions

- NonLinearLeastSquare (Constraint &c, Real accuracy=1e-4, Size maxiter=100)
  Default constructor.
- NonLinearLeastSquare (Constraint &c, Real accuracy, Size maxiter, boost::shared_ptr< OptimizationMethod > om)
  Default constructor.
- ~NonLinearLeastSquare ()
  Destructor.
- Array & perform (LeastSquareProblem &lsProblem)
  Solve least square problem using numerix solver.
- void **setInitialValue** (const Array &initialValue)
- Array & results ()
  return the results
- Real residualNorm ()
  return the least square residual norm
- Real lastValue ()
  return last function value
- Integer exitFlag ()
  return exit flag
- Integer iterationsNumber ()
  return the performed number of iterations

7.404 NormalDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/normaldistribution.hpp>`

7.404.1 Detailed Description

Normal distribution function.

Given x, it returns its probability in a Gaussian normal distribution. It provides the first derivative too.

Test

the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the CumulativeNormalDistribution and InverseCumulativeNormal classes.

Public Member Functions

- **NormalDistribution** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const
- Real **derivative** (Real x) const

7.405 NPRCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for NPRCurrency:

7.405.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

7.406 Null Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/null.hpp>`

7.406.1 Detailed Description

template<class Type> class QuantLib::Null< Type >

template class providing a null value for a given type.

Public Member Functions

- **operator Type** () const

7.407 NullCalendar Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/nullcalendar.hpp>`

Inheritance diagram for NullCalendar:

7.407.1 Detailed Description

Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

7.408 NullCondition Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/stepcondition.hpp>`

Inheritance diagram for NullCondition:

7.408.1 Detailed Description

template<class array_type> class QuantLib::NullCondition< array_type >

null step condition

Public Member Functions

- void **applyTo** (array_type &, Time) const

7.409 NullParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

Inheritance diagram for NullParameter:

7.409.1 Detailed Description

Parameter which is always zero $a(t) = 0$

7.410 NumericalMethod Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/numericalmethod.hpp>`

Inheritance diagram for NumericalMethod:

7.410.1 Detailed Description

Numerical method (tree, finite-differences) base class.

Public Member Functions

- **NumericalMethod** (const TimeGrid &timeGrid)
- virtual Disposable< Array > **grid** (Time) const =0

Inspectors

- const TimeGrid & **timeGrid** () const

Numerical method interface

These methods are to be used by discretized assets and must be overridden by developers implementing numerical methods. Users are advised to use the corresponding methods of DiscretizedAsset instead.

- virtual void initialize (DiscretizedAsset &, Time time) const =0

  initialize an asset at the given time.

- virtual void rollback (DiscretizedAsset &, Time to) const =0
- virtual void partialRollback (DiscretizedAsset &, Time to) const =0
- virtual Real presentValue (DiscretizedAsset &) const =0

  computes the present value of an asset.

Protected Attributes

- TimeGrid **t_**

7.410.2 Member Function Documentation

7.410.2.1 virtual void rollback (DiscretizedAsset &, Time *to*) const [pure virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implemented in Lattice, Lattice< OneFactorModel::ShortRateTree >, Lattice< TwoFactorModel::ShortRateTree >, and Lattice< BlackScholesLattice< T > >.

7.410.2.2 virtual void partialRollback (DiscretizedAsset &, Time *to*) const [pure virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

Warning:

In version 0.3.7 and earlier, this method was called rollAlmostBack method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);
asset->preAdjustValues();
```

Implemented in Lattice, Lattice< OneFactorModel::ShortRateTree >, Lattice< TwoFactorModel::ShortRateTree >, and Lattice< BlackScholesLattice< T > >.

7.411 NZDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/oceania.hpp>`

Inheritance diagram for NZDCurrency:

7.411.1 Detailed Description

New Zealand dollar.

The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

7.412 NZDLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/nzdlibor.hpp>`

Inheritance diagram for NZDLibor:

7.412.1 Detailed Description

NZD LIBOR rate

New Zealand Dollar LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Public Member Functions

- **NZDLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.413 Observable Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/observable.hpp>`

Inheritance diagram for Observable:

7.413.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- void notifyObservers ()

Friends

- class **Observer**

7.413.2 Member Function Documentation

7.413.2.1 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

7.414 ObservableValue Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/observablevalue.hpp>`

7.414.1 Detailed Description

template<class T> class QuantLib::ObservableValue< T >

observable and assignable proxy to concrete value

Observers can be registered with instances of this class so that they are notified when a different value is assigned to such instances. Client code can copy the contained value or pass it to functions via implicit conversion.

Note:
it is not possible to call non-const method on the returned value. This is by design, as this possibility would necessarily bypass the notification code; client code should modify the value via re-assignment instead.

Public Member Functions

- **ObservableValue** (const T &)
- operator T () const
  implicit conversion
- const T & value () const
  explicit inspector

controlled assignment

- ObservableValue< T > & **operator=** (const T &)
- ObservableValue< T > & **operator=** (const ObservableValue< T > &)

7.415 Observer Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/observable.hpp>`

Inheritance diagram for Observer:

7.415.1 Detailed Description

Object that gets notified when a given observable changes.

Public Member Functions

- **Observer** (const Observer &)
- Observer & **operator=** (const Observer &)
- template<class T> void **registerWith** (const boost::shared_ptr< T > &h)
- template<class T> void **unregisterWith** (const boost::shared_ptr< T > &h)
- virtual void update ()=0

7.415.2 Member Function Documentation

7.415.2.1 virtual void update () [pure virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in IndexedCoupon, ParCoupon, Link, Xibor, LazyObject, BlackModel, GenericModelEngine, LatticeShortRateModelEngine, BlackScholesProcess, DerivedQuote, CompositeQuote, CalibrationHelper, ShortRateModel, StochasticProcess, TermStructure, AffineTermStructure, ExtendedDiscountCurve, FlatForward, PiecewiseYieldCurve, RateHelper, CapVolatilityVector, GenericModelEngine< ShortRateModel, Arguments, Results >, GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >, GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >, GenericModelEngine< G2, Swaption::arguments, Swaption::results >, GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >, GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >, GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >, GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >, GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >, LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >, and LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >.

7.416 OneAssetOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetoption.hpp>`

Inheritance diagram for OneAssetOption:

7.416.1 Detailed Description

Base class for options on a single asset.

Public Member Functions

- **OneAssetOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< Payoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- Volatility impliedVolatility (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=QL_MIN_VOLATILITY, Volatility maxVol=QL_MAX_VOLATILITY) const
- void setupArguments (Arguments *) const

Instrument interface

- bool isExpired () const

  returns whether the instrument is still tradable.

greeks

- Real **delta** () const
- Real **deltaForward** () const
- Real **elasticity** () const
- Real **gamma** () const
- Real **theta** () const
- Real **thetaPerDay** () const
- Real **vega** () const
- Real **rho** () const
- Real **dividendRho** () const
- Real **itmCashProbability** () const

Protected Member Functions

- void setupExpired () const
- void performCalculations () const

Protected Attributes

- Real **delta_**
- Real **deltaForward_**
- Real **elasticity_**
- Real **gamma_**
- Real **theta_**

- Real **thetaPerDay_**
- Real **vega_**
- Real **rho_**
- Real **dividendRho_**
- Real **itmCashProbability_**
- boost::shared_ptr< StochasticProcess > **stochasticProcess_**

Classes

- class arguments

  Arguments for single-asset option calculation

- class results

  Results from single-asset option calculation

7.416.2 Member Function Documentation

7.416.2.1 Volatility impliedVolatility (Real *price*, Real *accuracy* = `1.0e-4`, Size *maxEvaluations* = `100`, Volatility *minVol* = `QL_MIN_VOLATILITY`, Volatility *maxVol* = `QL_MAX_VOLATILITY`) const

Warning:

currently, this method returns the Black-Scholes implied volatility. It will give unconsistent results if the pricing was performed with any other methods (such as jump-diffusion models.) options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases the calculation can fail and the result (if any) is almost meaningless. Another possible source of failure is to have a target value that is not attainable with any volatility, e.g., a target value lower than the intrinsic value in the case of American options.

7.416.2.2 void setupArguments (Arguments *) const `[virtual]`

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from Instrument.

Reimplemented in ContinuousAveragingAsianOption, DiscreteAveragingAsianOption, BarrierOption, CliquetOption, DividendVanillaOption, ForwardVanillaOption, OneAssetStrikedOption, QuantoForwardVanillaOption, and QuantoVanillaOption.

7.416.2.3 void setupExpired () const `[protected, virtual]`

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from Instrument.

Reimplemented in QuantoVanillaOption.

7.416.2.4 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from Instrument.

Reimplemented in BarrierOption, ForwardVanillaOption, OneAssetStrikedOption, and QuantoVanillaOption.

7.417 OneAssetOption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetoption.hpp>`

Inheritance diagram for OneAssetOption::arguments:

7.417.1 Detailed Description

Arguments for single-asset option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< StochasticProcess > **stochasticProcess**

7.418 OneAssetOption::results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetoption.hpp>`

Inheritance diagram for OneAssetOption::results:

7.418.1 Detailed Description

Results from single-asset option calculation

Public Member Functions

- void **reset** ()

7.419 OneAssetStrikedOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetstrikedoption.hpp>`

Inheritance diagram for OneAssetStrikedOption:

7.419.1 Detailed Description

Base class for options on a single asset with striked payoff.

Public Member Functions

- **OneAssetStrikedOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void setupArguments (Arguments *) const

greeks

- Real **strikeSensitivity** () const

Protected Member Functions

- void performCalculations () const

Protected Attributes

- Real **strikeSensitivity_**

7.419.2 Member Function Documentation

7.419.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetOption.

Reimplemented in ContinuousAveragingAsianOption, DiscreteAveragingAsianOption, BarrierOption, CliquetOption, DividendVanillaOption, ForwardVanillaOption, QuantoForwardVanillaOption, and QuantoVanillaOption.

7.419.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from OneAssetOption.

Reimplemented in BarrierOption, ForwardVanillaOption, and QuantoVanillaOption.

7.420 OneDayCounter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/one.hpp>`

Inheritance diagram for OneDayCounter:

7.420.1 Detailed Description

1/1 day count convention

7.421 OneFactorAffineModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/onefactormodel.hpp>`

Inheritance diagram for OneFactorAffineModel:

7.421.1 Detailed Description

Single-factor affine base class.

Single-factor models with an analytical formula for discount bonds should inherit from this class. They must then implement the functions $A(t,T)$ and $B(t,T)$ such that

$$P(t,T,r_t) = A(t,T)e^{-B(t,T)r_t}.$$

Public Member Functions

- **OneFactorAffineModel** (Size nArguments)
- Real **discountBond** (Time now, Time maturity, Rate rate) const
- DiscountFactor discount (Time t) const

  Implied discount curve.

Protected Member Functions

- virtual Real **A** (Time t, Time T) const =0
- virtual Real **B** (Time t, Time T) const =0

7.422 OneFactorModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/onefactormodel.hpp>`

Inheritance diagram for OneFactorModel:

7.422.1 Detailed Description

Single-factor short-rate model abstract class.

Public Member Functions

- **OneFactorModel** (Size nArguments)
- virtual boost::shared_ptr< ShortRateDynamics > dynamics () const =0

  returns the short-rate dynamics

- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Return by default a trinomial recombining tree.

Classes

- class ShortRateDynamics

  Base class describing the short-rate dynamics.

- class ShortRateTree

  Recombining trinomial tree discretizing the state variable.

7.423 OneFactorModel::ShortRateDynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/onefactormodel.hpp>`

7.423.1 Detailed Description

Base class describing the short-rate dynamics.

Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< StochasticProcess1D > &process)
- virtual Real variable (Time t, Rate r) const =0

  Compute state variable from short rate.

- virtual Rate shortRate (Time t, Real variable) const =0

  Compute short rate from state variable.

- const boost::shared_ptr< StochasticProcess1D > & process ()

  Returns the risk-neutral dynamics of the state variable.

7.424 OneFactorModel::ShortRateTree Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/onefactormodel.hpp>`

Inheritance diagram for OneFactorModel::ShortRateTree:

7.424.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

Public Member Functions

- ShortRateTree (const boost::shared_ptr< TrinomialTree > &tree, const boost::shared_ptr< ShortRateDynamics > &dynamics, const TimeGrid &timeGrid)

  Plain tree build-up from short-rate dynamics.

- ShortRateTree (const boost::shared_ptr< TrinomialTree > &tree, const boost::shared_ptr< ShortRateDynamics > &dynamics, const boost::shared_ptr< TermStructureFitting-Parameter::NumericalImpl > &phi, const TimeGrid &timeGrid)

  Tree build-up + numerical fitting to term-structure.

- Size **size** (Size i) const
- DiscountFactor **discount** (Size i, Size index) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

7.425 OneFactorOperator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/onefactoroperator.hpp>`

Inheritance diagram for OneFactorOperator:

7.425.1 Detailed Description

Interest-rate single factor model differential operator.

Public Member Functions

- **OneFactorOperator** (const Array &grid, const boost::shared_ptr< OneFactorModel::ShortRateDynamics > &)

7.426 OptimizationMethod Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/method.hpp>`

Inheritance diagram for OptimizationMethod:

7.426.1 Detailed Description

Abstract class for constrained optimization method.

Public Member Functions

- void setInitialValue (const Array &initialValue)

  Set initial value.

- void setEndCriteria (const EndCriteria &endCriteria)

  Set optimization end criteria.

- Integer & iterationNumber () const

  current iteration number

- EndCriteria & endCriteria () const

  optimization end criteria

- Integer & functionEvaluation () const

  number of evaluation of cost function

- Integer & gradientEvaluation () const

  number of evaluation of cost function gradient

- Real & functionValue () const

  value of cost function

- Real & gradientNormValue () const

  value of cost function gradient norm

- Array & x () const

  current value of the local minimum

- Array & searchDirection () const

  current value of the search direction

- virtual void minimize (const Problem &P) const =0

  minimize the optimization problem P

Protected Attributes

- Array initialValue_
  initial value of unknowns
- Integer iterationNumber_
  current iteration step in the Optimization process
- EndCriteria endCriteria_
  optimization end criteria
- Integer functionEvaluation_
  number of evaluation of cost function and its gradient
- Integer **gradientEvaluation_**
- Real functionValue_
  function and gradient norm values of the last step
- Real **squaredNorm_**
- Array x_
  current values of the local minimum and the search direction
- Array **searchDirection_**

7.427 Option Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/option.hpp>`

Inheritance diagram for Option:

7.427.1 Detailed Description

base option class

Public Types

- enum **Type** { **Call**, **Put** }

Public Member Functions

- **Option** (const boost::shared_ptr< Payoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())

Protected Attributes

- boost::shared_ptr< Payoff > **payoff_**
- boost::shared_ptr< Exercise > **exercise_**

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &, Option::Type)

Classes

- class arguments

7.428 Option::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/option.hpp>`

Inheritance diagram for Option::arguments:

7.428.1 Detailed Description

basic option arguments

Todo

- remove std::vector<Time> stoppingTimes
- how to handle strike-less option (asian average strike, forward, etc.)?

Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< Payoff > **payoff**
- boost::shared_ptr< Exercise > **exercise**
- std::vector< Time > **stoppingTimes**

7.429 OrnsteinUhlenbeckProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/ornsteinuhlenbeckprocess.hpp>`

Inheritance diagram for OrnsteinUhlenbeckProcess:

7.429.1 Detailed Description

Ornstein-Uhlenbeck process class.

This class describes the Ornstein-Uhlenbeck process governed by

$$dx = -ax_t dt + \sigma dW_t.$$

Public Member Functions

- **OrnsteinUhlenbeckProcess** (Real speed, Volatility vol, Real x0=0.0)

StochasticProcess interface

- Real x0 () const
  returns the initial value of the state variable
- Real drift (Time t, Real x) const
  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real diffusion (Time t, Real x) const
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Real expectation (Time t0, Real x0, Time dt) const
- Real stdDeviation (Time t0, Real x0, Time dt) const
- Real variance (Time t0, Real x0, Time dt) const

7.429.2 Member Function Documentation

7.429.2.1 Real expectation (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess1D.

7.429.2.2 Real stdDeviation (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess1D.

7.429.2.3 Real variance (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess1D.

7.430 Oslo Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/oslo.hpp>`

Inheritance diagram for Oslo:

7.430.1 Detailed Description

Oslo calendar

Holidays:

- Saturdays
- Sundays
- Holy Thursday
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- May Day, May 1st
- National Independence Day, May 17st
- Christmas, December 25th
- Boxing Day, December 26th

7.431 Parameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

Inheritance diagram for Parameter:

7.431.1 Detailed Description

Base class for model arguments.

Public Member Functions

- const Array & **params** () const
- void **setParam** (Size i, Real x)
- bool **testParams** (const Array &params) const
- Size **size** () const
- Real **operator()** (Time t) const
- const boost::shared_ptr< ParameterImpl > & **implementation** () const

Protected Member Functions

- **Parameter** (Size size, const boost::shared_ptr< ParameterImpl > &impl, const Constraint &constraint)

Protected Attributes

- Array **params_**
- Constraint **constraint_**

7.432 ParameterImpl Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

7.432.1 Detailed Description

Base class for model parameter implementation.

Public Member Functions

- virtual Real **value** (const Array &params, Time t) const =0

7.433 ParCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/parcoupon.hpp>`

Inheritance diagram for ParCoupon:

7.433.1 Detailed Description

coupon at par on a term structure

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ParCoupon** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Xibor > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())

CashFlow interface

- Real amount () const
  returns the amount of the cash flow

Coupon interface

- DayCounter dayCounter () const
  day counter for accrual calculation

FloatingRateCoupon interface

- Rate indexFixing () const
  fixing of the underlying index
- Date fixingDate () const
  fixing date

Inspectors

- const boost::shared_ptr< Xibor > & **index** () const

Observer interface

- void update ()

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.433.2 Member Function Documentation

7.433.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements CashFlow.

Reimplemented in Short< ParCoupon >.

7.433.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.434 Path Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/path.hpp>`

7.434.1 Detailed Description

single-factor random walk

Note:
the path includes the initial asset value as its first point.

iterators

- typedef Array::const_iterator **iterator**
- typedef Array::const_reverse_iterator **reverse_iterator**
- iterator **begin** () const
- iterator **end** () const
- reverse_iterator **rbegin** () const
- reverse_iterator **rend** () const

Public Member Functions

- **Path** (const TimeGrid &timeGrid, const Array &values=Array())

inspectors

- bool **empty** () const
- Size **length** () const
- Real operator[ ] (Size i) const

  asset value at the i -th point

- Real & **operator[ ]** (Size i)
- Real **value** (Size i) const
- Real & **value** (Size i)
- Time time (Size i) const

  time at the i -th point

- Real front () const

  initial asset value

- Real & **front** ()
- Real back () const

  final asset value

- Real & **back** ()
- const TimeGrid & timeGrid () const

  time grid

7.435 PathGenerator Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/pathgenerator.hpp>`

7.435.1 Detailed Description

template<class GSG> class QuantLib::PathGenerator< GSG >

Generates random paths using a sequence generator.

Generates random paths with drift(S,t) and variance(S,t) using a gaussian sequence generator

Test
the generated paths are checked against cached results

Public Types

- typedef Sample< Path > **sample_type**

Public Member Functions

- **PathGenerator** (const boost::shared_ptr< StochasticProcess1D > &, Time length, Size timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared_ptr< StochasticProcess1D > &, const TimeGrid &timeGrid, const GSG &generator, bool brownianBridge)

inspectors

- const sample_type & **next** () const
- const sample_type & **antithetic** () const
- Size **size** () const
- const TimeGrid & **timeGrid** () const

7.436 PathPricer Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/pathpricer.hpp>`

7.436.1 Detailed Description

template<class PathType, class ValueType = Real> class QuantLib::PathPricer< PathType, ValueType >

base class for path pricers

Returns the value of an option on a given path.

Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

7.437 Payoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/payoff.hpp>`

Inheritance diagram for Payoff:

7.437.1 Detailed Description

Base class for option payoffs.

Public Member Functions

- virtual Real **operator()** (Real price) const =0

7.438 PercentageStrikePayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for PercentageStrikePayoff:

7.438.1 Detailed Description

Payoff with strike expressed as percentage

Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, Real moneyness)
- Real **operator()** (Real price) const

7.439 Period Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/date.hpp>`

7.439.1 Detailed Description

Time period described by a number of a given time unit.

Public Member Functions

- **Period** (Integer n, TimeUnit units)
- Integer **length** () const
- TimeUnit **units** () const

Related Functions

(Note that these are not member functions.)

- Period **operator** * (Integer n, TimeUnit units)
- Period **operator** * (TimeUnit units, Integer n)
- bool **operator**< (const Period &, const Period &)
- bool **operator==** (const Period &, const Period &)
- bool **operator!=** (const Period &, const Period &)
- bool **operator**> (const Period &, const Period &)
- bool **operator**<= (const Period &, const Period &)
- bool **operator**>= (const Period &, const Period &)
- std::ostream & **operator**<< (std::ostream &, const Period &)

7.440 PiecewiseConstantParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

Inheritance diagram for PiecewiseConstantParameter:

7.440.1 Detailed Description

Piecewise-constant parameter.

$a(t) = a_i if t_{i-1} \geq t < t_i$. This kind of parameter is usually used to enhance the fitting of a model

Public Member Functions

- **PiecewiseConstantParameter** (const std::vector< Time > &times)

7.441 PiecewiseYieldCurve Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/piecewiseyieldcurve.hpp>`

Inheritance diagram for PiecewiseYieldCurve:

7.441.1 Detailed Description

template<class Traits, class Interpolator> class QuantLib::PiecewiseYieldCurve< Traits, Interpolator >

Piecewise yield term structure.

This term structure is bootstrapped on a number of interest rate instruments which are passed as a vector of handles to RateHelper instances. Their maturities mark the boundaries of the interpolated segments.

Each segment is determined sequentially starting from the earliest period to the latest and is chosen so that the instrument whose maturity marks the end of such segment is correctly repriced on the curve.

Warning:
The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

Test

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Public Member Functions

Constructors

- **PiecewiseYieldCurve** (const Date &referenceDate, const std::vector< boost::shared_ptr< RateHelper > > &instruments, const DayCounter &dayCounter, Real accuracy=1.0e-12, const Interpolator &i=Interpolator())
- **PiecewiseYieldCurve** (Integer settlementDays, const Calendar &calendar, const std::vector< boost::shared_ptr< RateHelper > > &instruments, const DayCounter &dayCounter, Real accuracy=1.0e-12, const Interpolator &i=Interpolator())

YieldTermStructure interface

- const std::vector< Date > & **dates** () const
- Date **maxDate** () const
- const std::vector< Time > & **times** () const
- Time **maxTime** () const

Observer interface

- void update ()

Friends

- class **ObjectiveFunction**

7.441.2 Member Function Documentation

7.441.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from LazyObject.

7.442 PKRCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for PKRCurrency:

7.442.1 Detailed Description

Pakistani rupee.

The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

7.443 PlainVanillaPayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for PlainVanillaPayoff:

7.443.1 Detailed Description

Plain-vanilla payoff.

Public Member Functions

- **PlainVanillaPayoff** (Option::Type type, Real strike)
- Real **operator()** (Real price) const

7.444 PLNCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for PLNCurrency:

7.444.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.

7.445 PoissonDistribution Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/poissondistribution.hpp>`

7.445.1 Detailed Description

Normal distribution function.

Given an integer k , it returns its probability in a Poisson distribution.

Test
the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **PoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

7.446 PositiveConstraint Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp>`

Inheritance diagram for PositiveConstraint:

7.446.1 Detailed Description

Constraint imposing positivity to all arguments

7.447 Prague Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/prague.hpp>`

Inheritance diagram for Prague:

7.447.1 Detailed Description

Prague calendar

Holidays (see http://www.pse.cz/):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Easter Monday
- Labour Day, May 1st
- Liberation Day, May 8th
- SS. Cyril and Methodius, July 5th
- Jan Hus Day, July 6th
- Czech Statehood Day, September 28th
- Independence Day, October 28th
- Struggle for Freedom and Democracy Day, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

7.448 PricingEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/pricingengine.hpp>`

Inheritance diagram for PricingEngine:

7.448.1 Detailed Description

interface for pricing engines

Public Member Functions

- virtual Arguments * **arguments** () const =0
- virtual const Results * **results** () const =0
- virtual void **reset** () const =0
- virtual void **calculate** () const =0

7.449 PrimeNumbers Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/primenumbers.hpp>`

7.449.1 Detailed Description

Prime numbers calculator.

Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

Static Public Member Functions

- static BigNatural get (Size absoluteIndex)

  Get and store one after another.

7.450 Problem Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/problem.hpp>`

7.450.1 Detailed Description

Constrained optimization problem.

Public Member Functions

- Problem (CostFunction &f, Constraint &c, OptimizationMethod &meth)
  default constructor
- Real value (const Array &x) const
  call cost function computation and increment evaluation counter
- void gradient (Array &grad_f, const Array &x) const
  call cost function gradient computation and increment
- Real valueAndGradient (Array &grad_f, const Array &x) const
  call cost function computation and it gradient
- OptimizationMethod & method () const
  Constrained optimization method.
- Constraint & constraint () const
  Constraint.
- CostFunction & costFunction () const
  Cost function.
- void minimize () const
  Minimization.
- Array & **minimumValue** () const

Protected Attributes

- CostFunction & costFunction_
  Unconstrained cost function.
- Constraint & constraint_
  Constraint.
- OptimizationMethod & method_
  constrained optimization method

7.451 PTECurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for PTECurrency:

7.451.1 Detailed Description

Portuguese escudo.

The ISO three-letter code is PTE; the numeric code is 620. It is divided in 100 centavos.

7.452 QuantoEngine Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Quanto/quantoengine.hpp>`

Inheritance diagram for QuantoEngine:

7.452.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::QuantoEngine< ArgumentsType, ResultsType >

Quanto engine base class.

Warning:
for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

Test

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **QuantoEngine** (const boost::shared_ptr< GenericEngine< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- ArgumentsType * underlyingArgs () const

Protected Attributes

- boost::shared_ptr< GenericEngine< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

7.452.2 Member Function Documentation

7.452.2.1 ArgumentsType* underlyingArgs () const

Access to the arguments of the underlying engine is needed as this engine is not able to set them completely. When necessary, it must be done by the instrument: see QuantoForwardVanillaOption for an example.

7.453 QuantoForwardVanillaOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/quantoforwardvanillaoption.hpp>`

Inheritance diagram for QuantoForwardVanillaOption:

7.453.1 Detailed Description

Quanto version of a forward vanilla option.

Public Types

- typedef QuantoOptionArguments< ForwardVanillaOption::arguments > **arguments**
- typedef QuantoOptionResults< ForwardVanillaOption::results > **results**
- typedef QuantoEngine< ForwardVanillaOption::arguments, ForwardVanillaOption::results > **engine**

Public Member Functions

- **QuantoForwardVanillaOption** (const Handle< YieldTermStructure > &foreignRiskFreeTS, const Handle< BlackVolTermStructure > &exchRateVolTS, const Handle< Quote > &correlation, Real moneyness, Date resetDate, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine)
- void setupArguments (Arguments *) const

7.453.2 Member Function Documentation

7.453.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from QuantoVanillaOption.

7.454 QuantoOptionArguments Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Quanto/quantoengine.hpp>`

7.454.1 Detailed Description

template<class ArgumentsType> class QuantLib::QuantoOptionArguments< ArgumentsType >

Arguments for quanto option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Real **correlation**
- Handle< YieldTermStructure > **foreignRiskFreeTS**
- Handle< BlackVolTermStructure > **exchRateVolTS**

7.455 QuantoOptionResults Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Quanto/quantoengine.hpp>`

7.455.1 Detailed Description

template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >

Results from quanto option calculation

Public Member Functions

- void **reset** ()

Public Attributes

- Real **qvega**
- Real **qrho**
- Real **qlambda**

7.456 QuantoTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/quantotermstructure.hpp>`

Inheritance diagram for QuantoTermStructure:

7.456.1 Detailed Description

Quanto term structure.

Quanto term structure for modelling quanto effect in option pricing.

Note:
This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **QuantoTermStructure** (const Handle< YieldTermStructure > &underlyingDividendTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< YieldTermStructure > &foreignRiskFreeTS, const Handle< BlackVolTermStructure > &underlyingBlackVolTS, Real strike, const Handle< BlackVolTermStructure > &exchRateBlackVolTS, Real exchRateATMlevel, Real underlyingExchRateCorrelation)

YieldTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Calendar calendar () const
  the calendar used for reference date calculation
- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- Date maxDate () const
  the latest date for which the curve can return rates

Protected Member Functions

- Rate zeroYieldImpl (Time) const
  returns the zero yield as seen from the evaluation date

7.457 QuantoVanillaOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/quantovanillaoption.hpp>`

Inheritance diagram for QuantoVanillaOption:

7.457.1 Detailed Description

quanto version of a vanilla option

Public Types

- typedef QuantoOptionArguments< VanillaOption::arguments > **arguments**
- typedef QuantoOptionResults< VanillaOption::results > **results**
- typedef QuantoEngine< VanillaOption::arguments, VanillaOption::results > **engine**

Public Member Functions

- **QuantoVanillaOption** (const Handle< YieldTermStructure > &foreignRiskFreeTS, const Handle< BlackVolTermStructure > &exchRateVolTS, const Handle< Quote > &correlation, const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &)
- void setupArguments (Arguments *) const

greeks

- Real **qvega** () const
- Real **qrho** () const
- Real **qlambda** () const

Protected Member Functions

- void setupExpired () const
- void performCalculations () const

Protected Attributes

- Handle< YieldTermStructure > **foreignRiskFreeTS_**
- Handle< BlackVolTermStructure > **exchRateVolTS_**
- Handle< Quote > **correlation_**
- Real **qvega_**
- Real **qrho_**
- Real **qlambda_**

7.457.2 Member Function Documentation

7.457.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from OneAssetStrikedOption.

Reimplemented in QuantoForwardVanillaOption.

7.457.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from OneAssetOption.

7.457.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from OneAssetStrikedOption.

7.458 Quote Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/quote.hpp>`

Inheritance diagram for Quote:

7.458.1 Detailed Description

purely virtual base class for market observables

Test
the observability of class instances is tested.

Public Member Functions

- virtual Real value () const =0
  returns the current value

7.459 RamdomizedLDS Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/randomizedlds.hpp>`

7.459.1 Detailed Description

template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniformRng>> class QuantLib::RamdomizedLDS< LDS, PRS >

Randomized (random shift) low-discrepancy sequence.

Random-shifts a uniform low-discrepancy sequence of dimension N by adding (modulo 1 for each coordinate) a pseudo-random uniform deviate in $(0,1)^N$. It is used for implementing Randomized Quasi Monte Carlo.

The uniform low discrepancy sequence is supplied by LDS; the uniform pseudo-random sequence is supplied by PRS.

Both class LDS and PRS must implement the following interface:

```
LDS::sample_type LDS::nextSequence() const;
Size LDS::dimension() const;
```

Precondition:
LDS and PRS must have the same dimension N

Warning:
Inverting LDS and PRS is possible, but it doesn't make sense

Todo
implement the other randomization algorithms

Test
correct initialization is tested.

Public Types

- typedef Sample< Array > **sample_type**

Public Member Functions

- **RamdomizedLDS** (const LDS &ldsg, const PRS &prsg)
- **RamdomizedLDS** (const LDS &ldsg)
- **RamdomizedLDS** (Size dimensionality, BigNatural ldsSeed=0, BigNatural prsSeed=0)
- const sample_type & nextSequence () const

  returns next sample using a given randomizing vector

- const sample_type & **lastSequence** () const
- void nextRandomizer ()
- Size **dimension** () const

7.459.2 Member Function Documentation

7.459.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

7.460 RandomSequenceGenerator Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/randomsequencegenerator.hpp>`

7.460.1 Detailed Description

template<class RNG> class QuantLib::RandomSequenceGenerator< RNG >

Random sequence generator based on a pseudo-random number generator.

Random sequence generator based on a pseudo-random number generator RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Warning:
do not use with low-discrepancy sequence generator

Public Types

- typedef Sample< Array > **sample_type**

Public Member Functions

- **RandomSequenceGenerator** (Size dimensionality, const RNG &rng)
- **RandomSequenceGenerator** (Size dimensionality, BigNatural seed=0)
- const sample_type & **nextSequence** () const
- std::vector< BigNatural > **nextInt32Sequence** () const
- const sample_type & **lastSequence** () const
- Size **dimension** () const

7.461 RateHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp>`

Inheritance diagram for RateHelper:

7.461.1 Detailed Description

Base class for rate helpers.

This class provides an abstraction for the instruments used to bootstrap a term structure. It is advised that a rate helper for an instrument contains an instance of the actual instrument class to ensure consistancy between the algorithms used during bootstrapping and later instrument pricing. This is not yet fully enforced in the available rate helpers, though - only SwapRateHelper contains a Swap instrument for the time being.

Public Member Functions

- **RateHelper** (const Handle< Quote > &quote)
- **RateHelper** (Real quote)

RateHelper interface

- Real **quoteError** () const
- Real **referenceQuote** () const
- virtual Real **impliedQuote** () const =0
- virtual DiscountFactor **discountGuess** () const
- virtual void setTermStructure (YieldTermStructure *)
  sets the term structure to be used for pricing
- virtual Date latestDate () const =0
  latest relevant date

Observer interface

- void update ()

Protected Attributes

- Handle< Quote > **quote_**
- YieldTermStructure * **termStructure_**

7.461.2 Member Function Documentation

7.461.2.1 virtual void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented in FixedCouponBondHelper, DepositRateHelper, FraRateHelper, and SwapRateHelper.

7.461.2.2 virtual Date latestDate () const [pure virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implemented in FixedCouponBondHelper, DepositRateHelper, FraRateHelper, FuturesRateHelper, and SwapRateHelper.

7.461.2.3 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.462 Results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/argsandresults.hpp>`

Inheritance diagram for Results:

7.462.1 Detailed Description

base class for generic result groups

Public Member Functions

- virtual void **reset** ()=0

7.463 Ridder Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/ridder.hpp>`

Inheritance diagram for Ridder:

7.463.1 Detailed Description

Ridder 1-D solver

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAcc) const

7.464 Riyadh Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/riyadh.hpp>`

Inheritance diagram for Riyadh:

7.464.1 Detailed Description

Riyadh calendar

Holidays:

- Fridays

Other holidays for which no rule is given (data available for 2004-2005 only:)

- EID AL-ADHA
- EID AL-FITR

7.465 ROLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for ROLCurrency:

7.465.1 Detailed Description

Romanian leu.

The ISO three-letter code is ROL; the numeric code is 642. It is divided in 100 bani.

7.466 Rounding Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for Rounding:

7.466.1 Detailed Description

basic rounding class

Test
the correctness of the returned values is tested by checking them against known good results.

Inspectors

- Integer **precision** () const
- Type **type** () const
- Integer **roundingDigit** () const

Public Types

- enum Type {
  None, Up, Down, Closest,
  Floor, Ceiling }
  rounding methods

Public Member Functions

- Rounding ()
  default constructor
- **Rounding** (Integer precision, Type type=Closest, Integer digit=5)
- Decimal operator() (Decimal value) const
  perform rounding

7.466.2 Member Enumeration Documentation

7.466.2.1 enum Type

rounding methods

The rounding methods follow the OMG specification available at `ftp://ftp.omg.org/pub/docs/formal/00-06-29.pdf`

Warning:
the names of the Floor and Ceiling methods might be misleading

Enumerator:

None do not round: return the number unmodified

Up the first decimal place past the precision will be rounded up. This differs from the OMG rule which rounds up only if the decimal to be rounded is greater than or equal to the rounding digit

Down all decimal places past the precision will be truncated

Closest the first decimal place past the precision will be rounded up if greater than or equal to the rounding digit; this corresponds to the OMG round-up rule. When the rounding digit is 5, the result will be the one closest to the original number, hence the name.

Floor positive numbers will be rounded up and negative numbers will be rounded down using the OMG round up and round down rules

Ceiling positive numbers will be rounded down and negative numbers will be rounded up using the OMG round up and round down rules

7.466.3 Constructor & Destructor Documentation

7.466.3.1 Rounding ()

default constructor

Instances built through this constructor don't perform any rounding.

7.467 SalvagingAlgorithm Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/pseudosqrt.hpp>`

7.467.1 Detailed Description

algorithm used for matricial pseudo square root

Public Types

- enum **Type** { **None**, **Spectral**, **Hypersphere** }

7.468 Sample Struct Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/sample.hpp>`

7.468.1 Detailed Description

template<class T> struct QuantLib::Sample< T >

weighted sample

Public Types

- typedef T **value_type**

Public Member Functions

- **Sample** (const T &value, Real weight)

Public Attributes

- T **value**
- Real **weight**

7.469 SampledCurve Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/sampledcurve.hpp>`

7.469.1 Detailed Description

This class contains a sampled curve.

Initially the class will contain one indexed curve

Public Member Functions

- **SampledCurve** (Size gridSize)
- void **setLogSpacing** (Real min, Real max)
- void **setLinearSpacing** (Real min, Real max)
- template<class F> void **sample** (F &f)
- void **setGrid** (const Array &g)
- void **setValues** (const Array &g)
- Array & **grid** ()
- Array & **values** ()
- Real & **value** (int i)

7.470 SARCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for SARCurrency:

7.470.1 Detailed Description

Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

7.471 Schedule Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/schedule.hpp>`

7.471.1 Detailed Description

Payment schedule.

Iterators

- typedef std::vector< Date >::const_iterator **const_iterator**
- const_iterator **begin** () const
- const_iterator **end** () const

Public Member Functions

- **Schedule** (const Calendar &calendar, const Date &startDate, const Date &endDate, Frequency frequency, BusinessDayConvention convention, const Date &stubDate=Date(), bool startFromEnd=false, bool longFinal=false)
- **Schedule** (const std::vector< Date > &, const Calendar &calendar=NullCalendar(), BusinessDayConvention convention=Unadjusted)

Date access

- Size **size** () const
- const Date & **operator[ ]** (Size i) const
- const Date & **date** (Size i) const
- const std::vector< Date > & **dates** () const
- bool **isRegular** (Size i) const

Other inspectors

- const Calendar & **calendar** () const
- const Date & **startDate** () const
- const Date & **endDate** () const
- Frequency **frequency** () const
- BusinessDayConvention **businessDayConvention** () const

7.472 Secant Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/secant.hpp>`

Inheritance diagram for Secant:

7.472.1 Detailed Description

Secant 1-D solver

Test
the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- template<class F> Real **solveImpl** (const F &f, Real xAccuracy) const

7.473 SeedGenerator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/seedgenerator.hpp>`

Inheritance diagram for SeedGenerator:

7.473.1 Detailed Description

Random seed generator.

Random number generator used for automatic generation of initialization seeds.

Test
correct initializaion of the single instance is tested.

Public Member Functions

- unsigned long **get** ()

Friends

- class **Singleton**< **SeedGenerator** >

7.474 SegmentIntegral Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/segmentintegral.hpp>`

7.474.1 Detailed Description

Integral of a one-dimensional function.

Given a number N of intervals, the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f \mathrm{d}x = \frac{1}{2} f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \frac{1}{2} f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b - a)/N$.

Test
the correctness of the result is tested by checking it against known good values.

Public Member Functions

- **SegmentIntegral** (Size intervals)
- template<class F> Real **operator()** (const F &f, Real a, Real b) const

7.475 SEKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for SEKCurrency:

7.475.1 Detailed Description

Swedish krona.

The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

7.476 Seoul Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/seoul.hpp>`

Inheritance diagram for Seoul:

7.476.1 Detailed Description

Seoul calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Independence Day, March 1st
- Arbour Day, April 5th
- Labor Day, May 1st
- Children's Day, May 5th
- Memorial Day, June 6th
- Constitution Day, July 17th
- Liberation Day, August 15th
- National Fondation Day, October 3th
- Christmas Day, December 25th

Other holidays for which no rule is given (data available for 2004-2006 only:)

- Lunar New Year
- Election Day 2004
- Buddha's birthday
- Harvest Moon Day

Data from http://www.kofex.com

7.477 SequenceStatistics Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/sequencestatistics.hpp>`

7.477.1 Detailed Description

template<class StatisticsType = Statistics> class QuantLib::SequenceStatistics< StatisticsType >

Statistics analysis of N-dimensional (sequence) data.

It provides 1-dimensional statistics as discrepancy plus N-dimensional (sequence) statistics (e.g. mean, variance, skewness, kurtosis, etc.) with one component for each dimension of the sample space.

For most of the statistics this class relies on the StatisticsType underlying class to provide 1-D methods that will be iterated for all the components of the N-D data. These lifted methods are the union of all the methods that might be requested to the 1-D underlying StatisticsType class, with the usual compile-time checks provided by the template approach.

Test
the correctness of the returned values is tested by checking them against numerical calculations.

Public Types

- typedef StatisticsType **statistics_type**

Public Member Functions

- **SequenceStatistics** (Size dimension)

inspectors

- Size **size** () const

covariance and correlation

- Disposable< Matrix > covariance () const
  returns the covariance Matrix
- Disposable< Matrix > correlation () const
  returns the correlation Matrix

1-D inspectors lifted from underlying statistics class

- Size **samples** () const
- Real **weightSum** () const

N-D inspectors lifted from underlying statistics class

- std::vector< Real > **mean** () const
- std::vector< Real > **variance** () const
- std::vector< Real > **standardDeviation** () const
- std::vector< Real > **downsideVariance** () const
- std::vector< Real > **downsideDeviation** () const
- std::vector< Real > **semiVariance** () const
- std::vector< Real > **semiDeviation** () const
- std::vector< Real > **errorEstimate** () const
- std::vector< Real > **skewness** () const
- std::vector< Real > **kurtosis** () const
- std::vector< Real > **min** () const
- std::vector< Real > **max** () const
- std::vector< Real > **gaussianPercentile** (Real y) const
- std::vector< Real > **percentile** (Real y) const
- std::vector< Real > **gaussianPotentialUpside** (Real percentile) const
- std::vector< Real > **potentialUpside** (Real percentile) const
- std::vector< Real > **gaussianValueAtRisk** (Real percentile) const
- std::vector< Real > **valueAtRisk** (Real percentile) const
- std::vector< Real > **gaussianExpectedShortfall** (Real percentile) const
- std::vector< Real > **expectedShortfall** (Real percentile) const
- std::vector< Real > **regret** (Real target) const
- std::vector< Real > **gaussianShortfall** (Real target) const
- std::vector< Real > **shortfall** (Real target) const
- std::vector< Real > **gaussianAverageShortfall** (Real target) const
- std::vector< Real > **averageShortfall** (Real target) const

Modifiers

- void **reset** (Size dimension=0)
- template<class Sequence> void **add** (const Sequence &sample, Real weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, Real weight=1.0)

Protected Attributes

- Size **dimension_**
- std::vector< statistics_type > **stats_**
- std::vector< Real > **results_**
- Matrix **quadraticSum_**

7.478 Settings Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/settings.hpp>`

Inheritance diagram for Settings:

7.478.1 Detailed Description

global repository for run-time library settings

Public Member Functions

- DateProxy & evaluationDate ()
  the date at which pricing is to be performed.
- const DateProxy & **evaluationDate** () const

Friends

- class **Singleton**< **Settings** >
- std::ostream & **operator**<< (std::ostream &, const DateProxy &)

7.478.2 Member Function Documentation

7.478.2.1 Settings::DateProxy & evaluationDate ()

the date at which pricing is to be performed.

Client code can inspect the evaluation date, as in:

```
Date d = Settings::instance().evaluationDate();
```

where today's date is returned if the evaluation date is set to the null date (its default value;) can set it to a new value, as in:

```
Settings::instance().evaluationDate() = d;
```

and can register with it, as in:

```
registerWith(Settings::instance().evaluationDate());
```

to be notified when it is set to a new value.

Warning:
a notification is not sent when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.

7.479 SGDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for SGDCurrency:

7.479.1 Detailed Description

Singapore dollar.

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

7.480 Short Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/shortindexedcoupon.hpp>`

7.480.1 Detailed Description

template<class IndexedCouponType> class QuantLib::Short< IndexedCouponType >

Short indexed coupon

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **Short** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Xibor > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())
- Real amount () const
  inhibit calculation

7.480.2 Member Function Documentation

7.480.2.1 Real amount () const

inhibit calculation

Unlike ParCoupon, this coupon can't calculate its fixing for future dates, either.

7.481 Short< ParCoupon > Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/shortfloatingcoupon.hpp>`

Inheritance diagram for Short< ParCoupon >:

7.481.1 Detailed Description

template<> class QuantLib::Short< ParCoupon >

Short coupon at par on a term structure

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **Short** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Xibor > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())
- Real amount () const

  throws when an interpolated fixing is needed

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.482 ShortRateModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/model.hpp>`

Inheritance diagram for ShortRateModel:

7.482.1 Detailed Description

Abstract short-rate model class.

Public Member Functions

- **ShortRateModel** (Size nArguments)
- void update ()
- virtual boost::shared_ptr< NumericalMethod > **tree** (const TimeGrid &) const =0
- void calibrate (const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod &method, const Constraint &constraint=Constraint())

  Calibrate to a set of market instruments (caps/swaptions).

- const boost::shared_ptr< Constraint > & **constraint** () const
- Disposable< Array > params () const

  Returns array of arguments on which calibration is done.

- void **setParams** (const Array &params)

Protected Member Functions

- virtual void **generateArguments** ()

Protected Attributes

- std::vector< Parameter > **arguments_**
- boost::shared_ptr< Constraint > **constraint_**

Friends

- class **CalibrationFunction**

7.482.2 Member Function Documentation

7.482.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.482.2.2 void calibrate (const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod & *method*, const Constraint & *constraint* = Constraint())

Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

7.483 ShoutCondition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/shoutcondition.hpp>`

Inheritance diagram for ShoutCondition:

7.483.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

Todo

unify the intrinsicValues/Payoff thing

Public Member Functions

- **ShoutCondition** (Option::Type type, Real strike, Time resTime, Rate rate)
- **ShoutCondition** (const Array &intrinsicValues, Time resTime, Rate rate)
- void **applyTo** (Array &a, Time t) const

7.484 SimpleCashFlow Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/simplecashflow.hpp>`

Inheritance diagram for SimpleCashFlow:

7.484.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **SimpleCashFlow** (Real amount, const Date &date)

CashFlow interface

- Real **amount** () const
  returns the amount of the cash flow
- Date **date** () const
  returns the date at which the cash flow is settled

Visitability

- virtual void **accept** (AcyclicVisitor &)

7.484.2 Member Function Documentation

7.484.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements CashFlow.

7.485 SimpleDayCounter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/simpledaycounter.hpp>`

Inheritance diagram for SimpleDayCounter:

7.485.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

This day counter tries to ensure that whole-month distances are returned as a simple fraction, i.e., 1 year = 1.0, 6 months = 0.5, 3 months = 0.25 and so forth.

Warning:
this day counter should be used together with NullCalendar, which ensures that dates at whole-month distances share the same day of month. It is **not** guaranteed to work with any other calendar.

Test
the correctness of the results is checked against known good values.

7.486 SimpleQuote Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/quote.hpp>`

Inheritance diagram for SimpleQuote:

7.486.1 Detailed Description

market element returning a stored value

Public Member Functions

- **SimpleQuote** (Real value)

Quote interface

- Real value () const
  returns the current value

Modifiers

- void **setValue** (Real value)

7.487 SimpleSwap Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/simpleswap.hpp>`

Inheritance diagram for SimpleSwap:

7.487.1 Detailed Description

Simple fixed-rate vs Libor swap.

Test

- the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- the correctness of the returned value is tested by checking it against a known good value.

Public Member Functions

- **SimpleSwap** (bool payFixedRate, Real nominal, const Schedule &fixedSchedule, Rate fixedRate, const DayCounter &fixedDayCount, const Schedule &floatSchedule, const boost::shared_ptr< Xibor > &index, Integer indexFixingDays, Spread spread, const Handle< YieldTermStructure > &termStructure)
- Rate **fairRate** () const
- Spread **fairSpread** () const
- Real **fixedLegBPS** () const
- Real **floatingLegBPS** () const
- Rate **fixedRate** () const
- Spread **spread** () const
- Real **nominal** () const
- bool **payFixedRate** () const
- const std::vector< boost::shared_ptr< CashFlow > > & **fixedLeg** () const
- const std::vector< boost::shared_ptr< CashFlow > > & **floatingLeg** () const
- void setupArguments (Arguments *args) const

Classes

- class arguments

  Arguments for simple swap calculation

- class results

  Results from simple swap calculation

7.487.2 Member Function Documentation

7.487.2.1 void setupArguments (Arguments * *args*) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from Instrument.

7.488 SimpleSwap::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/simpleswap.hpp>`

Inheritance diagram for SimpleSwap::arguments:

7.488.1 Detailed Description

Arguments for simple swap calculation

Public Member Functions

- void **validate** () const

Public Attributes

- bool **payFixed**
- Real **nominal**
- std::vector< Time > **fixedResetTimes**
- std::vector< Time > **fixedPayTimes**
- std::vector< Real > **fixedCoupons**
- std::vector< Time > **floatingAccrualTimes**
- std::vector< Time > **floatingResetTimes**
- std::vector< Time > **floatingFixingTimes**
- std::vector< Time > **floatingPayTimes**
- std::vector< Spread > **floatingSpreads**
- Real **currentFloatingCoupon**

7.489 SimpleSwap::results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/simpleswap.hpp>`

Inheritance diagram for SimpleSwap::results:

7.489.1 Detailed Description

Results from simple swap calculation

7.490 Simplex Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/simplex.hpp>`

Inheritance diagram for Simplex:

7.490.1 Detailed Description

Multi-dimensional simplex class.

Public Member Functions

- Simplex (Real lambda, Real tol)
- virtual void minimize (const Problem &P) const

  minimize the optimization problem P

7.490.2 Constructor & Destructor Documentation

7.490.2.1 Simplex (Real *lambda*, Real *tol*)

Constructor taking as input the characteristic length and tolerance

7.491 SimpsonIntegral Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/simpsonintegral.hpp>`

Inheritance diagram for SimpsonIntegral:

7.491.1 Detailed Description

Integral of a one-dimensional function.

Test
the correctness of the result is tested by checking it against known good values.

Public Member Functions

- **SimpsonIntegral** (Real accuracy, Size maxIterations=Null< Size >())
- template<class F> Real **operator()** (const F &f, Real a, Real b) const

7.492 Singapore Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/singapore.hpp>`

Inheritance diagram for Singapore:

7.492.1 Detailed Description

Singapore calendar

Holidays:

- Saturdays
- Sundays
- New Year's day, January 1st
- Good Friday
- Labour Day, May 1st
- National Day, August 9th
- Christmas, December 25th
- Boxing Day, December 26th

Other holidays for which no rule is given (data available for 2004-2005 only:)

- Chinese New Year
- Hari Raya Haji
- Vesak Poya Day
- Deepavali
- Diwali
- Hari Raya Puasa

Data from http://www.asx.com.au and http://www.ses.com.sg

7.493 SingleAsset Struct Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctraits.hpp>`

7.493.1 Detailed Description

template<class rng_traits = PseudoRandom> struct QuantLib::SingleAsset< rng_traits >

Deprecated
use SingleVariate instead

Public Types

- typedef SingleVariate< rng_traits >::path_type **path_type**
- typedef SingleVariate< rng_traits >::path_pricer_type **path_pricer_type**
- typedef SingleVariate< rng_traits >::rsg_type **rsg_type**
- typedef SingleVariate< rng_traits >::path_generator_type **path_generator_type**

7.494 SingleAssetOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/singleassetoption.hpp>`

Inheritance diagram for SingleAssetOption:

7.494.1 Detailed Description

Black-Scholes-Merton option.

Public Member Functions

- **SingleAssetOption** (Option::Type type, Real underlying, Real strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, Volatility volatility)
- virtual void **setVolatility** (Volatility newVolatility)
- virtual void **setRiskFreeRate** (Rate newRate)
- virtual void **setDividendYield** (Rate newDividendYield)
- virtual Real **value** () const =0
- virtual Real **delta** () const =0
- virtual Real **gamma** () const =0
- virtual Real **theta** () const
- virtual Real **vega** () const
- virtual Real **rho** () const
- virtual Real **dividendRho** () const
- Volatility **impliedVolatility** (Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100, Volatility minVol=QL_MIN_VOLATILITY, Volatility maxVol=QL_MAX_VOLATILITY) const
- Spread **impliedDivYield** (Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100, Spread minYield=QL_MIN_DIVYIELD, Spread maxYield=QL_MAX_DIVYIELD) const
- virtual boost::shared_ptr< SingleAssetOption > **clone** () const =0

Protected Attributes

- Real **underlying_**
- PlainVanillaPayoff **payoff_**
- Spread **dividendYield_**
- Rate **riskFreeRate_**
- Time **residualTime_**
- Volatility **volatility_**
- bool **hasBeenCalculated_**
- Real **rho_**
- Real **dividendRho_**
- Real **vega_**
- Real **theta_**
- bool **rhoComputed_**
- bool **dividendRhoComputed_**
- bool **vegaComputed_**
- bool **thetaComputed_**

Static Protected Attributes

- static const Real **dVolMultiplier_**
- static const Real **dRMultiplier_**

Friends

- class **VolatilityFunction**
- class **DivYieldFunction**

7.494.2 Member Function Documentation

7.494.2.1 Volatility impliedVolatility (Real *targetValue*, Real *accuracy* = `1e-4`, Size *maxEvaluations* = `100`, Volatility *minVol* = `QL_MIN_VOLATILITY`, Volatility *maxVol* = `QL_MAX_VOLATILITY`) const

Warning:

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases impliedVolatility can fail and in any case is meaningless. Another possible source of failure is to have a targetValue that is not attainable with any volatility, e.g. a targetValue lower than the intrinsic value in the case of American options.

7.495 Singleton Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/singleton.hpp>`

Inheritance diagram for Singleton:

7.495.1 Detailed Description

template<class T> class QuantLib::Singleton< T >

Basic support for the singleton pattern.

The typical use of this class is:

```
class Foo : public Singleton<Foo> {
    friend class Singleton<Foo>;
  private:
    Foo() {}
  public:
    ...
};
```

which, albeit sub-optimal, frees one from the concerns of creating and managing the unique instance and can serve later as a single implemementation point should synchronization features be added.

Static Public Member Functions

- static T & instance ()

  access to the unique instance

7.496 SingleVariate Struct Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctraits.hpp>`

7.496.1 Detailed Description

template<class rng_traits = PseudoRandom> struct QuantLib::SingleVariate< rng_traits >

default Monte Carlo traits for single-variate models

Public Types

- typedef Path **path_type**
- typedef PathPricer< path_type > **path_pricer_type**
- typedef rng_traits::rsg_type **rsg_type**
- typedef PathGenerator< rsg_type > **path_generator_type**

7.497 SITCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for SITCurrency:

7.497.1 Detailed Description

Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

7.498 SKKCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for SKKCurrency:

7.498.1 Detailed Description

Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

7.499 SobolRsg Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/sobolrsg.hpp>`

7.499.1 Detailed Description

Sobol low-discrepancy sequence generator.

A Gray code counter and bitwise operations are used for very fast sequence generation.

The implementation relies on primitive polynomials modulo two from the book "Monte Carlo Methods in Finance" by Peter Jäckel.

21 200 primitive polynomials modulo two are provided by default in QuantLib. Jäckel has calculated 8 129 334 polynomials, also available in a different file that can be downloaded from http://quantlib.org. If you need that many dimensions you must replace the default version of the primitivepolynomials.c file with the extended one.

The choice of initialization numbers (also know as free direction integers) is crucial for the homogeneity properties of the sequence. Sobol defines two homogeneity properties: Property A and Property A'.

The unit initialization numbers suggested in "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery (section 7.7) fail the test for Property A even for low dimensions.

Bratley and Fox published coefficients of the free direction integers up to dimension 40, crediting unpublished work of Sobol' and Levitan. See Bratley, P., Fox, B.L. (1988) "Algorithm 659: Implementing Sobol's quasirandom sequence generator," ACM Transactions on Mathematical Software 14:88-100. These values satisfy Property A for d<=20 and d = 23, 31, 33, 34, 37; Property A' holds for d<=6.

Jäckel provides in his book (section 8.3) initialization numbers up to dimension 32. Coefficients for d<=8 are the same as in Bradley-Fox, so Property A' holds for d<=6 but Property A holds for d<=32.

The implementation of Lemieux, Cieslak, and Luttmer includes coefficients of the free direction integers up to dimension 360. Coefficients for d<=40 are the same as in Bradley-Fox. For dimension 40<d<=360 the coefficients have been calculated as optimal values based on the "resolution" criterion. See "RandQMC user's guide - A package for randomized quasi-Monte Carlo methods in C," by C. Lemieux, M. Cieslak, and K. Luttmer, version January 13 2004, and references cited there (http://www.math.ucalgary.ca/~lemieux/randqmc.html). The values up to d<=360 has been provided to the QuantLib team by Christiane Lemieux, private communication, September 2004.

For more info on Sobol' sequences see also "Monte Carlo Methods in Financial Engineering," by P. Glasserman, 2004, Springer, section 5.2.3

Test

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- typedef Sample< Array > **sample_type**

- enum **DirectionIntegers** { **Unit**, **Jaeckel**, **SobolLevitan**, **SobolLevitanLemieux** }

Public Member Functions

- SobolRsg (Size dimensionality, unsigned long seed=0, DirectionIntegers directionIntegers=Jaeckel)
- const std::vector< unsigned long > & **nextInt32Sequence** () const
- const SobolRsg::sample_type & **nextSequence** () const
- const sample_type & **lastSequence** () const
- Size **dimension** () const

7.499.2 Constructor & Destructor Documentation

7.499.2.1 SobolRsg (Size *dimensionality*, unsigned long *seed* = 0, DirectionIntegers *directionIntegers* = Jaeckel)

Precondition:
dimensionality must be <= PPMT_MAX_DIM

7.500 Solver1D Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/solver1d.hpp>`

Inheritance diagram for Solver1D:

7.500.1 Detailed Description

template<class Impl> class QuantLib::Solver1D< Impl >

Base class for 1-D solvers.

The implementation of this class uses the so-called "Barton-Nackman trick", also known as "the curiously recurring template pattern". Concrete solvers will be declared as:

```
class Foo : public Solver1D<Foo> {
  public:
    ...
    template <class F>
    Real solveImpl(const F& f, Real accuracy) const {
        ...
    }
};
```

Before calling `solveImpl`, the base class will set its protected data members so that:

- `xMin_` and `xMax_` form a valid bracket;
- `fxMin_` and `fxMax_` contain the values of the function in `xMin_` and `xMax_`;
- `root_` is a valid initial guess. The implementation of `solveImpl` can safely assume all of the above.

Todo

- clean up the interface so that it is clear whether the accuracy is specified for x or $f(x)$.
- add target value (now the target value is 0.0)

Public Member Functions

Modifiers

- template<class F> Real solve (const F &f, Real accuracy, Real guess, Real step) const
- template<class F> Real solve (const F &f, Real accuracy, Real guess, Real xMin, Real xMax) const
- void setMaxEvaluations (Size evaluations)
- void setLowerBound (Real lowerBound)

  sets the lower bound for the function domain

- void setUpperBound (Real upperBound)

  sets the upper bound for the function domain

Protected Attributes

- Real **root_**
- Real **xMin_**
- Real **xMax_**
- Real **fxMin_**
- Real **fxMax_**
- Size **maxEvaluations_**
- Size **evaluationNumber_**

7.500.2 Member Function Documentation

7.500.2.1 Real solve (const F & *f*, Real *accuracy*, Real *guess*, Real *step*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| <$ *accuracy*). This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

7.500.2.2 Real solve (const F & *f*, Real *accuracy*, Real *guess*, Real *xMin*, Real *xMax*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| <$ *accuracy*). An initial guess must be supplied, as well as two values $x_{\min}$ and $x_{\max}$ which must bracket the zero (i.e., either $f(x_{\min}) \leq 0 \leq f(x_{\max})$, or $f(x_{\max}) \leq 0 \leq f(x_{\min})$ must be true).

7.500.2.3 void setMaxEvaluations (Size *evaluations*)

This method sets the maximum number of function evaluations for the bracketing routine. An error is thrown if a bracket is not found after this number of evaluations.

7.501 SquareRootProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/squarerootprocess.hpp>`

Inheritance diagram for SquareRootProcess:

7.501.1 Detailed Description

Square-root process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma \sqrt{x_t} dW_t.$$

Public Member Functions

- **SquareRootProcess** (Real b, Real a, Volatility sigma, Real x0=0.0, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new EulerDiscretization))

StochasticProcess interface

- Real x0 () const
  returns the initial value of the state variable
- Real drift (Time t, Real x) const
  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real diffusion (Time t, Real x) const
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

7.502 StatsHolder Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianstatistics.hpp>`

7.502.1 Detailed Description

Helper class for precomputed distributions.

Public Member Functions

- **StatsHolder** (Real mean, Real standardDeviation)
- Real **mean** () const
- Real **standardDeviation** () const

7.503 SteepestDescent Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/steepestdescent.hpp>`

Inheritance diagram for SteepestDescent:

7.503.1 Detailed Description

Multi-dimensional steepest-descent class.

User has to provide line-search method and optimization end criteria

search direction = $-f'(x)$

Public Member Functions

- SteepestDescent ()
  default default constructor (msvc bug)
- SteepestDescent (const boost::shared_ptr< LineSearch > &lineSearch)
  default constructor
- virtual ~SteepestDescent ()
  destructor
- virtual void minimize (const Problem &P) const
  minimize the optimization problem P

7.504 step_iterator Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/steppingiterator.hpp>`

7.504.1 Detailed Description

template<class Iterator> class QuantLib::step_iterator< Iterator >

Iterator advancing in constant steps.

This iterator advances an underlying random-access iterator in steps of n positions, where n is a positive integer given upon construction.

Public Member Functions

- **step_iterator** (const Iterator &base, Size step)
- template<class OtherIterator> **step_iterator** (const step_iterator< OtherIterator > &i, typename boost::enable_if_convertible< OtherIterator, Iterator >::type *=0)
- Size **step** () const
- void **increment** ()
- void **decrement** ()
- void **advance** (typename super_t::difference_type n)
- super_t::difference_type **distance_to** (const step_iterator &i) const

Related Functions

(Note that these are not member functions.)

- step_iterator< Iterator > make_step_iterator (Iterator it, Size step)
  helper function to create step iterators

7.505 StepCondition Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/stepcondition.hpp>`

Inheritance diagram for StepCondition:

7.505.1 Detailed Description

template<class array_type> class QuantLib::StepCondition< array_type >

condition to be applied at every time step

Public Member Functions

- virtual void **applyTo** (array_type &a, Time t) const =0

7.506 StepConditionSet Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/parallelevolver.hpp>`

7.506.1 Detailed Description

template<typename array_type> class QuantLib::StepConditionSet< array_type >

Parallel evolver for multiple arrays.

Public Member Functions

- void **applyTo** (std::vector< array_type > &a, Time t) const
- void **push_back** (const itemType &a)

7.507 StochasticProcess Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/stochasticprocess.hpp>`

Inheritance diagram for StochasticProcess:

7.507.1 Detailed Description

multi-dimensional stochastic process class.

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t) \cdot dW_t.$$

Public Member Functions

Stochastic process interface

- virtual Size size () const =0
  returns the number of dimensions of the stochastic process
- virtual Size factors () const
  returns the number of independent factors of the process
- virtual Disposable< Array > initialValues () const =0
  returns the initial values of the state variables
- virtual Disposable< Array > drift (Time t, const Array &x) const =0
  returns the drift part of the equation, i.e., $\mu(t, x_t)$
- virtual Disposable< Matrix > diffusion (Time t, const Array &x) const =0
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual Disposable< Array > expectation (Time t0, const Array &x0, Time dt) const
- virtual Disposable< Matrix > stdDeviation (Time t0, const Array &x0, Time dt) const
- virtual Disposable< Matrix > covariance (Time t0, const Array &x0, Time dt) const
- virtual Disposable< Array > evolve (Time t0, const Array &x0, Time dt, const Array &dw) const
- virtual Disposable< Array > apply (const Array &x0, const Array &dx) const

utilities

- virtual Time time (const Date &) const

Observer interface

- void update ()

Protected Member Functions

- **StochasticProcess** (const boost::shared_ptr< discretization > &)

Protected Attributes

- boost::shared_ptr< discretization > **discretization_**

Classes

- class discretization

  discretization of a stochastic process over a given time interval

7.507.2 Member Function Documentation

7.507.2.1 virtual Disposable<Array> expectation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in StochasticProcessArray.

7.507.2.2 virtual Disposable<Matrix> stdDeviation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in StochasticProcessArray.

7.507.2.3 virtual Disposable<Matrix> covariance (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in StochasticProcessArray.

7.507.2.4 virtual Disposable<Array> evolve (Time *t0*, const Array & *x0*, Time *dt*, const Array & *dw*) const [virtual]

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented in CapletLiborMarketModelProcess.

7.507.2.5 virtual Disposable<Array> apply (const Array & *x0*, const Array & *dx*) const [virtual]

applies a change to the asset value. By default, it returns x + Δx .

Reimplemented in CapletLiborMarketModelProcess, HestonProcess, and StochasticProcess-Array.

7.507.2.6 virtual Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented in BlackScholesProcess, HestonProcess, Merton76Process, and StochasticProcess-Array.

7.507.2.7 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

Reimplemented in BlackScholesProcess.

7.508 StochasticProcess1D Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/stochasticprocess.hpp>`

Inheritance diagram for StochasticProcess1D:

7.508.1 Detailed Description

1-dimensional stochastic process

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t)dW_t.$$

Public Member Functions

1-D stochastic process interface

- virtual Real x0 () const =0
  returns the initial value of the state variable
- virtual Real drift (Time t, Real x) const =0
  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- virtual Real diffusion (Time t, Real x) const =0
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual Real expectation (Time t0, Real x0, Time dt) const
- virtual Real stdDeviation (Time t0, Real x0, Time dt) const
- virtual Real variance (Time t0, Real x0, Time dt) const
- virtual Real evolve (Time t0, Real x0, Time dt, Real dw) const
- virtual Real apply (Real x0, Real dx) const

Protected Member Functions

- **StochasticProcess1D** (const boost::shared_ptr< discretization > &)

Protected Attributes

- boost::shared_ptr< discretization > **discretization_**

Classes

- class discretization
  discretization of a 1-D stochastic process

7.508.2 Member Function Documentation

7.508.2.1 virtual Real expectation (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in OrnsteinUhlenbeckProcess.

7.508.2.2 virtual Real stdDeviation (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in OrnsteinUhlenbeckProcess.

7.508.2.3 virtual Real variance (Time *t0*, Real *x0*, Time *dt*) const [virtual]

returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in OrnsteinUhlenbeckProcess.

7.508.2.4 virtual Real evolve (Time *t0*, Real *x0*, Time *dt*, Real *dw*) const [virtual]

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

7.508.2.5 virtual Real apply (Real *x0*, Real *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented in BlackScholesProcess, and Merton76Process.

7.509 StochasticProcess1D::discretization Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/stochasticprocess.hpp>`

Inheritance diagram for StochasticProcess1D::discretization:

7.509.1 Detailed Description

discretization of a 1-D stochastic process

Public Member Functions

- virtual Real **drift** (const StochasticProcess1D &, Time t0, Real x0, Time dt) const =0
- virtual Real **diffusion** (const StochasticProcess1D &, Time t0, Real x0, Time dt) const =0
- virtual Real **variance** (const StochasticProcess1D &, Time t0, Real x0, Time dt) const =0

7.510 StochasticProcess::discretization Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/stochasticprocess.hpp>`

Inheritance diagram for StochasticProcess::discretization:

7.510.1 Detailed Description

discretization of a stochastic process over a given time interval

Public Member Functions

- virtual Disposable< Array > **drift** (const StochasticProcess &, Time t0, const Array &x0, Time dt) const =0
- virtual Disposable< Matrix > **diffusion** (const StochasticProcess &, Time t0, const Array &x0, Time dt) const =0
- virtual Disposable< Matrix > **covariance** (const StochasticProcess &, Time t0, const Array &x0, Time dt) const =0

7.511 StochasticProcessArray Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/stochasticprocessarray.hpp>`

Inheritance diagram for StochasticProcessArray:

7.511.1 Detailed Description

Array of correlated 1-D stochastic processes.

Public Member Functions

- **StochasticProcessArray** (const std::vector< boost::shared_ptr< StochasticProcess1D > > &, const Matrix &correlation)
- Size size () const

  returns the number of dimensions of the stochastic process

- Disposable< Array > initialValues () const

  returns the initial values of the state variables

- Disposable< Array > drift (Time t, const Array &x) const

  returns the drift part of the equation, i.e., $\mu(t, x_t)$

- Disposable< Matrix > diffusion (Time t, const Array &x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

- Disposable< Array > expectation (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > stdDeviation (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > covariance (Time t0, const Array &x0, Time dt) const
- Disposable< Array > apply (const Array &x0, const Array &dx) const
- Time time (const Date &) const
- const boost::shared_ptr< StochasticProcess1D > & **process** (Size i) const
- Disposable< Matrix > **correlation** () const

Protected Attributes

- std::vector< boost::shared_ptr< StochasticProcess1D > > **processes_**
- Matrix **sqrtCorrelation_**

7.511.2 Member Function Documentation

7.511.2.1 Disposable<Array> expectation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess.

7.511.2.2 Disposable<Matrix> stdDeviation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess.

7.511.2.3 Disposable<Matrix> covariance (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from StochasticProcess.

7.511.2.4 Disposable<Array> apply (const Array & *x0*, const Array & *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from StochasticProcess.

7.511.2.5 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from StochasticProcess.

7.512 Stock Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/stock.hpp>`

Inheritance diagram for Stock:

7.512.1 Detailed Description

Simple stock class.

Public Member Functions

- **Stock** (const Handle< Quote > &quote)
- bool isExpired () const

  returns whether the instrument is still tradable.

Protected Member Functions

- void performCalculations () const

7.512.2 Member Function Documentation

7.512.2.1 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from Instrument.

7.513 Stockholm Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/stockholm.hpp>`

Inheritance diagram for Stockholm:

7.513.1 Detailed Description

Stockholm calendar

Holidays:

- Saturdays
- Sundays
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- Midsummer Eve (Friday between June 18-24)
- New Year's Day, January 1st
- Epiphany, January 6th
- May Day, May 1st
- National Day, June 6th
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

7.514 StrikedTypePayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for StrikedTypePayoff:

7.514.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

Public Member Functions

- **StrikedTypePayoff** (Option::Type type, Real strike)
- Real **strike** () const

Protected Attributes

- Real **strike_**

7.515 StulzEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Basket/stulzengine.hpp>`

Inheritance diagram for StulzEngine:

7.515.1 Detailed Description

Pricing engine for 2D European Baskets.

This class implements formulae from "Options on the Minimum or the Maximum of Two Risky Assets", Rene Stulz, Journal of Financial Ecomomics (1982) 10, 161-185.

Test
the correctness of the returned value is tested by reproducing results available in literature.

Public Member Functions

- void **calculate** () const

7.516 SuperSharePayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for SuperSharePayoff:

7.516.1 Detailed Description

Binary supershare payoff.

Public Member Functions

- **SuperSharePayoff** (Option::Type type, Real strike, Real strikeIncrement)
- Real **operator()** (Real price) const
- Real **strikeIncrement** () const

7.517 SVD Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/svd.hpp>`

7.517.1 Detailed Description

Singular value decomposition.

Refer to Golub and Van Loan: Matrix computation, The Johns Hopkins University Press

Test
the correctness of the returned values is tested by checking their properties.

Public Member Functions

- **SVD** (const Matrix &)
- const Matrix & **U** () const
- const Matrix & **V** () const
- const Array & **singularValues** () const
- Disposable< Matrix > **S** () const
- Real **norm2** ()
- Real **cond** ()
- Integer **rank** ()

7.518 Swap Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swap.hpp>`

Inheritance diagram for Swap:

7.518.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

Public Member Functions

- **Swap** (const std::vector< boost::shared_ptr< CashFlow > > &firstLeg, const std::vector< boost::shared_ptr< CashFlow > > &secondLeg, const Handle< YieldTermStructure > &termStructure)

Instrument interface

- bool isExpired () const
  returns whether the instrument is still tradable.

Additional interface

- Date **startDate** () const
- Date **maturity** () const
- Real **firstLegBPS** () const
- Real **secondLegBPS** () const
- TimeBasket sensitivity (Integer basis=2) const

Protected Member Functions

- void setupExpired () const
- void performCalculations () const

Protected Attributes

- std::vector< boost::shared_ptr< CashFlow > > **firstLeg_**
- std::vector< boost::shared_ptr< CashFlow > > **secondLeg_**
- Handle< YieldTermStructure > **termStructure_**
- Real **firstLegBPS_**
- Real **secondLegBPS_**

7.518.2 Member Function Documentation

7.518.2.1 TimeBasket sensitivity (Integer *basis* = 2) const

Bug
this method must still be checked. It is not guaranteed to yield the right results.

7.518.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from Instrument.

7.518.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from Instrument.

7.519 SwapRateHelper Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp>`

Inheritance diagram for SwapRateHelper:

7.519.1 Detailed Description

Swap rate

Todo

currency and day counter of Xibor should be added to obtain well-defined SwapRateHelper

Warning:

This class assumes that the settlement date does not change between calls of setTermStructure().

Public Member Functions

- **SwapRateHelper** (const Handle< Quote > &rate, Integer n, TimeUnit units, Integer settlementDays, const Calendar &calendar, Frequency fixedFrequency, BusinessDayConvention fixedConvention, const DayCounter &fixedDayCount, Frequency floatingFrequency, BusinessDayConvention floatingConvention)
- **SwapRateHelper** (Rate rate, Integer n, TimeUnit units, Integer settlementDays, const Calendar &calendar, Frequency fixedFrequency, BusinessDayConvention fixedConvention, const DayCounter &fixedDayCount, Frequency floatingFrequency, BusinessDayConvention floatingConvention)
- Real **impliedQuote** () const
- Date latestDate () const

  latest relevant date

- void setTermStructure (YieldTermStructure *)

  sets the term structure to be used for pricing

Protected Attributes

- Integer **n_**
- TimeUnit **units_**
- Integer **settlementDays_**
- Calendar **calendar_**
- BusinessDayConvention **fixedConvention_**
- BusinessDayConvention **floatingConvention_**
- Frequency **fixedFrequency_**
- Frequency **floatingFrequency_**
- DayCounter **fixedDayCount_**
- Date **settlement_**
- Date **latestDate_**
- boost::shared_ptr< SimpleSwap > **swap_**
- Handle< YieldTermStructure > **termStructureHandle_**

7.519.2 Member Function Documentation

7.519.2.1 Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements RateHelper.

7.519.2.2 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from RateHelper.

7.520 Swaption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swaption.hpp>`

Inheritance diagram for Swaption:

7.520.1 Detailed Description

Swaption class

Test

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.

Todo

add explicit exercise lag

Public Member Functions

- **Swaption** (const boost::shared_ptr< SimpleSwap > &swap, const boost::shared_ptr< Exercise > &exercise, const Handle< YieldTermStructure > &termStructure, const boost::shared_ptr< PricingEngine > &engine)
- bool isExpired () const

  returns whether the instrument is still tradable.

- void setupArguments (Arguments *) const

Classes

- class arguments

  Arguments for swaption calculation

- class results

  Results from swaption calculation

7.520.2 Member Function Documentation

7.520.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from Instrument.

7.521 Swaption::arguments Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swaption.hpp>`

Inheritance diagram for Swaption::arguments:

7.521.1 Detailed Description

Arguments for swaption calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Rate **fairRate**
- Rate **fixedRate**
- Real **fixedBPS**

7.522 Swaption::results Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swaption.hpp>`

Inheritance diagram for Swaption::results:

7.522.1 Detailed Description

Results from swaption calculation

7.523 SwaptionVolatilityMatrix Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/swaptionvolmatrix.hpp>`

Inheritance diagram for SwaptionVolatilityMatrix:

7.523.1 Detailed Description

At-the-money swaption-volatility matrix.

This class provides the at-the-money volatility for a given swaption by interpolating a volatility matrix whose elements are the market volatilities of a set of swaption with given exercise date and length.

Todo

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Public Member Functions

- **SwaptionVolatilityMatrix** (const Date &referenceDate, const std::vector< Date > &exerciseDates, const std::vector< Period > &lengths, const Matrix &volatilities, const DayCounter &dayCounter)
- DayCounter dayCounter () const

  the day counter used for date/time conversion

- const std::vector< Date > & **exerciseDates** () const
- const std::vector< Period > & **lengths** () const
- Date maxStartDate () const

  the latest start date for which the term structure can return vols

- Time maxStartTime () const

  the latest start time for which the term structure can return vols

- Period maxLength () const

  the largest length for which the term structure can return vols

- Time maxTimeLength () const

  the largest length for which the term structure can return vols

- Real minStrike () const

  the minimum strike for which the term structure can return vols

- Real maxStrike () const

  the maximum strike for which the term structure can return vols

7.524 SwaptionVolatilityStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/swaptionvolstructure.hpp>`

Inheritance diagram for SwaptionVolatilityStructure:

7.524.1 Detailed Description

Swaption-volatility structure

This class is purely abstract and defines the interface of concrete swaption volatility structures which will be derived from this one.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- SwaptionVolatilityStructure ()
  default constructor
- SwaptionVolatilityStructure (const Date &referenceDate)
  initialize with a fixed reference date
- SwaptionVolatilityStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Volatility

- Volatility volatility (const Date &start, const Period &length, Rate strike, bool extrapolate=false) const
  returns the volatility for a given starting date and length
- Volatility volatility (Time start, Time length, Rate strike, bool extrapolate=false) const
  returns the volatility for a given starting time and length

Limits

- virtual Date maxStartDate () const =0
  the latest start date for which the term structure can return vols
- virtual Time maxStartTime () const
  the latest start time for which the term structure can return vols
- virtual Period maxLength () const =0
  the largest length for which the term structure can return vols
- virtual Time maxTimeLength () const
  the largest length for which the term structure can return vols

- virtual Real minStrike () const =0
  the minimum strike for which the term structure can return vols
- virtual Real maxStrike () const =0
  the maximum strike for which the term structure can return vols

Protected Member Functions

- virtual Volatility volatilityImpl (Time start, Time length, Rate strike) const =0
  implements the actual volatility calculation in derived classes
- virtual std::pair< Time, Time > convertDates (const Date &start, const Period &length) const
  implements the conversion between dates and times

7.524.2 Constructor & Destructor Documentation

7.524.2.1 SwaptionVolatilityStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.525 Sydney Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/sydney.hpp>`

Inheritance diagram for Sydney:

7.525.1 Detailed Description

Sydney calendar (New South Wales, Australia)

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Australia Day, January 26th (possibly moved to Monday)
- Good Friday
- Easter Monday
- ANZAC Day. April 25th (possibly moved to Monday)
- Queen's Birthday, second Monday in June
- Bank Holiday, first Monday in August
- Labour Day, first Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.526 SymmetricSchurDecomposition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/symmetricschurdecomposition.hpp>`

7.526.1 Detailed Description

symmetric threshold Jacobi algorithm.

Given a real symmetric matrix S, the Schur decomposition finds the eigenvalues and eigenvectors of S. If D is the diagonal matrix formed by the eigenvalues and U the unitarian matrix of the eigenvectors we can write the Schur decomposition as

$$S = U \cdot D \cdot U^T ,$$

where $\cdot$ is the standard matrix product and ^{T} is the transpose operator. This class implements the Schur decomposition using the symmetric threshold Jacobi algorithm. For details on the different Jacobi transfomations see "Matrix computation," second edition, by Golub and Van Loan, The Johns Hopkins University Press

Test
the correctness of the returned values is tested by checking their properties.

Public Member Functions

- SymmetricSchurDecomposition (const Matrix &s)
- const Array & **eigenvalues** () const
- const Matrix & **eigenvectors** () const

7.526.2 Constructor & Destructor Documentation

7.526.2.1 SymmetricSchurDecomposition (const Matrix & *s*)

Precondition:
s must be symmetric

7.527 TabulatedGaussLegendre Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp>`

7.527.1 Detailed Description

tabulated Gauss-Legendre quadratures

Public Member Functions

- **TabulatedGaussLegendre** (Size n=20)
- template<class F> Real **operator()** (const F &f) const
- void **order** (Size)
- Size **order** () const

7.528 Taipei Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/taipei.hpp>`

Inheritance diagram for Taipei:

7.528.1 Detailed Description

Taipei calendar

Holidays (data from <http://www.tse.com.tw/en/trading/trading_days.php>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Memorial Day, February 28
- Labor Day, May 1st
- Double Tenth National Day, October 10th

Other holidays for which no rule is given (data available for 2002-2005 only:)

- Chinese Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Festival
- Moon Festival

7.529 Taiwan Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/taiwan.hpp>`

Inheritance diagram for Taiwan:

7.529.1 Detailed Description

Taiwan calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Day, February 28th
- Labor Day, May 1st
- National Day, October 10th

Other holidays for which no rule is given (data available for 2004-2006 only:)

- Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Day
- Mid-Autumn Festival

Data from http://www.taifex.com.tw

7.530 TARGET Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/target.hpp>`

Inheritance diagram for TARGET:

7.530.1 Detailed Description

TARGET calendar

Holidays (see http://www.ecb.int):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday (since 2000)
- Easter Monday (since 2000)
- Labour Day, May 1st (since 2000)
- Christmas, December 25th
- Day of Goodwill, December 26th (since 2000)
- December 31st (1998, 1999, and 2001)

Test

the correctness of the returned results is tested against a list of known holidays.

7.531 TermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/termstructure.hpp>`

Inheritance diagram for TermStructure:

7.531.1 Detailed Description

Basic term-structure functionality.

Public Member Functions

Constructors

There are three ways in which a term structure can keep track of its reference date. The first is that such date is fixed; the second is that it is determined by advancing the current date of a given number of business days; and the third is that it is based on the reference date of some other structure.

In the first case, the constructor taking a date is to be used; the default implementation of referenceDate() will then return such date. In the second case, the constructor taking a number of days and a calendar is to be used; referenceDate() will return a date calculated based on the current evaluation date, and the term structure and its observers will be notified when the evaluation date changes. In the last case, the referenceDate() method must be overridden in derived classes so that it fetches and return the appropriate date.

- TermStructure ()
  default constructor
- TermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- TermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

Dates

- virtual const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- virtual Calendar calendar () const
  the calendar used for reference date calculation
- virtual DayCounter dayCounter () const =0
  the day counter used for date/time conversion

Observer interface

- void update ()

Protected Member Functions

- Time timeFromReference (const Date &date) const

  date/time conversion

7.531.2 Constructor & Destructor Documentation

7.531.2.1 TermStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.531.3 Member Function Documentation

7.531.3.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

Reimplemented in AffineTermStructure, ExtendedDiscountCurve, FlatForward, and CapVolatilityVector.

7.532 TermStructureConsistentModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/model.hpp>`

Inheritance diagram for TermStructureConsistentModel:

7.532.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

Public Member Functions

- **TermStructureConsistentModel** (const Handle< YieldTermStructure > &termStructure)
- const Handle< YieldTermStructure > & **termStructure** () const

7.533 TermStructureFittingParameter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp>`

Inheritance diagram for TermStructureFittingParameter:

7.533.1 Detailed Description

Deterministic time-dependent parameter used for yield-curve fitting.

Public Member Functions

- **TermStructureFittingParameter** (const boost::shared_ptr< Parameter::Impl > &impl)
- **TermStructureFittingParameter** (const Handle< YieldTermStructure > &term)

7.534 THBCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for THBCurrency:

7.534.1 Detailed Description

Thai baht.

The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

7.535 Thirty360 Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/thirty360.hpp>`

Inheritance diagram for Thirty360:

7.535.1 Detailed Description

30/360 day count convention

The 30/360 day count can be calculated according to US, European, or Italian conventions.

US (NASD) convention: if the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month. Also known as "30/360", "360/360", or "Bond Basis"

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month. Also known as "30E/360", or "Eurobond Basis"

Italian convention: starting dates or ending dates that occur on February and are grater than 27 become equal to 30 for computational sake.

Public Types

- enum **Convention** {
  USA, **BondBasis**, **European**, **EurobondBasis**,
  Italian }

Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

7.536 Tian Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for Tian:

7.536.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

Public Member Functions

- **Tian** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

Protected Attributes

- Real **up_**
- Real **down_**
- Real **pu_**
- Real **pd_**

7.537 Tibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/tibor.hpp>`

Inheritance diagram for Tibor:

7.537.1 Detailed Description

JPY TIBOR index

Tokyo Interbank Offered Rate.

Warning:
This is the rate fixed in Tokio by JBA. Use JPYLibor if you're interested in the London fixing by BBA.

Todo
check settlement days.

Public Member Functions

- **Tibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual365Fixed())

7.538 TimeBasket Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/timebasket.hpp>`

7.538.1 Detailed Description

Distribution over a number of dates.

Map interface

- typedef super::iterator **iterator**
- typedef super::const_iterator **const_iterator**
- typedef super::reverse_iterator **reverse_iterator**
- typedef super::const_reverse_iterator **const_reverse_iterator**
- bool hasDate (const Date &) const

  membership

Public Member Functions

- **TimeBasket** (const std::vector< Date > &dates, const std::vector< Real > &values)

Algebra

- TimeBasket & **operator+=** (const TimeBasket &other)
- TimeBasket & **operator-=** (const TimeBasket &other)

Other methods

- TimeBasket rebin (const std::vector< Date > &buckets) const

  redistribute the entries over the given dates

7.539 TimeGrid Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/timegrid.hpp>`

7.539.1 Detailed Description

time grid class

Todo

what was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

sequence interface

- typedef std::vector< Time >::const_iterator **const_iterator**
- typedef std::vector< Time >::const_reverse_iterator **const_reverse_iterator**
- Time **operator[ ]** (Size i) const
- Size **size** () const
- bool **empty** () const
- const_iterator **begin** () const
- const_iterator **end** () const
- const_reverse_iterator **rbegin** () const
- const_reverse_iterator **rend** () const
- Time **front** () const
- Time **back** () const

Public Member Functions

Constructors

- TimeGrid (Time end, Size steps)

  Regularly spaced time-grid.

- template<class Iterator> TimeGrid (Iterator begin, Iterator end)

  Time grid with mandatory time points.

- template<class Iterator> TimeGrid (Iterator begin, Iterator end, Size steps)

  Time grid with mandatory time points.

Time grid interface

- Size **findIndex** (Time t) const
- const std::vector< Time > & **mandatoryTimes** () const
- Time **dt** (Size i) const

7.539.2 Constructor & Destructor Documentation

7.539.2.1 TimeGrid (Iterator *begin*, Iterator *end*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. No additional points are added.

7.539.2.2 TimeGrid (Iterator *begin*, Iterator *end*, Size *steps*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. Additional points are then added with regular spacing between pairs of mandatory times in order to reach the desired number of steps.

7.540 Tokyo Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/tokyo.hpp>`

Inheritance diagram for Tokyo:

7.540.1 Detailed Description

Tokyo calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Bank Holiday, January 2nd
- Bank Holiday, January 3rd
- Coming of Age Day, 2nd Monday in January
- National Foundation Day, February 11th
- Vernal Equinox
- Greenery Day, April 29th
- Constitution Memorial Day, May 3rd
- Holiday for a Nation, May 4th
- Children's Day, May 5th
- Marine Day, 3rd Monday in July
- Respect for the Aged Day, 3rd Monday in September
- Autumnal Equinox
- Health and Sports Day, 2nd Monday in October
- National Culture Day, November 3rd
- Labor Thanksgiving Day, November 23rd
- Emperor's Birthday, December 23rd
- Bank Holiday, December 31st
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.

7.541 Toronto Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/toronto.hpp>`

Inheritance diagram for Toronto:

7.541.1 Detailed Description

Toronto calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Victoria Day, The Monday on or preceding 24 May
- Canada Day, July 1st (possibly moved to Monday)
- Provincial Holiday, first Monday of August
- Labour Day, first Monday of September
- Thanksgiving Day, second Monday of October
- Remembrance Day, November 11th
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.542 TqrEigenDecomposition Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/tqreigendecomposition.hpp>`

7.542.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

References:

Wilkinson, J.H. and Reinsch, C. 1971, Linear Algebra, vol. II of Handbook for Automatic Computation (New York: Springer-Verlag)

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

Test
the correctness of the result is tested by checking it against known good values.

Public Types

- enum **EigenVectorCalculation** { **WithEigenVector**, **WithoutEigenVector**, **OnlyFirstRowEigenVector** }
- enum **ShiftStrategy** { **NoShift**, **Overrelaxation**, **CloseEigenValue** }

Public Member Functions

- **TqrEigenDecomposition** (const Array &diag, const Array &sub, EigenVectorCalculation calc=WithEigenVector, ShiftStrategy strategy=CloseEigenValue)
- const Array & **eigenvalues** () const
- const Matrix & **eigenvectors** () const
- Size **iterations** () const

7.543 TrapezoidIntegral Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/trapezoidintegral.hpp>`

Inheritance diagram for TrapezoidIntegral:

7.543.1 Detailed Description

Integral of a one-dimensional function.

Given a target accuracy ϵ , the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f \mathrm{d}x = \frac{1}{2} f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \frac{1}{2} f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b - a)/N$. The number N of intervals is repeatedly increased until the target accuracy is reached.

Test
the correctness of the result is tested by checking it against known good values.

Public Types

- enum **Method** { **Default**, **MidPoint** }

Public Member Functions

- **TrapezoidIntegral** (Real accuracy, Method method=Default, Size maxIterations=Null< Size >())
- template<class F> Real **operator()** (const F &f, Real a, Real b) const
- Real **accuracy** () const
- Real & **accuracy** ()
- Method **method** () const
- Method & **method** ()
- Size **maxIterations** () const
- Size & **maxIterations** ()

Protected Member Functions

- template<class F> Real **defaultIteration** (const F &f, Real a, Real b, Real I, Size N) const
- template<class F> Real **midPointIteration** (const F &f, Real a, Real b, Real I, Size N) const

Protected Attributes

- Real **accuracy_**
- Method **method_**
- Size **maxIterations_**

7.544 Tree Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/tree.hpp>`

Inheritance diagram for Tree:

7.544.1 Detailed Description

template<class T> class QuantLib::Tree< T >

Tree approximating a single-factor diffusion

Derived classes must implement the following interface:

```
public:
  Real underlying(Size i, Size index) const;
  Size size(Size i) const;
  Size descendant(Size i, Size index, Size branch) const;
  Real probability(Size i, Size index, Size branch) const;
```

and provide a public enumeration

```
enum { branches = N };
```

where N is a suitable constant (2 for binomial, 3 for trinomial...)

Public Member Functions

- **Tree** (Size columns)
- Size **columns** () const

7.545 TreeCapFloorEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/treecapfloorengine.hpp>`

Inheritance diagram for TreeCapFloorEngine:

7.545.1 Detailed Description

Numerical lattice engine for cap/floors.

Public Member Functions

- **TreeCapFloorEngine** (const boost::shared_ptr< ShortRateModel > &model, Size timeSteps)
- **TreeCapFloorEngine** (const boost::shared_ptr< ShortRateModel > &model, const TimeGrid &timeGrid)
- void **calculate** () const

7.546 TreeSwaptionEngine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/treeswaptionengine.hpp>`

Inheritance diagram for TreeSwaptionEngine:

7.546.1 Detailed Description

Numerical lattice engine for swaptions.

Warning:
This engine is not guaranteed to work if the underlying swap has a start date in the past. When using this engine, prune the initial part of the swap so that it starts at $t \geq 0$.

Test
calculations are checked against cached results

Public Member Functions

- **TreeSwaptionEngine** (const boost::shared_ptr< ShortRateModel > &model, Size timeSteps)
- **TreeSwaptionEngine** (const boost::shared_ptr< ShortRateModel > &model, const TimeGrid &timeGrid)
- void **calculate** () const

7.547 TridiagonalOperator Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/tridiagonaloperator.hpp>`

Inheritance diagram for TridiagonalOperator:

7.547.1 Detailed Description

Base implementation for tridiagonal operator.

Warning:
to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class

Operator interface

- Disposable< Array > applyTo (const Array &v) const
  apply operator to a given array
- Disposable< Array > solveFor (const Array &rhs) const
  solve linear system for a given right-hand side
- Disposable< Array > SOR (const Array &rhs, Real tol) const
  solve linear system with SOR approach
- static Disposable< TridiagonalOperator > identity (Size size)
  identity instance

Public Types

- typedef Array **array_type**

Public Member Functions

- **TridiagonalOperator** (Size size=0)
- **TridiagonalOperator** (const Array &low, const Array &mid, const Array &high)
- **TridiagonalOperator** (const Disposable< TridiagonalOperator > &)
- TridiagonalOperator & **operator=** (const Disposable< TridiagonalOperator > &)

Inspectors

- Size **size** () const
- bool **isTimeDependent** ()
- const Array & **lowerDiagonal** () const
- const Array & **diagonal** () const
- const Array & **upperDiagonal** () const

Modifiers

- void **setFirstRow** (Real, Real)
- void **setMidRow** (Size, Real, Real, Real)
- void **setMidRows** (Real, Real, Real)
- void **setLastRow** (Real, Real)
- void **setTime** (Time t)

Utilities

- void **swap** (TridiagonalOperator &)

Protected Attributes

- Array **diagonal_**
- Array **lowerDiagonal_**
- Array **upperDiagonal_**
- boost::shared_ptr< TimeSetter > **timeSetter_**

Friends

- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &, const TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &, const TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator** * (Real, const TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator** * (const TridiagonalOperator &, Real)
- Disposable< TridiagonalOperator > **operator/** (const TridiagonalOperator &, Real)

Classes

- class TimeSetter

  encapsulation of time-setting logic

7.548 TridiagonalOperator::TimeSetter Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/tridiagonaloperator.hpp>`

7.548.1 Detailed Description

encapsulation of time-setting logic

Public Member Functions

- virtual void **setTime** (Time t, TridiagonalOperator &L) const =0

7.549 Trigeorgis Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp>`

Inheritance diagram for Trigeorgis:

7.549.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

Public Member Functions

- **Trigeorgis** (const boost::shared_ptr< StochasticProcess1D > &process, Time end, Size steps, Real strike)

7.550 TrinomialTree Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/trinomialtree.hpp>`

Inheritance diagram for TrinomialTree:

7.550.1 Detailed Description

Recombining trinomial tree class.

This class defines a recombining trinomial tree approximating a 1-D stochastic process.

Warning:
The diffusion term of the SDE must be independent of the underlying process.

Public Types

- enum { **branches** = 3 }

Public Member Functions

- **TrinomialTree** (const boost::shared_ptr< StochasticProcess1D > &process, const TimeGrid &timeGrid, bool isPositive=false)
- Real **dx** (Size i) const
- const TimeGrid & **timeGrid** () const
- Size **size** (Size i) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

Protected Attributes

- std::vector< Branching > **branchings_**
- Real **x0_**
- std::vector< Real > **dx_**
- TimeGrid **timeGrid_**

7.551 TRLCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for TRLCurrency:

7.551.1 Detailed Description

Turkish lira.

The ISO three-letter code is TRL; the numeric code is 792. It is divided in 100 kurus.

Obsoleted by the new Turkish lira since 2005.

7.552 TRLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/trlibor.hpp>`

Inheritance diagram for TRLibor:

7.552.1 Detailed Description

TRY LIBOR rate

TRY LIBOR fixed by TBA.

See <http://www.trlibor.org/trlibor/english/default.asp>

Public Member Functions

- **TRLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.553 TRYCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp>`

Inheritance diagram for TRYCurrency:

7.553.1 Detailed Description

New Turkish lira.

The ISO three-letter code is TRY; the numeric code is 949. It is divided in 100 new kurus.

7.554 TTDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for TTDCurrency:

7.554.1 Detailed Description

Trinidad & Tobago dollar.

The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

7.555 TWDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp>`

Inheritance diagram for TWDCurrency:

7.555.1 Detailed Description

Taiwan dollar.

The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

7.556 TwoFactorModel Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/twofactormodel.hpp>`

Inheritance diagram for TwoFactorModel:

7.556.1 Detailed Description

Abstract base-class for two-factor models.

Public Member Functions

- **TwoFactorModel** (Size nParams)
- virtual boost::shared_ptr< ShortRateDynamics > dynamics () const =0

  Returns the short-rate dynamics.

- boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const

  Returns a two-dimensional trinomial tree.

Classes

- class ShortRateDynamics

  Class describing the dynamics of the two state variables.

- class ShortRateTree

  Recombining two-dimensional tree discretizing the state variable.

7.557 TwoFactorModel::ShortRateDynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/twofactormodel.hpp>`

7.557.1 Detailed Description

Class describing the dynamics of the two state variables.

We assume here that the short-rate is a function of two state variables x and y.

$$r_t = f(t, x_t, y_t)$$

of two state variables x_t and y_t . These stochastic processes satisfy

$$x_t = \mu_x(t, x_t)dt + \sigma_x(t, x_t)dW_t^x$$

and

$$y_t = \mu_y(t, y_t)dt + \sigma_y(t, y_t)dW_t^y$$

where W^x and W^y are two brownian motions satisfying

$$dW_t^x dW_t^y = \rho dt$$

.

Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< StochasticProcess1D > &xProcess, const boost::shared_ptr< StochasticProcess1D > &yProcess, Real correlation)
- virtual Rate **shortRate** (Time t, Real x, Real y) const =0
- const boost::shared_ptr< StochasticProcess1D > & xProcess () const

  Risk-neutral dynamics of the first state variable x.

- const boost::shared_ptr< StochasticProcess1D > & yProcess () const

  Risk-neutral dynamics of the second state variable y.

- Real correlation () const

  Correlation ρ between the two brownian motions.

7.558 TwoFactorModel::ShortRateTree Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/twofactormodel.hpp>`

Inheritance diagram for TwoFactorModel::ShortRateTree:

7.558.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

Public Member Functions

- ShortRateTree (const boost::shared_ptr< TrinomialTree > &tree1, const boost::shared_ptr< TrinomialTree > &tree2, const boost::shared_ptr< ShortRateDynamics > &dynamics)

  Plain tree build-up from short-rate dynamics.

- DiscountFactor **discount** (Size i, Size index) const

7.559 TypePayoff Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp>`

Inheritance diagram for TypePayoff:

7.559.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

Protected Attributes

- Option::Type **type_**

7.560 UnitedKingdom Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/unitedkingdom.hpp>`

Inheritance diagram for UnitedKingdom:

7.560.1 Detailed Description

United Kingdom calendars.

Public holidays (data from http://www.dti.gov.uk/er/bankhol.htm):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the stock exchange:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the metals exchange:

- Saturdays
- Sundays

- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Todo
add LIFFE

Test
the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum Market { Settlement, Exchange, **Metals** }
  UK calendars.

Public Member Functions

- **UnitedKingdom** (Market market=Settlement)

7.560.2 Member Enumeration Documentation

7.560.2.1 enum Market

UK calendars.

Enumerator:
Settlement generic settlement calendar
Exchange London stock-exchange calendar.

7.561 UnitedStates Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/unitedstates.hpp>`

Inheritance diagram for UnitedStates:

7.561.1 Detailed Description

United States calendars.

Public holidays (see: http://www.opm.gov/fedhol/):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday, or to Friday if on Saturday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Holidays for the stock exchange (data from http://www.nyse.com):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January (since 1998)
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Presidential election day, first Tuesday in November of election years (until 1980)

- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)
- Special historic closings (see http://www.nyse.com/about/1022221392381.html)

Holidays for the government bond market (data from http://www.bondmarkets.com):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Test
the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum Market { Settlement, Exchange, **GovernmentBond** }
  US calendars.

Public Member Functions

- **UnitedStates** (Market market=Settlement)

7.561.2 Member Enumeration Documentation

7.561.2.1 enum Market

US calendars.

Enumerator:

Settlement generic settlement calendar

Exchange New York stock-exchange calendar.

7.562 UpFrontIndexedCoupon Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/upfrontindexedcoupon.hpp>`

Inheritance diagram for UpFrontIndexedCoupon:

7.562.1 Detailed Description

up front indexed coupon class

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **UpFrontIndexedCoupon** (Real nominal, const Date &paymentDate, const boost::shared_ptr< Xibor > &index, const Date &startDate, const Date &endDate, Integer fixingDays, Spread spread=0.0, const Date &refPeriodStart=Date(), const Date &refPeriodEnd=Date(), const DayCounter &dayCounter=DayCounter())

FloatingRateCoupon interface

- Date fixingDate () const
  fixing date

Visitability

- virtual void **accept** (AcyclicVisitor &)

Protected Attributes

- Calendar **calendar_**

7.563 UpRounding Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp>`

Inheritance diagram for UpRounding:

7.563.1 Detailed Description

Up-rounding.

Public Member Functions

- **UpRounding** (Integer precision, Integer digit=5)

7.564 USDCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for USDCurrency:

7.564.1 Detailed Description

U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

7.565 USDLibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/usdlibor.hpp>`

Inheritance diagram for USDLibor:

7.565.1 Detailed Description

USD LIBOR rate

US Dollar LIBOR fixed by BBA.

See <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>.

Public Member Functions

- **USDLibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.566 Value Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/instrument.hpp>`

Inheritance diagram for Value:

7.566.1 Detailed Description

pricing results

Public Member Functions

- void **reset** ()

Public Attributes

- Real **value**
- Real **errorEstimate**

7.567 VanillaOption Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/vanillaoption.hpp>`

Inheritance diagram for VanillaOption:

7.567.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

Public Member Functions

- **VanillaOption** (const boost::shared_ptr< StochasticProcess > &, const boost::shared_ptr< StrikedTypePayoff > &, const boost::shared_ptr< Exercise > &, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())

Classes

- class engine

  Vanilla option engine base class.

7.568 VanillaOption::engine Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/vanillaoption.hpp>`

Inheritance diagram for VanillaOption::engine:

7.568.1 Detailed Description

Vanilla option engine base class.

7.569 Vasicek Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/vasicek.hpp>`

Inheritance diagram for Vasicek:

7.569.1 Detailed Description

Vasicek model class

This class implements the Vasicek model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where a , b and σ are constants; a risk premium λ can also be specified.

Public Member Functions

- **Vasicek** (Rate r0=0.05, Real a=0.1, Real b=0.05, Real sigma=0.01, Real lambda=0.0)
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- virtual boost::shared_ptr< ShortRateDynamics > dynamics () const

  returns the short-rate dynamics

Protected Member Functions

- virtual Real **A** (Time t, Time T) const
- virtual Real **B** (Time t, Time T) const
- Real **a** () const
- Real **b** () const
- Real **lambda** () const
- Real **sigma** () const

Protected Attributes

- Real **r0_**
- Parameter & **a_**
- Parameter & **b_**
- Parameter & **sigma_**
- Parameter & **lambda_**

Classes

- class Dynamics

  Short-rate dynamics in the Vasicek model.

7.570 Vasicek::Dynamics Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactor-Models/vasicek.hpp>`

7.570.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b .

Public Member Functions

- **Dynamics** (Real a, Real b, Real sigma, Real r0)
- virtual Real **variable** (Time, Rate r) const
- virtual Real **shortRate** (Time, Real x) const

7.571 VEBCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp>`

Inheritance diagram for VEBCurrency:

7.571.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

7.572 Visitor Class Template Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/visitor.hpp>`

7.572.1 Detailed Description

template<class T> class QuantLib::Visitor< T >

Visitor for a specific class

Public Member Functions

- virtual void **visit** (T &)=0

7.573 Warsaw Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/warsaw.hpp>`

Inheritance diagram for Warsaw:

7.573.1 Detailed Description

Warsaw calendar

Holidays:

- Saturdays
- Sundays
- Easter Monday
- Corpus Christi
- New Year's Day, January 1st
- May Day, May 1st
- Constitution Day, May 3rd
- Assumption of the Blessed Virgin Mary, August 15th
- All Saints Day, November 1st
- Independence Day, November 11th
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

7.574 Wellington Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/wellington.hpp>`

Inheritance diagram for Wellington:

7.574.1 Detailed Description

Wellington calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday or Tuesday)
- Day after New Year's Day, January 2st (possibly moved to Monday or Tuesday)
- Anniversary Day, Monday nearest January 22nd
- Waitangi Day. February 6th
- Good Friday
- Easter Monday
- ANZAC Day. April 25th
- Queen's Birthday, first Monday in June
- Labour Day, fourth Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Note:

The holiday rules for Wellington were documented by David Gilbert for IDB (http://www.jrefinery.com/ibd/)

7.575 Xibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/xibor.hpp>`

Inheritance diagram for Xibor:

7.575.1 Detailed Description

base class for LIBOR-like indexes

Public Member Functions

- **Xibor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency &currency, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter, const Handle< YieldTermStructure > &h)
- virtual Date **valueDate** (const Date &fixingDate) const
- virtual Date **maturityDate** (const Date &valueDate) const

Index interface

- Rate fixing (const Date &fixingDate) const
  returns the fixing at the given date

Observer interface

- void update ()

Inspectors

- std::string name () const
  Returns the name of the index.
- Period **tenor** () const
- Frequency **frequency** () const
- Integer **settlementDays** () const
- const Currency & **currency** () const
- Calendar **calendar** () const
- bool **isAdjusted** () const
- BusinessDayConvention **businessDayConvention** () const
- DayCounter **dayCounter** () const
- boost::shared_ptr< YieldTermStructure > **termStructure** () const

Protected Attributes

- std::string **familyName_**
- Integer **n_**
- TimeUnit **units_**
- Integer **settlementDays_**
- Currency **currency_**
- Calendar **calendar_**
- BusinessDayConvention **convention_**
- DayCounter **dayCounter_**
- Handle< YieldTermStructure > **termStructure_**

7.575.2 Member Function Documentation

7.575.2.1 Rate fixing (const Date & *fixingDate*) const [virtual]

returns the fixing at the given date

Note:
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implements Index.

7.575.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements Observer.

7.575.2.3 std::string name () const [virtual]

Returns the name of the index.

Warning:
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements Index.

7.576 YieldTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/yieldtermstructure.hpp>`

Inheritance diagram for YieldTermStructure:

7.576.1 Detailed Description

Interest-rate term structure.

This abstract class defines the interface of concrete rate structures which will be derived from this one.

Rates are assumed to be annual continuous compounding.

Todo

add derived class ParSwapTermStructure similar to ZeroYieldTermStructure, DiscountStructure, ForwardRateStructure

Test

observability against evaluation date changes is checked.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- YieldTermStructure ()
  default constructor
- YieldTermStructure (const Date &referenceDate)
  initialize with a fixed reference date
- YieldTermStructure (Integer settlementDays, const Calendar &)
  calculate the reference date based on the global evaluation date

zero-yield rates

These methods return the implied zero-yield rate for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.

- InterestRate zeroRate (const Date &d, const DayCounter &resultDayCounter, Compounding comp, Frequency freq=Annual, bool extrapolate=false) const
- InterestRate zeroRate (Time t, Compounding comp, Frequency freq=Annual, bool extrapolate=false) const

discount factors

These methods return the discount factor for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.

- DiscountFactor **discount** (const Date &, bool extrapolate=false) const
- DiscountFactor discount (Time, bool extrapolate=false) const

forward rates

These methods returns the implied forward interest rate between two dates or times. In the former case, times are calculated as fractions of year from the reference date.

- InterestRate forwardRate (const Date &d1, const Date &d2, const DayCounter &resultDayCounter, Compounding comp, Frequency freq=Annual, bool extrapolate=false) const
- InterestRate forwardRate (Time t1, Time t2, Compounding comp, Frequency freq=Annual, bool extrapolate=false) const

par rates

These methods returns the implied par rate for a given sequence of payments at the given dates or times. In the former case, times are calculated as fractions of year from the reference date.

Warning:
though somewhat related to a swap rate, this method is not to be used for the fair rate of a real swap, since it does not take into account all the market conventions' details. The correct way to evaluate such rate is to instantiate a SimpleSwap with the correct conventions, pass it the term structure and call the swap's fairRate() method.

- Rate **parRate** (Integer tenor, const Date &startDate, Frequency freq=Annual, bool extrapolate=false) const
- Rate parRate (const std::vector< Date > &dates, Frequency freq=Annual, bool extrapolate=false) const
- Rate parRate (const std::vector< Time > &times, Frequency freq=Annual, bool extrapolate=false) const
- Rate parRate (Year tenor, Time t0, Frequency freq=Annual, bool extrapolate=false) const

Dates

- virtual Date maxDate () const =0
  the latest date for which the curve can return rates
- virtual Time maxTime () const
  the latest time for which the curve can return rates

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual discount and rate calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual DiscountFactor discountImpl (Time) const =0
  discount calculation

7.576.2 Constructor & Destructor Documentation

7.576.2.1 YieldTermStructure ()

default constructor

Warning:
term structures initialized by means of this constructor must manage their own reference date by overriding the referenceDate() method.

7.576.3 Member Function Documentation

7.576.3.1 InterestRate zeroRate (const Date & *d*, const DayCounter & *resultDayCounter*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required daycounting rule.

7.576.3.2 InterestRate zeroRate (Time *t*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for calculating the passed time t.

7.576.3.3 DiscountFactor discount (Time, bool *extrapolate* = false) const

The same day-counting rule used by the term structure should be used for calculating the passed time t.

7.576.3.4 InterestRate forwardRate (const Date & *d1*, const Date & *d2*, const DayCounter & *resultDayCounter*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required day-counting rule.

7.576.3.5 InterestRate forwardRate (Time *t1*, Time *t2*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for the calculating the passed times t1 and t2.

7.576.3.6 Rate parRate (const std::vector< Date > & *dates*, Frequency *freq* = Annual, bool *extrapolate* = false) const

the first date in the vector must equal the start date; the following dates must equal the payment dates.

7.576.3.7 Rate parRate (const std::vector< Time > & *times*, Frequency *freq* = Annual, bool *extrapolate* = false) const

the first time in the vector must equal the start time; the following times must equal the payment times.

7.576.3.8 Rate parRate (Year *tenor*, Time *t0*, Frequency *freq* = `Annual`, bool *extrapolate* = `false`) const

Deprecated
use the overload taking a vector of times

7.577 ZARCurrency Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/africa.hpp>`

Inheritance diagram for ZARCurrency:

7.577.1 Detailed Description

South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

7.578 ZeroCouponBond Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/zerocouponbond.hpp>`

Inheritance diagram for ZeroCouponBond:

7.578.1 Detailed Description

zero-coupon bond

Test
calculations are tested by checking results against cached values.

Public Member Functions

- **ZeroCouponBond** (const Date &issueDate, const Date &maturityDate, Integer settlementDays, const DayCounter &dayCounter, const Calendar &calendar, BusinessDayConvention convention=Following, Real redemption=100.0, const Handle< YieldTermStructure > &discountCurve=Handle< YieldTermStructure >())

7.579 ZeroSpreadedTermStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zerospreadedtermstructure.hpp>`

Inheritance diagram for ZeroSpreadedTermStructure:

7.579.1 Detailed Description

Term structure with an added spread on the zero yield rate.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Test

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Public Member Functions

- **ZeroSpreadedTermStructure** (const Handle< YieldTermStructure > &, const Handle< Quote > &spread)

YieldTermStructure interface

- DayCounter dayCounter () const
  the day counter used for date/time conversion
- Calendar calendar () const
  the calendar used for reference date calculation
- const Date & referenceDate () const
  the reference date, i.e., the date at which discount = 1
- Date maxDate () const
  the latest date for which the curve can return rates
- Time maxTime () const
  the latest time for which the curve can return rates

Protected Member Functions

- Rate zeroYieldImpl (Time) const
  returns the spreaded zero yield rate
- Rate forwardImpl (Time) const
  returns the spreaded forward rate

7.579.2 Member Function Documentation

7.579.2.1 Rate forwardImpl (Time) const [protected]

returns the spreaded forward rate

Warning:
This method must disappear should the spread become a curve

7.580 ZeroYield Struct Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bootstraptraits.hpp>`

7.580.1 Detailed Description

Zero-curve traits.

Static Public Member Functions

- static Rate **initialValue** ()
- static Rate **initialGuess** ()
- static Rate **guess** (const YieldTermStructure *c, const Date &d)
- static Rate **minValueAfter** (Size, const std::vector< Real > &)
- static Rate **maxValueAfter** (Size, const std::vector< Real > &)
- static void **updateGuess** (std::vector< Rate > &data, Rate rate, Size i)

7.581 ZeroYieldStructure Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zeroyieldstructure.hpp>`

Inheritance diagram for ZeroYieldStructure:

7.581.1 Detailed Description

Zero-yield term structure.

This abstract class acts as an adapter to YieldTermStructure allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes. Discount and forward are calculated from zero yields.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- **ZeroYieldStructure** ()
- **ZeroYieldStructure** (const Date &referenceDate)
- **ZeroYieldStructure** (Integer settlementDays, const Calendar &)

Protected Member Functions

YieldTermStructure implementation

- DiscountFactor discountImpl (Time) const
- virtual Rate zeroYieldImpl (Time) const =0

  zero-yield calculation

7.581.2 Member Function Documentation

7.581.2.1 DiscountFactor discountImpl (Time) const `[protected, virtual]`

Returns the discount factor for the given date calculating it from the zero yield.

Implements YieldTermStructure.

7.582 Zibor Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/zibor.hpp>`

Inheritance diagram for Zibor:

7.582.1 Detailed Description

CHF ZIBOR rate

Zurich Interbank Offered Rate.

Warning:
This is the rate fixed in Zurich by BBA. Use CHFLibor if you're interested in the London fixing by BBA.

Todo
check settlement days and day-count.

Public Member Functions

- **Zibor** (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const DayCounter &dc=Actual360())

7.583 Zurich Class Reference

`#include <builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/zurich.hpp>`

Inheritance diagram for Zurich:

7.583.1 Detailed Description

Zurich calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Berchtoldstag, January 2nd
- Good Friday
- Easter Monday
- Ascension Day
- Whit Monday
- Labour Day, May 1st
- National Day, August 1st
- Christmas, December 25th
- St. Stephen's Day, December 26th

Chapter 8

QuantLib File Documentation

8.1 builddir/build/BUILD/QuantLib-0.3.11/ql/argsandresults.hpp File Reference

8.1.1 Detailed Description

Base classes for generic arguments and results.

`#include <ql/qldefines.hpp>`

Include dependency graph for argsandresults.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Arguments

  base class for generic argument groups

- class Results

  base class for generic result groups

Defines

- #define **QL_MIN_VOLATILITY** 1.0e-7
- #define **QL_MIN_DIVYIELD** 1.0e-7
- #define **QL_MAX_VOLATILITY** 4.0
- #define **QL_MAX_DIVYIELD** 4.0

8.2 builddir/build/BUILD/QuantLib-0.3.11/ql/calendar.hpp File Reference

8.2.1 Detailed Description

calendar class

`#include <ql/date.hpp>`

`#include <ql/Patterns/bridge.hpp>`

`#include <set>`

Include dependency graph for calendar.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CalendarImpl

  abstract base class for calendar implementations

- class Calendar

  calendar class

- class Calendar::WesternImpl

  partial calendar implementation

Enumerations

- enum QuantLib::BusinessDayConvention {

  QuantLib::Unadjusted, QuantLib::Preceding, QuantLib::ModifiedPreceding, QuantLib::Following,

  QuantLib::ModifiedFollowing, QuantLib::MonthEndReference }

  Business Day conventions.

8.3 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/beijing.hpp File Reference

8.3.1 Detailed Description

Beijing calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for beijing.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Beijing
  Beijing calendar

8.4 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/bombay.hpp File Reference

8.4.1 Detailed Description

Bombay calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for bombay.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Bombay

  Bombay calendar

8.5 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/bratislava.hpp File Reference

8.5.1 Detailed Description

Bratislava calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for bratislava.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Bratislava

  Bratislava calendar

8.6 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/budapest.hpp File Reference

8.6.1 Detailed Description

Budapest calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for budapest.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Budapest

  Budapest calendar

8.7 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/copenhagen.hpp File Reference

8.7.1 Detailed Description

Copenhagen calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for copenhagen.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Copenhagen

  Copenhagen calendar

8.8 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/germany.hpp File Reference

8.8.1 Detailed Description

German calendars.

`#include <ql/calendar.hpp>`

Include dependency graph for germany.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Germany

  German calendars.

8.9 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/helsinki.hpp File Reference

8.9.1 Detailed Description

Helsinki calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for helsinki.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Helsinki

  Helsinki calendar

8.10 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/hongkong.hpp File Reference

8.10.1 Detailed Description

Hong Kong calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for hongkong.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HongKong
  Hong Kong calendar.

8.11 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/istanbul.hpp File Reference

8.11.1 Detailed Description

Istanbul calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for istanbul.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Istanbul

  Istanbul calendar

8.12 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/italy.hpp File Reference

8.12.1 Detailed Description

Italian calendars.

`#include <ql/calendar.hpp>`

Include dependency graph for italy.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Italy
  Italian calendars.

8.13 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/johannesburg.hpp File Reference

8.13.1 Detailed Description

Johannesburg calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for johannesburg.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Johannesburg
  Johannesburg calendar

8.14 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/jointcalendar.hpp File Reference

8.14.1 Detailed Description

Joint calendar.

`#include <ql/calendar.hpp>`

`#include <vector>`

Include dependency graph for jointcalendar.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class JointCalendar
  Joint calendar.

Enumerations

- enum **JointCalendarRule** { QuantLib::JoinHolidays, QuantLib::JoinBusinessDays }
  rules for joining calendars

8.15 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/nullcalendar.hpp File Reference

8.15.1 Detailed Description

Calendar for reproducing theoretical calculations.

`#include <ql/calendar.hpp>`

Include dependency graph for nullcalendar.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class NullCalendar

  Calendar for reproducing theoretical calculations.

8.16 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/oslo.hpp File Reference

8.16.1 Detailed Description

Oslo calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for oslo.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Oslo

  Oslo calendar

8.17 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/prague.hpp File Reference

8.17.1 Detailed Description

Prague calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for prague.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Prague
  Prague calendar

8.18 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/riyadh.hpp File Reference

8.18.1 Detailed Description

Riyadh calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for riyadh.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Riyadh
  Riyadh calendar

8.19 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/seoul.hpp File Reference

8.19.1 Detailed Description

South Korea calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for seoul.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Seoul

  Seoul calendar

8.20 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/singapore.hpp File Reference

8.20.1 Detailed Description

Singapore calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for singapore.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Singapore
  Singapore calendar

8.21 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/stockholm.hpp File Reference

8.21.1 Detailed Description

Stockholm calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for stockholm.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Stockholm

  Stockholm calendar

8.22 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/sydney.hpp File Reference

8.22.1 Detailed Description

Sydney calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for sydney.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Sydney
  Sydney calendar (New South Wales, Australia)

8.23 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/taipei.hpp File Reference

8.23.1 Detailed Description

Taipei calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for taipei.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Taipei
  Taipei calendar

8.24 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/taiwan.hpp File Reference

8.24.1 Detailed Description

Taiwan calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for taiwan.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Taiwan
  Taiwan calendar

8.25 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/target.hpp File Reference

8.25.1 Detailed Description

TARGET calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for target.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TARGET
  TARGET calendar

8.26 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/tokyo.hpp File Reference

8.26.1 Detailed Description

Tokyo calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for tokyo.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Tokyo
  Tokyo calendar

8.27 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/toronto.hpp File Reference

8.27.1 Detailed Description

Toronto calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for toronto.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Toronto

  Toronto calendar

8.28 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/unitedkingdom.hp File Reference

8.28.1 Detailed Description

UK calendars.

`#include <ql/calendar.hpp>`

Include dependency graph for unitedkingdom.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class UnitedKingdom
  United Kingdom calendars.

8.29 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/unitedstates.hpp File Reference

8.29.1 Detailed Description

US calendars.

`#include <ql/calendar.hpp>`

Include dependency graph for unitedstates.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class UnitedStates

  United States calendars.

8.30 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/warsaw.hpp File Reference

8.30.1 Detailed Description

Warsaw calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for warsaw.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Warsaw
  Warsaw calendar

8.31 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/wellington.hpp File Reference

8.31.1 Detailed Description

Wellington calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for wellington.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Wellington
  Wellington calendar

8.32 builddir/build/BUILD/QuantLib-0.3.11/ql/Calendars/zurich.hpp File Reference

8.32.1 Detailed Description

Zurich calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for zurich.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Zurich
  Zurich calendar

8.33 builddir/build/BUILD/QuantLib-0.3.11/ql/capvolstructures.hpp File Reference

8.33.1 Detailed Description

Cap/Floor volatility structures.

`#include <ql/termstructure.hpp>`

`#include <ql/Math/extrapolation.hpp>`

Include dependency graph for capvolstructures.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CapVolatilityStructure

  Cap/floor term-volatility structure.

- class CapletVolatilityStructure

  Caplet/floorlet forward-volatility structure.

8.34 builddir/build/BUILD/QuantLib-0.3.11/ql/cashflow.hpp File Reference

8.34.1 Detailed Description

Base class for cash flows.

`#include <ql/date.hpp>`

`#include <ql/Patterns/observable.hpp>`

`#include <ql/Patterns/visitor.hpp>`

Include dependency graph for cashflow.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CashFlow

  Base class for cash flows.

8.35 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/analysis.hpp File Reference

8.35.1 Detailed Description

Cash-flow analysis functions.

```
#include <ql/cashflow.hpp>
#include <ql/interestrate.hpp>
#include <ql/yieldtermstructure.hpp>
#include <vector>
```

Include dependency graph for analysis.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct Duration
  duration type
- class Cashflows
  cashflows analysis functions

8.36 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/basispointsensitivity.hpp File Reference

8.36.1 Detailed Description

basis point sensitivity calculator

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/CashFlows/fixedratecoupon.hpp>`

`#include <ql/CashFlows/timebasket.hpp>`

Include dependency graph for basispointsensitivity.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BPSCalculator

  basis point sensitivity (BPS) calculator

- class BPSBasketCalculator

Functions

- Real QuantLib::BasisPointSensitivity (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &)

  Collective basis-point sensitivity of a cash-flow sequence.

- TimeBasket QuantLib::BasisPointSensitivityBasket (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &, Integer basis)

8.37 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/cashflowvectors.hpp File Reference

8.37.1 Detailed Description

Cash flow vector builders.

`#include <ql/cashflow.hpp>`

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/schedule.hpp>`

Include dependency graph for cashflowvectors.hpp:

Namespaces

- namespace **QuantLib**

Functions

- std::vector< boost::shared_ptr< CashFlow > > QuantLib::FixedRateCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const std::vector< Rate > &couponRates, const DayCounter &dayCount, const DayCounter &firstPeriodDayCount=DayCounter())

  helper function building a sequence of fixed rate coupons

- std::vector< boost::shared_ptr< CashFlow > > QuantLib::FloatingRateCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< Xibor > &index, Integer fixingDays, const std::vector< Spread > &spreads=std::vector< Spread >(), const DayCounter &dayCounter=DayCounter())

  helper function building a sequence of par coupons

8.38 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/coupon.hpp File Reference

8.38.1 Detailed Description

Coupon accruing over a fixed period.

`#include <ql/cashflow.hpp>`

`#include <ql/calendar.hpp>`

`#include <ql/daycounter.hpp>`

Include dependency graph for coupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Coupon

  coupon accruing over a fixed period

8.39 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/fixedratecoupon.hpp File Reference

8.39.1 Detailed Description

Coupon paying a fixed annual rate.

`#include <ql/CashFlows/coupon.hpp>`

Include dependency graph for fixedratecoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FixedRateCoupon

  Coupon paying a fixed interest rate

8.40 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/floatingratecoupon.hpp File Reference

8.40.1 Detailed Description

Coupon paying a variable rate.

`#include <ql/CashFlows/coupon.hpp>`

`#include <ql/Utilities/null.hpp>`

Include dependency graph for floatingratecoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FloatingRateCoupon

  Coupon paying a variable rate

8.41 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/inarrearindexedcoupon.hpp File Reference

8.41.1 Detailed Description

in-arrear floating-rate coupon

`#include <ql/CashFlows/indexedcoupon.hpp>`

`#include <ql/capvolstructures.hpp>`

Include dependency graph for inarrearindexedcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InArrearIndexedCoupon

  In-arrear floating-rate coupon.

8.42 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/indexedcashflowvectors.hpp File Reference

8.42.1 Detailed Description

Indexed cash-flow vector builders.

`#include <ql/CashFlows/shortindexedcoupon.hpp>`

`#include <ql/schedule.hpp>`

Include dependency graph for indexedcashflowvectors.hpp:

Namespaces

- namespace **QuantLib**

Functions

- template<class IndexedCouponType> std::vector< boost::shared_ptr< CashFlow > > QuantLib::IndexedCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< Xibor > &index, Integer fixingDays, const std::vector< Spread > &spreads, const DayCounter &dayCounter=DayCounter())

  helper function building a leg of floating coupons

8.43 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/indexedcoupon.hpp File Reference

8.43.1 Detailed Description

indexed coupon

`#include <ql/CashFlows/floatingratecoupon.hpp>`

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for indexedcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class IndexedCoupon

  Base indexed coupon class.

8.44 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/parcoupon.hpp File Reference

8.44.1 Detailed Description

Coupon at par on a term structure.

`#include <ql/CashFlows/floatingratecoupon.hpp>`

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for parcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ParCoupon

  coupon at par on a term structure

8.45 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/shortfloatingcoupon.hpp File Reference

8.45.1 Detailed Description

Short (or long) coupon at par on a term structure.

`#include <ql/CashFlows/parcoupon.hpp>`

`#include <ql/CashFlows/shortindexedcoupon.hpp>`

Include dependency graph for shortfloatingcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Short< ParCoupon >

  Short coupon at par on a term structure

8.46 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/shortindexedcoupon.hpp File Reference

8.46.1 Detailed Description

Short (or long) indexed coupon.

`#include <ql/CashFlows/indexedcoupon.hpp>`

Include dependency graph for shortindexedcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Short
  Short indexed coupon

8.47 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/simplecashflow.hpp File Reference

8.47.1 Detailed Description

Predetermined cash flow.

`#include <ql/cashflow.hpp>`

Include dependency graph for simplecashflow.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SimpleCashFlow
  Predetermined cash flow.

8.48 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/timebasket.hpp File Reference

8.48.1 Detailed Description

Distribution over a number of dates

```
#include <ql/date.hpp>
#include <ql/Utilities/null.hpp>
#include <vector>
#include <map>
```

Include dependency graph for timebasket.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TimeBasket

  Distribution over a number of dates.

8.49 builddir/build/BUILD/QuantLib-0.3.11/ql/CashFlows/upfrontindexedcoupon.hpp File Reference

8.49.1 Detailed Description

Up front indexed coupon.

`#include <ql/CashFlows/indexedcoupon.hpp>`

Include dependency graph for upfrontindexedcoupon.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class UpFrontIndexedCoupon
  up front indexed coupon class

8.50 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/africa.hpp File Reference

8.50.1 Detailed Description

African currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and http://www.thefinancials.com/vortex/CurrencyFormats.html

`#include <ql/currency.hpp>`

Include dependency graph for africa.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ZARCurrency

  South-African rand.

8.51 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/america.hpp File Reference

8.51.1 Detailed Description

American currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and http://www.thefinancials.com/vortex/CurrencyFormats.html

`#include <ql/currency.hpp>`

Include dependency graph for america.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ARSCurrency
  Argentinian peso.
- class BRLCurrency
  Brazilian real.
- class CADCurrency
  Canadian dollar.
- class CLPCurrency
  Chilean peso.
- class COPCurrency
  Colombian peso.
- class MXNCurrency
  Mexican peso.
- class TTDCurrency
  Trinidad & Tobago dollar.
- class USDCurrency
  U.S. dollar.
- class VEBCurrency
  Venezuelan bolivar.

8.52 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/asia.hpp File Reference

8.52.1 Detailed Description

Asian currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and http://www.thefinancials.com/vortex/CurrencyFormats.html

`#include <ql/currency.hpp>`

Include dependency graph for asia.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BDTCurrency
  Bangladesh taka.
- class CNYCurrency
  Chinese yuan.
- class HKDCurrency
  Honk Kong dollar.
- class ILSCurrency
  Israeli shekel.
- class INRCurrency
  Indian rupee.
- class IQDCurrency
  Iraqi dinar.
- class IRRCurrency
  Iranian rial.
- class JPYCurrency
  Japanese yen.
- class KRWCurrency
  South-Korean won.
- class KWDCurrency
  Kuwaiti dinar.

- class NPRCurrency
  Nepal rupee.
- class PKRCurrency
  Pakistani rupee.
- class SARCurrency
  Saudi riyal.
- class SGDCurrency
  Singapore dollar.
- class THBCurrency
  Thai baht.
- class TWDCurrency
  Taiwan dollar.

8.53 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/europe.hpp File Reference

8.53.1 Detailed Description

European currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and http://www.thefinancials.com/vortex/CurrencyFormats.html

`#include <ql/currency.hpp>`

Include dependency graph for europe.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BGLCurrency
  Bulgarian lev.
- class BYRCurrency
  Belarussian ruble.
- class CHFCurrency
  Swiss franc.
- class CYPCurrency
  Cyprus pound.
- class CZKCurrency
  Czech koruna.
- class DKKCurrency
  Danish krone.
- class EEKCurrency
  Estonian kroon.
- class EURCurrency
  European Euro.
- class GBPCurrency
  British pound sterling.
- class HUFCurrency
  Hungarian forint.

- class ISKCurrency
  Iceland krona.
- class LTLCurrency
  Lithuanian litas.
- class LVLCurrency
  Latvian lat.
- class MTLCurrency
  Maltese lira.
- class NOKCurrency
  Norwegian krone.
- class PLNCurrency
  Polish zloty.
- class ROLCurrency
  Romanian leu.
- class SEKCurrency
  Swedish krona.
- class SITCurrency
  Slovenian tolar.
- class SKKCurrency
  Slovak koruna.
- class TRLCurrency
  Turkish lira.
- class TRYCurrency
  New Turkish lira.
- class ATSCurrency
  Austrian shilling.
- class BEFCurrency
  Belgian franc.
- class DEMCurrency
  Deutsche mark.
- class ESPCurrency
  Spanish peseta.
- class FIMCurrency
  Finnish markka.

- class FRFCurrency
  French franc.
- class GRDCurrency
  Greek drachma.
- class IEPCurrency
  Irish punt.
- class ITLCurrency
  Italian lira.
- class LUFCurrency
  Luxembourg franc.
- class NLGCurrency
  Dutch guilder.
- class PTECurrency
  Portuguese escudo.

8.54 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/exchangerateman File Reference

8.54.1 Detailed Description

exchange-rate repository

```
#include <ql/exchangerate.hpp>
#include <ql/date.hpp>
#include <ql/Patterns/singleton.hpp>
#include <list>
#include <map>
```

Include dependency graph for exchangeratemanager.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ExchangeRateManager
  exchange-rate repository

8.55 builddir/build/BUILD/QuantLib-0.3.11/ql/Currencies/oceania.hpp File Reference

8.55.1 Detailed Description

Oceanian currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and http://www.thefinancials.com/vortex/CurrencyFormats.html

`#include <ql/currency.hpp>`

Include dependency graph for oceania.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AUDCurrency
  Australian dollar.
- class NZDCurrency
  New Zealand dollar.

8.56 builddir/build/BUILD/QuantLib-0.3.11/ql/currency.hpp File Reference

8.56.1 Detailed Description

Known currencies.

```
#include <ql/Math/rounding.hpp>
#include <ql/errors.hpp>
#include <boost/shared_ptr.hpp>
#include <ostream>
```

Include dependency graph for currency.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Currency
  Currency specification

8.57 builddir/build/BUILD/QuantLib-0.3.11/ql/date.hpp File Reference

8.57.1 Detailed Description

date- and time-related classes, typedefs and enumerations

```
#include <ql/errors.hpp>
#include <ql/types.hpp>
#include <utility>
#include <functional>
#include <ostream>
```

Include dependency graph for date.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Classes

- struct IMM

  Main cycle of the International Money Market (a.k.a. IMM) Months.

- class Period

  Time period described by a number of a given time unit.

- class Date

  Concrete date class.

Typedefs

- typedef Integer QuantLib::Day

  Day number.

- typedef Integer QuantLib::Year

  Year number.

Enumerations

- enum QuantLib::Weekday {
  Sunday = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,
  Thursday = 5, **Friday** = 6, **Saturday** = 7, **Sun** = 1,

Mon = 2, **Tue** = 3, **Wed** = 4, **Thu** = 5,
Fri = 6, **Sat** = 7 }

- enum QuantLib::Month {
  January = 1, **February** = 2, **March** = 3, **April** = 4,
  May = 5, **June** = 6, **July** = 7, **August** = 8,
  September = 9, **October** = 10, **November** = 11, **December** = 12,
  Jan = 1, **Feb** = 2, **Mar** = 3, **Apr** = 4,
  Jun = 6, **Jul** = 7, **Aug** = 8, **Sep** = 9,
  Oct = 10, **Nov** = 11, **Dec** = 12 }
  Month names.

- enum QuantLib::IMMMonth { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }
  Main cycle of the International Money Market (a.k.a. IMM) Months.

- enum QuantLib::Frequency {
  QuantLib::NoFrequency = -1, QuantLib::Once = 0, QuantLib::Annual = 1, QuantLib::Semiannual = 2,
  QuantLib::EveryFourthMonth = 3, QuantLib::Quarterly = 4, QuantLib::Bimonthly = 6, QuantLib::Monthly = 12 }
  Frequency of events.

- enum QuantLib::TimeUnit { **Days**, **Weeks**, **Months**, **Years** }
  Units used to describe time periods.

Functions

- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const long_weekday_holder &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const short_weekday_holder &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const shortest_weekday_holder &)
- detail::long_weekday_holder QuantLib::io::long_weekday (Weekday)
  output weekdays in long format

- detail::short_weekday_holder QuantLib::io::short_weekday (Weekday)
  output weekdays in short format (three letters)

- detail::shortest_weekday_holder QuantLib::io::shortest_weekday (Weekday)
  output weekdays in shortest format (two letters)

- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const long_period_holder &)

- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const short_period_holder &)

- detail::long_period_holder QuantLib::io::long_period (const Period &)
  output periods in long format (e.g. "2 weeks")

- detail::short_period_holder QuantLib::io::short_period (const Period &)

  output periods in short format (e.g. "2w")

- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const short_date_holder &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const long_date_holder &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const iso_date_holder &)
- detail::short_date_holder QuantLib::io::short_date (const Date &)

  output dates in short format (mm/dd/yyyy)

- detail::long_date_holder QuantLib::io::long_date (const Date &)

  output dates in long format (Month ddth, yyyy)

- detail::iso_date_holder QuantLib::io::iso_date (const Date &)

  output dates in ISO format (yyyy-mm-dd)

- Period **QuantLib::operator** * (Integer n, TimeUnit units)
- Period **QuantLib::operator** * (TimeUnit units, Integer n)
- bool **QuantLib::operator==** (const Period &p1, const Period &p2)
- bool **QuantLib::operator!=** (const Period &p1, const Period &p2)
- bool **QuantLib::operator**> (const Period &p1, const Period &p2)
- bool **QuantLib::operator**<= (const Period &p1, const Period &p2)
- bool **QuantLib::operator**>= (const Period &p1, const Period &p2)
- BigInteger **QuantLib::operator-** (const Date &d1, const Date &d2)
- bool **QuantLib::operator==** (const Date &d1, const Date &d2)
- bool **QuantLib::operator!=** (const Date &d1, const Date &d2)
- bool **QuantLib::operator**< (const Date &d1, const Date &d2)
- bool **QuantLib::operator**<= (const Date &d1, const Date &d2)
- bool **QuantLib::operator**> (const Date &d1, const Date &d2)
- bool **QuantLib::operator**>= (const Date &d1, const Date &d2)

8.58 builddir/build/BUILD/QuantLib-0.3.11/ql/daycounter.hpp File Reference

8.58.1 Detailed Description

day counter class

`#include <ql/date.hpp>`

`#include <ql/Patterns/bridge.hpp>`

Include dependency graph for daycounter.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DayCounterImpl

  abstract base class for day counter implementations

- class DayCounter

  day counter class

8.59 builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/actual360.hpp File Reference

8.59.1 Detailed Description

act/360 day counter

`#include <ql/daycounter.hpp>`

Include dependency graph for actual360.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Actual360

  Actual/360 day count convention.

8.60 builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/actual365fixed.hpp File Reference

8.60.1 Detailed Description

Actual/365 (Fixed) day counter.

`#include <ql/daycounter.hpp>`

Include dependency graph for actual365fixed.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Actual365Fixed

  Actual/365 (Fixed) day count convention.

8.61 builddir/build/BUILD/QuantLib-0.3.11/ql/Day-Counters/actualactual.hpp File Reference

8.61.1 Detailed Description

act/act day counters

`#include <ql/daycounter.hpp>`

Include dependency graph for actualactual.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ActualActual

  Actual/Actual day count.

8.62 builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/one.hpp File Reference

8.62.1 Detailed Description

1/1 day counter

`#include <ql/daycounter.hpp>`

Include dependency graph for one.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OneDayCounter
  1/1 day count convention

8.63 builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/simpledaycounter.hpp File Reference

8.63.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

`#include <ql/daycounter.hpp>`

Include dependency graph for simpledaycounter.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SimpleDayCounter

  Simple day counter for reproducing theoretical calculations.

8.64 builddir/build/BUILD/QuantLib-0.3.11/ql/DayCounters/thirty360.hpp File Reference

8.64.1 Detailed Description

30/360 day counters

`#include <ql/daycounter.hpp>`

Include dependency graph for thirty360.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Thirty360

  30/360 day count convention

8.65 builddir/build/BUILD/QuantLib-0.3.11/ql/discretizedasset.hpp File Reference

8.65.1 Detailed Description

Discretized asset classes.

```
#include <ql/numericalmethod.hpp>
#include <ql/Math/array.hpp>
#include <ql/Math/comparison.hpp>
#include <ql/exercise.hpp>
```

Include dependency graph for discretizedasset.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DiscretizedAsset

  Discretized asset class used by numerical methods.

- class DiscretizedDiscountBond

  Useful discretized discount bond asset.

- class DiscretizedOption

  Discretized option on a given asset.

8.66 builddir/build/BUILD/QuantLib-0.3.11/ql/errors.hpp File Reference

8.66.1 Detailed Description

Classes and functions for error handling.

```
#include <ql/qldefines.hpp>
#include <boost/assert.hpp>
#include <boost/current_function.hpp>
#include <boost/shared_ptr.hpp>
#include <exception>
#include <sstream>
```

Include dependency graph for errors.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Error

  Base error class.

Defines

- #define QL_FAIL(message)

  throw an error (possibly with file and line information)

- #define QL_ASSERT(condition, message)

  throw an error if the given condition is not verified

- #define QL_REQUIRE(condition, message)

  throw an error if the given pre-condition is not verified

- #define QL_ENSURE(condition, message)

  throw an error if the given post-condition is not verified

8.66.2 Define Documentation

8.66.2.1 #define QL_FAIL(message)

Value:

```
do { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__,__LINE__, \
                          BOOST_CURRENT_FUNCTION,_ql_msg_stream.str()); \
} while (false)
```

throw an error (possibly with file and line information)

8.66.2.2 #define QL_ASSERT(condition, message)

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__,__LINE__, \
                          BOOST_CURRENT_FUNCTION,_ql_msg_stream.str()); \
} else
```

throw an error if the given condition is not verified

8.66.2.3 #define QL_REQUIRE(condition, message)

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__,__LINE__, \
                          BOOST_CURRENT_FUNCTION,_ql_msg_stream.str()); \
} else
```

throw an error if the given pre-condition is not verified

Examples:
DiscreteHedging.cpp, and swapvaluation.cpp.

8.66.2.4 #define QL_ENSURE(condition, message)

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__,__LINE__, \
                          BOOST_CURRENT_FUNCTION,_ql_msg_stream.str()); \
} else
```

throw an error if the given post-condition is not verified

8.67 builddir/build/BUILD/QuantLib-0.3.11/ql/exchangerate.hpp File Reference

8.67.1 Detailed Description

exchange rate between two currencies

```
#include <ql/money.hpp>
#include <ql/Utilities/null.hpp>
#include <utility>
```

Include dependency graph for exchangerate.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ExchangeRate

  exchange rate between two currencies

8.68 builddir/build/BUILD/QuantLib-0.3.11/ql/exercise.hpp File Reference

8.68.1 Detailed Description

Option exercise classes and payoff function.

`#include <ql/date.hpp>`

`#include <vector>`

Include dependency graph for exercise.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Exercise
  Base exercise class.
- class EarlyExercise
  Early-exercise base class.
- class AmericanExercise
  American exercise.
- class BermudanExercise
  Bermudan exercise.
- class EuropeanExercise
  European exercise.

8.69 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/americancondition.hpp File Reference

8.69.1 Detailed Description

american option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
#include <ql/discretizedasset.hpp>
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americancondition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AmericanCondition

  American exercise condition.

8.70 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/boundarycondition.hpp File Reference

8.70.1 Detailed Description

boundary conditions for differential operators

`#include <ql/Utilities/null.hpp>`

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for boundarycondition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BoundaryCondition

  Abstract boundary condition class for finite difference problems.

- class NeumannBC

  Neumann boundary condition (i.e., constant derivative).

- class DirichletBC

  Neumann boundary condition (i.e., constant value).

8.71 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/bsmoperator.hpp File Reference

8.71.1 Detailed Description

differential operator for Black-Scholes-Merton equation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

`#include <ql/Processes/blackscholesprocess.hpp>`

Include dependency graph for bsmoperator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BSMOperator

  Black-Scholes-Merton differential operator.

8.72 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/bsmtermoperator.hpp File Reference

8.72.1 Detailed Description

differential operator for Black-Scholes-Merton equation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

`#include <ql/Processes/blackscholesprocess.hpp>`

Include dependency graph for bsmtermoperator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BSMTermOperator

  Black-Scholes-Merton differential operator.

8.73 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/cranknicolson.hpp File Reference

8.73.1 Detailed Description

Crank-Nicolson scheme for finite difference methods.

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for cranknicolson.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CrankNicolson

  Crank-Nicolson scheme for finite difference methods.

8.74 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/dminus.hpp File Reference

8.74.1 Detailed Description

D_- matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dminus.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DMinus

  D_- matricial representation

8.75 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/dplus.hpp File Reference

8.75.1 Detailed Description

D_+ matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dplus.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DPlus

  D_+ matricial representation

8.76 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/dplusdminus.hpp File Reference

8.76.1 Detailed Description

D_+D_- matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dplusdminus.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DPlusDMinus

  D_+D_- *matricial representation*

8.77 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/dzero.hpp File Reference

8.77.1 Detailed Description

D_0 matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dzero.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DZero

  D_0 *matricial representation*

8.78 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/expliciteuler.hpp File Reference

8.78.1 Detailed Description

explicit Euler scheme for finite difference methods

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for expliciteuler.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ExplicitEuler

  Forward Euler scheme for finite difference methods.

8.79 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/fdtypedefs.hpp File Reference

8.79.1 Detailed Description

default choices for template instantiations

`#include <ql/FiniteDifferences/cranknicolson.hpp>`

`#include <ql/FiniteDifferences/parallelevolver.hpp>`

Include dependency graph for fdtypedefs.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > > QuantLib::StandardFiniteDifferenceModel

  default choice for finite-difference model

- typedef FiniteDifferenceModel< ParallelEvolver< CrankNicolson< TridiagonalOperator > > > QuantLib::StandardSystemFiniteDifferenceModel

  default choice for parallel finite-difference model

- typedef StepCondition< Array > QuantLib::StandardStepCondition

  default choice for step condition

8.80 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

8.80.1 Detailed Description

generic finite difference model

`#include <ql/FiniteDifferences/stepcondition.hpp>`

`#include <ql/FiniteDifferences/boundarycondition.hpp>`

`#include <ql/FiniteDifferences/operatortraits.hpp>`

Include dependency graph for finitedifferencemodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FiniteDifferenceModel

  Generic finite difference model.

8.81 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/impliciteuler.hpp File Reference

8.81.1 Detailed Description

implicit Euler scheme for finite difference methods

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for impliciteuler.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ImplicitEuler

  Backward Euler scheme for finite difference methods.

8.82 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/mixedscheme.hpp File Reference

8.82.1 Detailed Description

Mixed (explicit/implicit) scheme for finite difference methods.

`#include <ql/FiniteDifferences/finitedifferencemodel.hpp>`

Include dependency graph for mixedscheme.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MixedScheme

  Mixed (explicit/implicit) scheme for finite difference methods.

8.83 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/onefactoroperator.hpp File Reference

8.83.1 Detailed Description

general differential operator for one-factor interest rate models

```
#include <ql/qldefines.hpp>
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for onefactoroperator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OneFactorOperator

  Interest-rate single factor model differential operator.

8.84 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/operatortraits.hpp File Reference

8.84.1 Detailed Description

Differential operator traits.

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
#include <ql/FiniteDifferences/stepcondition.hpp>
#include <vector>
```

Include dependency graph for operatortraits.hpp:

Namespaces

- namespace **QuantLib**

8.85 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/parallelevolver.hpp File Reference

8.85.1 Detailed Description

Parallel evolver for multiple arrays.

This class takes the evolver class and creates a new class which evolves each of the evolvers in parallel. Part of what this does is to take the types for each evolver class and then wrapper them so that they create new types which are sets of the old types.

This class is intended to be run in situations where there are parallel differential equations such as with some convertible bond models.

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
#include <ql/FiniteDifferences/stepcondition.hpp>
#include <ql/numericalmethod.hpp>
#include <vector>
```

Include dependency graph for parallelevolver.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class StepConditionSet

  Parallel evolver for multiple arrays.

8.86 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/shoutcondition.hpp File Reference

8.86.1 Detailed Description

shout option exercise condition

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/discretizedasset.hpp>`

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for shoutcondition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ShoutCondition

  Shout option condition.

8.87 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/stepcondition.hpp File Reference

8.87.1 Detailed Description

conditions to be applied at every time step

`#include <ql/types.hpp>`

Include dependency graph for stepcondition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class StepCondition

  condition to be applied at every time step

- class NullCondition

  null step condition

8.88 builddir/build/BUILD/QuantLib-0.3.11/ql/Finite-Differences/tridiagonaloperator.hpp File Reference

8.88.1 Detailed Description

tridiagonal operator

`#include <ql/Math/array.hpp>`

`#include <boost/shared_ptr.hpp>`

Include dependency graph for tridiagonaloperator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TridiagonalOperator

  Base implementation for tridiagonal operator.

- class TridiagonalOperator::TimeSetter

  encapsulation of time-setting logic

Functions

- void **QuantLib::swap** (TridiagonalOperator &, TridiagonalOperator &)
- Disposable< TridiagonalOperator > **QuantLib::operator+** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator-** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator+** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **QuantLib::operator-** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **QuantLib::operator** * (Real a, const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator** * (const TridiagonalOperator &D, Real a)
- Disposable< TridiagonalOperator > **QuantLib::operator/** (const TridiagonalOperator &D, Real a)

8.89 builddir/build/BUILD/QuantLib-0.3.11/ql/FiniteDifferences/valueatcenter.hpp File Reference

8.89.1 Detailed Description

compute value, first, and second derivatives at grid center

`#include <ql/Math/array.hpp>`

Include dependency graph for valueatcenter.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Real QuantLib::valueAtCenter (const Array &a)
- Real QuantLib::firstDerivativeAtCenter (const Array &a, const Array &grid)
- Real QuantLib::secondDerivativeAtCenter (const Array &a, const Array &grid)

8.90 builddir/build/BUILD/QuantLib-0.3.11/ql/grid.hpp File Reference

8.90.1 Detailed Description

Grid constructors.

`#include <ql/Math/array.hpp>`

Include dependency graph for grid.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Disposable< Array > **QuantLib::CenteredGrid** (Real center, Real dx, Size steps)
- Disposable< Array > **QuantLib::BoundedGrid** (Real xMin, Real xMax, Size steps)

8.91 builddir/build/BUILD/QuantLib-0.3.11/ql/handle.hpp File Reference

8.91.1 Detailed Description

Globally accessible relinkable pointer.

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for handle.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Link

  Relinkable access to a shared pointer.

- class Handle

  Globally accessible relinkable pointer.

8.92 builddir/build/BUILD/QuantLib-0.3.11/ql/history.hpp File Reference

8.92.1 Detailed Description

history class

```
#include <ql/date.hpp>
#include <ql/Utilities/null.hpp>
#include <boost/iterator/filter_iterator.hpp>
#include <vector>
```

Include dependency graph for history.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class History
  Container for historical data.
- class History::Entry
  single datum in history
- class History::const_iterator
  random access iterator on history entries

8.93 builddir/build/BUILD/QuantLib-0.3.11/ql/index.hpp File Reference

8.93.1 Detailed Description

purely virtual base class for indexes

`#include <ql/calendar.hpp>`

`#include <ql/currency.hpp>`

`#include <ql/daycounter.hpp>`

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for index.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Index

  purely virtual base class for indexes

8.94 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/audlibor.hpp File Reference

8.94.1 Detailed Description

AUD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/sydney.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for audlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AUDLibor
  AUD LIBOR rate

8.95 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/cadlibor.hpp File Reference

8.95.1 Detailed Description

CAD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/toronto.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for cadlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CADLibor
  CAD LIBOR rate

8.96 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/cdor.hpp File Reference

8.96.1 Detailed Description

CDOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/toronto.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for cdor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Cdor
  CDOR rate

8.97 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/chflibor.hpp File Reference

8.97.1 Detailed Description

CHF LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/zurich.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for chflibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CHFLibor
  CHF LIBOR rate

8.98 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/dkklibor.hpp File Reference

8.98.1 Detailed Description

DKK LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/copenhagen.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for dkklibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DKKLibor
  DKK LIBOR rate

8.99 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/euribor.hpp File Reference

8.99.1 Detailed Description

Euribor index

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/target.hpp>`

`#include <ql/DayCounters/actual360.hpp>`

`#include <ql/Currencies/europe.hpp>`

Include dependency graph for euribor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Euribor

  Euribor index

8.100 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/eurlibor.hpp File Reference

8.100.1 Detailed Description

EUR LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/target.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for eurlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class EURLibor
  EUR LIBOR rate

8.101 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/gbplibor.hpp File Reference

8.101.1 Detailed Description

GBP LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for gbplibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GBPLibor
  GBP LIBOR rate

8.102 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/indexmanager.hpp File Reference

8.102.1 Detailed Description

global repository for past index fixings

`#include <ql/history.hpp>`

`#include <ql/Patterns/singleton.hpp>`

`#include <map>`

Include dependency graph for indexmanager.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class IndexManager
  global repository for past index fixings

8.103 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/jibar.hpp File Reference

8.103.1 Detailed Description

JIBAR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/johannesburg.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/africa.hpp>
```

Include dependency graph for jibar.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Jibar
  JIBAR rate

8.104 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/jpylibor.hpp File Reference

8.104.1 Detailed Description

JPY LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/tokyo.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/asia.hpp>
```

Include dependency graph for jpylibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class JPYLibor
  JPY LIBOR rate

8.105 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/libor.hpp File Reference

8.105.1 Detailed Description

base class for BBA LIBOR indexes

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for libor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Libor
  base class for BBA LIBOR indexes

8.106 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/nzdlibor.hpp File Reference

8.106.1 Detailed Description

NZD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/wellington.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for nzdlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class NZDLibor
  NZD LIBOR rate

8.107 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/tibor.hpp File Reference

8.107.1 Detailed Description

JPY TIBOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/tokyo.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/asia.hpp>
```

Include dependency graph for tibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Tibor
  JPY TIBOR index

8.108 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/trlibor.hpp File Reference

8.108.1 Detailed Description

TRY LIBOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/istanbul.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for trlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TRLibor
  TRY LIBOR rate

8.109 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/usdlibor.hpp File Reference

8.109.1 Detailed Description

USD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/unitedstates.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for usdlibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class USDLibor
  USD LIBOR rate

8.110 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/xibor.hpp File Reference

8.110.1 Detailed Description

base class for LIBOR-like indexes

`#include <ql/index.hpp>`

`#include <ql/yieldtermstructure.hpp>`

Include dependency graph for xibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Xibor

  base class for LIBOR-like indexes

8.111 builddir/build/BUILD/QuantLib-0.3.11/ql/Indexes/zibor.hpp File Reference

8.111.1 Detailed Description

CHF ZIBOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/zurich.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for zibor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Zibor
  CHF ZIBOR rate

8.112 builddir/build/BUILD/QuantLib-0.3.11/ql/instrument.hpp File Reference

8.112.1 Detailed Description

Abstract instrument class.

`#include <ql/Patterns/lazyobject.hpp>`

`#include <ql/pricingengine.hpp>`

`#include <ql/errors.hpp>`

`#include <ql/Utilities/null.hpp>`

Include dependency graph for instrument.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Instrument
  Abstract instrument class.
- class Value
  pricing results

8.113 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/asianoption.hp File Reference

8.113.1 Detailed Description

Asian option on a single asset.

`#include <ql/Instruments/oneassetstrikedoption.hpp>`

Include dependency graph for asianoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct Average
  placeholder for enumerated averaging types
- class ContinuousAveragingAsianOption
  Continuous-averaging Asian option.
- class DiscreteAveragingAsianOption
  Discrete-averaging Asian option.
- class DiscreteAveragingAsianOption::arguments
  Extra arguments for single-asset discrete-average Asian option.
- class ContinuousAveragingAsianOption::arguments
  Extra arguments for single-asset continuous-average Asian option.
- class DiscreteAveragingAsianOption::engine
  Discrete-averaging Asian engine base class.
- class ContinuousAveragingAsianOption::engine
  Continuous-averaging Asian engine base class.

8.114 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/barrieroption.h File Reference

8.114.1 Detailed Description

Barrier option on a single asset.

`#include <ql/Instruments/oneassetstrikedoption.hpp>`

Include dependency graph for barrieroption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct Barrier

  Placeholder for enumerated barrier types.

- class BarrierOption

  Barrier option on a single asset.

- class BarrierOption::arguments

  Arguments for barrier option calculation

- class BarrierOption::engine

  Barrier engine base class

8.115 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/basketoption.h File Reference

8.115.1 Detailed Description

Basket option on a number of assets.

`#include <ql/Instruments/payoffs.hpp>`

`#include <ql/Instruments/multiassetoption.hpp>`

Include dependency graph for basketoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BasketOption

  Basket option on a number of assets.

- class BasketOption::arguments

  Arguments for basket option calculation

- class BasketOption::engine

  Basket option engine base class

8.116 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/bond.hpp File Reference

8.116.1 Detailed Description

concrete bond class

```
#include <ql/instrument.hpp>
#include <ql/cashflow.hpp>
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/interestrate.hpp>
#include <ql/yieldtermstructure.hpp>
#include <vector>
```

Include dependency graph for bond.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Bond

  Base bond class.

8.117 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/callabilitysche File Reference

8.117.1 Detailed Description

Schedule of put/call dates.

`#include <ql/date.hpp>`

`#include <vector>`

Include dependency graph for callabilityschedule.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef std::vector< Callability > **QuantLib::CallabilitySchedule**

8.118 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/capfloor.hpp File Reference

8.118.1 Detailed Description

Cap and Floor class.

```
#include <ql/numericalmethod.hpp>
#include <ql/instrument.hpp>
#include <ql/cashflow.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/quote.hpp>
```

Include dependency graph for capfloor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CapFloor
  Base class for cap-like instruments.
- class Cap
  Concrete cap class.
- class Floor
  Concrete floor class.
- class Collar
  Concrete collar class.
- class CapFloor::arguments
  Arguments for cap/floor calculation
- class CapFloor::results
  Results from cap/floor calculation

8.119 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/cliquetoption.h File Reference

8.119.1 Detailed Description

Cliquet option.

`#include <ql/Instruments/oneassetstrikedoption.hpp>`

Include dependency graph for cliquetoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CliquetOption
  cliquet (Ratchet) option
- class CliquetOption::arguments
  Arguments for cliquet option calculation
- class CliquetOption::engine
  Cliquet engine base class.

8.120 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/dividendsched File Reference

8.120.1 Detailed Description

Schedule of dividend dates.

`#include <ql/date.hpp>`

`#include <vector>`

Include dependency graph for dividendschedule.hpp:

Namespaces

- namespace **QuantLib**

8.121 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/dividendvanill File Reference

8.121.1 Detailed Description

Vanilla option on a single asset with discrete dividends.

`#include <ql/Instruments/vanillaoption.hpp>`

`#include <ql/Instruments/dividendschedule.hpp>`

Include dependency graph for dividendvanillaoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DividendVanillaOption

  Single-asset vanilla option (no barriers) with discrete dividends.

- class DividendVanillaOption::arguments

  Arguments for dividend vanilla option calculation

- class DividendVanillaOption::engine

  Dividend vanilla option engine base class.

8.122 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/europeanoption File Reference

8.122.1 Detailed Description

European option on a single asset.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for europeanoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class EuropeanOption

  European option on a single asset.

8.123 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/fixedcouponbo File Reference

8.123.1 Detailed Description

fixed-coupon bond

`#include <ql/Instruments/bond.hpp>`

Include dependency graph for fixedcouponbond.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FixedCouponBond
  fixed-coupon bond

8.124 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/floatingratebon File Reference

8.124.1 Detailed Description

floating-rate bond

`#include <ql/Instruments/bond.hpp>`

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for floatingratebond.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FloatingRateBond
  floating-rate bond

8.125 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/forwardvanilla File Reference

8.125.1 Detailed Description

Forward version of a vanilla option.

`#include <ql/Instruments/vanillaoption.hpp>`

`#include <ql/PricingEngines/Forward/forwardengine.hpp>`

Include dependency graph for forwardvanillaoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ForwardVanillaOption

  Forward version of a vanilla option.

8.126 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/multiassetoptio File Reference

8.126.1 Detailed Description

Option on multiple assets.

`#include <ql/option.hpp>`

`#include <ql/stochasticprocess.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for multiassetoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MultiAssetOption

  Base class for options on multiple assets.

- class MultiAssetOption::arguments

  Arguments for multi-asset option calculation

- class MultiAssetOption::results

  Results from multi-asset option calculation

8.127 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetoption File Reference

8.127.1 Detailed Description

Option on a single asset.

`#include <ql/option.hpp>`

`#include <ql/stochasticprocess.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for oneassetoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OneAssetOption

  Base class for options on a single asset.

- class OneAssetOption::arguments

  Arguments for single-asset option calculation

- class OneAssetOption::results

  Results from single-asset option calculation

8.128 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/oneassetstriked File Reference

8.128.1 Detailed Description

Option on a single asset with striked payoff.

`#include <ql/Instruments/oneassetoption.hpp>`

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for oneassetstrikedoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OneAssetStrikedOption

  Base class for options on a single asset with striked payoff.

8.129 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/payoffs.hpp File Reference

8.129.1 Detailed Description

Payoffs for various options.

`#include <ql/option.hpp>`

Include dependency graph for payoffs.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TypePayoff
  Intermediate class for call/put/straddle payoffs.
- class StrikedTypePayoff
  Intermediate class for payoffs based on a fixed strike.
- class PlainVanillaPayoff
  Plain-vanilla payoff.
- class PercentageStrikePayoff
  Payoff with strike expressed as percentage
- class CashOrNothingPayoff
  Binary cash-or-nothing payoff.
- class AssetOrNothingPayoff
  Binary asset-or-nothing payoff.
- class GapPayoff
  Binary gap payoff.
- class SuperSharePayoff
  Binary supershare payoff.

8.130 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/quantoforward File Reference

8.130.1 Detailed Description

Quanto version of a forward vanilla option.

`#include <ql/Instruments/quantovanillaoption.hpp>`

`#include <ql/Instruments/forwardvanillaoption.hpp>`

Include dependency graph for quantoforwardvanillaoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class QuantoForwardVanillaOption

  Quanto version of a forward vanilla option.

8.131 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/quantovanillao File Reference

8.131.1 Detailed Description

Quanto version of a vanilla option.

`#include <ql/Instruments/vanillaoption.hpp>`

`#include <ql/PricingEngines/Quanto/quantoengine.hpp>`

Include dependency graph for quantovanillaoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class QuantoVanillaOption

  quanto version of a vanilla option

8.132 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/simpleswap.hp File Reference

8.132.1 Detailed Description

Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/swap.hpp>
#include <ql/Indexes/xibor.hpp>
#include <ql/schedule.hpp>
```

Include dependency graph for simpleswap.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SimpleSwap
  Simple fixed-rate vs Libor swap.
- class SimpleSwap::arguments
  Arguments for simple swap calculation
- class SimpleSwap::results
  Results from simple swap calculation

8.133 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/stock.hpp File Reference

8.133.1 Detailed Description

concrete stock class

`#include <ql/instrument.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for stock.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Stock

  Simple stock class.

8.134 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swap.hpp File Reference

8.134.1 Detailed Description

Interest rate swap.

`#include <ql/instrument.hpp>`

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/cashflow.hpp>`

`#include <ql/CashFlows/timebasket.hpp>`

Include dependency graph for swap.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Swap

  Interest rate swap.

8.135 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/swaption.hpp File Reference

8.135.1 Detailed Description

Swaption class.

`#include <ql/numericalmethod.hpp>`

`#include <ql/option.hpp>`

`#include <ql/Instruments/simpleswap.hpp>`

Include dependency graph for swaption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Swaption
  Swaption class
- class Swaption::arguments
  Arguments for swaption calculation
- class Swaption::results
  Results from swaption calculation

8.136 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/vanillaoption.h File Reference

8.136.1 Detailed Description

Vanilla option on a single asset.

`#include <ql/Instruments/oneassetstrikedoption.hpp>`

Include dependency graph for vanillaoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class VanillaOption

  Vanilla option (no discrete dividends, no barriers) on a single asset.

- class VanillaOption::engine

  Vanilla option engine base class.

8.137 builddir/build/BUILD/QuantLib-0.3.11/ql/Instruments/zerocouponbon File Reference

8.137.1 Detailed Description

zero-coupon bond

`#include <ql/Instruments/bond.hpp>`

Include dependency graph for zerocouponbond.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ZeroCouponBond
  zero-coupon bond

8.138 builddir/build/BUILD/QuantLib-0.3.11/ql/interestrate.hpp File Reference

8.138.1 Detailed Description

Instrument rate class.

`#include <ql/types.hpp>`

`#include <ql/daycounter.hpp>`

Include dependency graph for interestrate.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InterestRate

  Concrete interest rate class.

Enumerations

- enum **Compounding** { QuantLib::Simple = 0, QuantLib::Compounded = 1, QuantLib::Continuous = 2, QuantLib::SimpleThenCompounded }

  Interest rate coumpounding rule.

8.139 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/binomialtree.hpp File Reference

8.139.1 Detailed Description

Binomial tree class.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/Lattices/tree.hpp>`

Include dependency graph for binomialtree.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BinomialTree

  Binomial tree base class.

- class EqualProbabilitiesBinomialTree

  Base class for equal probabilities binomial tree.

- class EqualJumpsBinomialTree

  Base class for equal jumps binomial tree.

- class JarrowRudd

  Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

- class CoxRossRubinstein

  Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

- class AdditiveEQPBinomialTree

  Additive equal probabilities binomial tree.

- class Trigeorgis

  Trigeorgis (additive equal jumps) binomial tree

- class Tian

  Tian tree: third moment matching, multiplicative approach

- class LeisenReimer

  Leisen & Reimer tree: multiplicative approach.

8.140 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/bsmlattice.hpp File Reference

8.140.1 Detailed Description

Binomial trees under the BSM model.

`#include <ql/Lattices/binomialtree.hpp>`

`#include <ql/Lattices/lattice1d.hpp>`

Include dependency graph for bsmlattice.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackScholesLattice

  Simple binomial lattice approximating the Black-Scholes model.

8.141 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice.hpp File Reference

8.141.1 Detailed Description

Lattice method class.

`#include <ql/numericalmethod.hpp>`

`#include <ql/discretizedasset.hpp>`

`#include <ql/Patterns/curiouslyrecurring.hpp>`

Include dependency graph for lattice.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Lattice

  Lattice-method base class.

8.142 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice1d.hpp File Reference

8.142.1 Detailed Description

One-dimensional lattice class.

`#include <ql/Lattices/lattice.hpp>`

Include dependency graph for lattice1d.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Lattice1D

  One-dimensional lattice.

8.143 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/lattice2d.hpp File Reference

8.143.1 Detailed Description

Two-dimensional lattice class.

`#include <ql/Lattices/lattice.hpp>`

`#include <ql/Lattices/trinomialtree.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for lattice2d.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Lattice2D

  Two-dimensional lattice.

8.144 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/tree.hpp File Reference

8.144.1 Detailed Description

Tree class.

`#include <ql/types.hpp>`

`#include <ql/Patterns/curiouslyrecurring.hpp>`

Include dependency graph for tree.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Tree

  Tree approximating a single-factor diffusion

8.145 builddir/build/BUILD/QuantLib-0.3.11/ql/Lattices/trinomialtree.hpp File Reference

8.145.1 Detailed Description

Trinomial tree class.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/Lattices/tree.hpp>`

`#include <ql/timegrid.hpp>`

Include dependency graph for trinomialtree.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TrinomialTree

  Recombining trinomial tree class.

8.146 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/array.hpp File Reference

8.146.1 Detailed Description

1-D array used in linear algebra.

```
#include <ql/types.hpp>
#include <ql/errors.hpp>
#include <ql/Utilities/disposable.hpp>
#include <boost/iterator/reverse_iterator.hpp>
#include <boost/scoped_array.hpp>
#include <functional>
#include <numeric>
#include <iomanip>
```

Include dependency graph for array.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Array

  1-D array used in linear algebra.

8.147 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/backwardflatinterpola File Reference

8.147.1 Detailed Description

backward-flat interpolation between discrete points

`#include <ql/Math/interpolation.hpp>`

`#include <vector>`

Include dependency graph for backwardflatinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class BackwardFlatInterpolation

  Backward-flat interpolation between discrete points.

- class BackwardFlat

  Backward-flat interpolation factory and traits.

8.148 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/beta.hpp File Reference

8.148.1 Detailed Description

Beta and beta incomplete functions.

`#include <ql/Math/gammadistribution.hpp>`

Include dependency graph for beta.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Real **QuantLib::betaFunction** (Real z, Real w)
- Real **QuantLib::betaContinuedFraction** (Real a, Real b, Real x, Real accuracy=1e-16, Integer maxIteration=100)
- Real QuantLib::incompleteBetaFunction (Real a, Real b, Real x, Real accuracy=1e-16, Integer maxIteration=100)

  Incomplete Beta function.

8.149 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bicubicsplineinterpol File Reference

8.149.1 Detailed Description

bicubic spline interpolation between discrete points

`#include <ql/Math/interpolation2D.hpp>`

`#include <ql/Math/cubicspline.hpp>`

Include dependency graph for bicubicsplineinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class BicubicSpline
- class Bicubic

  bicubic-spline interpolation factory

8.150 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bilinearinterpolation.hpp File Reference

8.150.1 Detailed Description

bilinear interpolation between discrete points

`#include <ql/Math/interpolation2D.hpp>`

Include dependency graph for bilinearinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class BilinearInterpolation
  bilinear interpolation between discrete points
- class Bilinear
  bilinear interpolation factory

8.151 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/binomialdistribution.hpp File Reference

8.151.1 Detailed Description

Binomial distribution.

`#include <ql/Math/factorial.hpp>`

`#include <ql/Math/beta.hpp>`

Include dependency graph for binomialdistribution.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BinomialDistribution

  Binomial probability distribution function.

- class CumulativeBinomialDistribution

  Cumulative binomial distribution function.

Functions

- Real **QuantLib::binomialCoefficientLn** (BigNatural n, BigNatural k)
- Real **QuantLib::binomialCoefficient** (BigNatural n, BigNatural k)
- Real QuantLib::PeizerPrattMethod2Inversion (Real z, BigNatural n)

8.152 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/bivariatenormaldistril File Reference

8.152.1 Detailed Description

bivariate cumulative normal distribution

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for bivariatenormaldistribution.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BivariateCumulativeNormalDistributionDr78

  Cumulative bivariate normal distribution function.

- class BivariateCumulativeNormalDistributionWe04DP

  Cumulative bivariate normal distibution function (West 2004).

Typedefs

- typedef BivariateCumulativeNormalDistributionWe04DP QuantLib::BivariateCumulative-NormalDistribution

  default bivariate implementation

8.153 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/chisquaredistribution.hpp File Reference

8.153.1 Detailed Description

Chi-square (central and non-central) distributions.

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for chisquaredistribution.hpp:

Namespaces

- namespace **QuantLib**

8.154 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/choleskydecompositio File Reference

8.154.1 Detailed Description

Cholesky decomposition.

`#include <ql/Math/matrix.hpp>`

Include dependency graph for choleskydecomposition.hpp:

Namespaces

- namespace **QuantLib**

8.155 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/comparison.hpp File Reference

8.155.1 Detailed Description

floating-point comparisons

`#include <ql/types.hpp>`

Include dependency graph for comparison.hpp:

Namespaces

- namespace **QuantLib**

Functions

- bool QuantLib::close (Real x, Real y)
- bool **QuantLib::close** (Real x, Real y, Size n)
- bool QuantLib::close_enough (Real x, Real y)
- bool **QuantLib::close_enough** (Real x, Real y, Size n)

8.156 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/convergencestatistics.hpp File Reference

8.156.1 Detailed Description

statistics tool with risk measures

`#include <ql/types.hpp>`

`#include <vector>`

Include dependency graph for convergencestatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ConvergenceStatistics

  statistics class with convergence table

8.157 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/cubicspline.hpp File Reference

8.157.1 Detailed Description

cubic spline interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
#include <ql/Utilities/null.hpp>
#include <vector>
```

Include dependency graph for cubicspline.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class CubicSpline

  Cubic spline interpolation between discrete points.

- class MonotonicCubicSpline

  Cubic spline with monotonicity constraint

- class NaturalCubicSpline

  Cubic spline with null second derivative at end points

- class NaturalMonotonicCubicSpline

  Natural cubic spline with monotonicity constraint.

- class Cubic

  cubic-spline interpolation factory and traits

8.158 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/discrepancystatistics.h File Reference

8.158.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

`#include <ql/Math/sequencestatistics.hpp>`

Include dependency graph for discrepancystatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DiscrepancyStatistics

  Statistic tool for sequences with discrepancy calculation.

8.159 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/errorfunction.hpp File Reference

8.159.1 Detailed Description

Error function.

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for errorfunction.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ErrorFunction

  Error function

8.160 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/extrapolation.hpp File Reference

8.160.1 Detailed Description

class-wide extrapolation settings

`#include <ql/qldefines.hpp>`

Include dependency graph for extrapolation.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Extrapolator

  base class for classes possibly allowing extrapolation

8.161 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/factorial.hpp File Reference

8.161.1 Detailed Description

Factorial numbers calculator.

`#include <ql/types.hpp>`

Include dependency graph for factorial.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Factorial

  Factorial numbers calculator

8.162 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/forwardflatinterpolati File Reference

8.162.1 Detailed Description

forward-flat interpolation between discrete points

`#include <ql/Math/interpolation.hpp>`

`#include <vector>`

Include dependency graph for forwardflatinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class ForwardFlatInterpolation

  Forward-flat interpolation between discrete points.

- class ForwardFlat

  Forward-flat interpolation factory and traits.

8.163 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/functional.hpp File Reference

8.163.1 Detailed Description

functionals and combinators not included in the STL

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for functional.hpp:

Namespaces

- namespace **QuantLib**

Functions

- template<class F, class R> clipped_function< F, R > **QuantLib::clip** (const F &f, const R &r)
- template<class F, class G> composed_function< F, G > **QuantLib::compose** (const F &f, const G &g)

8.164 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gammadistribution.hp File Reference

8.164.1 Detailed Description

Gamma distribution.

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for gammadistribution.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GammaFunction

  Gamma function class.

8.165 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianorthogonalpo File Reference

8.165.1 Detailed Description

orthogonal polynomials for gaussian quadratures

`#include <ql/types.hpp>`

Include dependency graph for gaussianorthogonalpolynomial.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GaussianOrthogonalPolynomial
  orthogonal polynomial for Gaussian quadratures
- class GaussLaguerrePolynomial
  Gauss-Laguerre polynomial.
- class GaussHermitePolynomial
  Gauss-Hermite polynomial.
- class GaussJacobiPolynomial
  Gauss-Jacobi polynomial.
- class GaussHyperbolicPolynomial
  Gauss hyperbolic polynomial.

8.166 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianquadratures.hpp File Reference

8.166.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures.

`#include <ql/Math/array.hpp>`

`#include <ql/Math/gaussianorthogonalpolynomial.hpp>`

Include dependency graph for gaussianquadratures.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GaussianQuadrature
  Integral of a 1-dimensional function using the Gauss quadratures method.
- class GaussLaguerreIntegration
  generalized Gauss-Laguerre integration
- class GaussHermiteIntegration
  generalized Gauss-Hermite integration
- class GaussJacobiIntegration
  Gauss-Jacobi integration.
- class GaussHyperbolicIntegration
  Gauss-Hyperbolic integration.
- class GaussLegendreIntegration
  Gauss-Legendre integration.
- class GaussChebyshevIntegration
  Gauss-Chebyshev integration.
- class GaussChebyshev2thIntegration
  Gauss-Chebyshev integration second kind.
- class GaussGegenbauerIntegration
  Gauss-Gegenbauer integration.
- class TabulatedGaussLegendre
  tabulated Gauss-Legendre quadratures

8.167 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/gaussianstatistics.hpp File Reference

8.167.1 Detailed Description

statistics tool for gaussian-assumption risk measures

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for gaussianstatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GaussianStatistics

  Statistics tool for gaussian-assumption risk measures.

- class StatsHolder

  Helper class for precomputed distributions.

8.168 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/generalstatistics.hpp File Reference

8.168.1 Detailed Description

statistics tool

```
#include <ql/Utilities/null.hpp>
#include <ql/errors.hpp>
#include <vector>
#include <utility>
```

Include dependency graph for generalstatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GeneralStatistics

  Statistics tool.

8.169 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/incompletegamma.hpp File Reference

8.169.1 Detailed Description

Incomplete Gamma function.

```
#include <ql/errors.hpp>
#include <ql/types.hpp>
#include <functional>
```

Include dependency graph for incompletegamma.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Real QuantLib::incompleteGammaFunction (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)

  Incomplete Gamma function.

- Real **QuantLib::incompleteGammaFunctionSeriesRepr** (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)
- Real **QuantLib::incompleteGammaFunctionContinuedFractionRepr** (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)

8.170 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/incrementalstatistics.h File Reference

8.170.1 Detailed Description

statistics tool based on incremental accumulation

`#include <ql/Utilities/null.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for incrementalstatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class IncrementalStatistics

  Statistics tool based on incremental accumulation.

8.171 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation.hpp File Reference

8.171.1 Detailed Description

base class for 1-D interpolations

`#include <ql/Patterns/bridge.hpp>`

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

Include dependency graph for interpolation.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InterpolationImpl
  abstract base class for interpolation implementations
- class Interpolation
  base class for 1-D interpolations.
- class Interpolation::templateImpl
  basic template implementation

8.172 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/interpolation2D.hpp File Reference

8.172.1 Detailed Description

abstract base classes for 2-D interpolations

`#include <ql/Patterns/bridge.hpp>`

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

Include dependency graph for interpolation2D.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Interpolation2DImpl
  abstract base class for 2-D interpolation implementations
- class Interpolation2D
  base class for 2-D interpolations.
- class Interpolation2D::templateImpl
  basic template implementation

8.173 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/kronrodintegral.hpp File Reference

8.173.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

`#include <ql/Utilities/null.hpp>`

Include dependency graph for kronrodintegral.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class KronrodIntegral

  Integral of a 1-dimensional function using the Gauss-Kronrod method.

8.174 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/lexicographicalview.h File Reference

8.174.1 Detailed Description

Lexicographical 2-D view of a contiguous set of data.

`#include <ql/Utilities/steppingiterator.hpp>`

`#include <boost/iterator/reverse_iterator.hpp>`

Include dependency graph for lexicographicalview.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LexicographicalView

  Lexicographical 2-D view of a contiguous set of data.

8.175 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/linearinterpolation.hp File Reference

8.175.1 Detailed Description

linear interpolation between discrete points

`#include <ql/Math/interpolation.hpp>`

`#include <vector>`

Include dependency graph for linearinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class LinearInterpolation

  Linear interpolation between discrete points

- class Linear

  Linear interpolation factory and traits.

8.176 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/loglinearinterpolation File Reference

8.176.1 Detailed Description

log-linear interpolation between discrete points

`#include <ql/Math/linearinterpolation.hpp>`

`#include <ql/Utilities/dataformatters.hpp>`

Include dependency graph for loglinearinterpolation.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class LogLinearInterpolation
- class LogLinear

  log-linear interpolation factory and traits

8.177 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/matrix.hpp File Reference

8.177.1 Detailed Description

matrix used in linear algebra.

`#include <ql/Math/array.hpp>`

`#include <ql/Utilities/steppingiterator.hpp>`

Include dependency graph for matrix.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Matrix

  Matrix used in linear algebra.

8.178 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/multicubicspline.hpp File Reference

8.178.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

`#include <functional>`

`#include <vector>`

Include dependency graph for multicubicspline.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class MultiCubicSpline

Typedefs

- typedef std::vector< std::vector< Real > > **QuantLib::detail::SplineGrid**
- typedef DataTable< Real > **QuantLib::detail::base_data_table**
- typedef Data< std::vector< Real >, EmptyArg > **QuantLib::detail::base_data**
- typedef Point< Real, EmptyArg > **QuantLib::detail::base_arg_type**
- typedef Point< Real, EmptyRes > **QuantLib::detail::base_return_type**
- typedef Point< Size, EmptyDim > **QuantLib::detail::base_dimensions**
- typedef Point< base_data_table, EmptyRes > **QuantLib::detail::base_output_data**
- typedef base_cubic_spline **QuantLib::detail::cubic_spline_01**
- typedef n_cubic_spline< cubic_spline_01 > **QuantLib::detail::cubic_spline_02**
- typedef n_cubic_spline< cubic_spline_02 > **QuantLib::detail::cubic_spline_03**
- typedef n_cubic_spline< cubic_spline_03 > **QuantLib::detail::cubic_spline_04**
- typedef n_cubic_spline< cubic_spline_04 > **QuantLib::detail::cubic_spline_05**
- typedef n_cubic_spline< cubic_spline_05 > **QuantLib::detail::cubic_spline_06**
- typedef n_cubic_spline< cubic_spline_06 > **QuantLib::detail::cubic_spline_07**
- typedef n_cubic_spline< cubic_spline_07 > **QuantLib::detail::cubic_spline_08**
- typedef n_cubic_spline< cubic_spline_08 > **QuantLib::detail::cubic_spline_09**
- typedef n_cubic_spline< cubic_spline_09 > **QuantLib::detail::cubic_spline_10**
- typedef n_cubic_spline< cubic_spline_10 > **QuantLib::detail::cubic_spline_11**
- typedef n_cubic_spline< cubic_spline_11 > **QuantLib::detail::cubic_spline_12**
- typedef n_cubic_spline< cubic_spline_12 > **QuantLib::detail::cubic_spline_13**
- typedef n_cubic_spline< cubic_spline_13 > **QuantLib::detail::cubic_spline_14**
- typedef n_cubic_spline< cubic_spline_14 > **QuantLib::detail::cubic_spline_15**
- typedef base_cubic_splint **QuantLib::detail::cubic_splint_01**

- typedef n_cubic_splint< cubic_splint_01 > **QuantLib::detail::cubic_splint_02**
- typedef n_cubic_splint< cubic_splint_02 > **QuantLib::detail::cubic_splint_03**
- typedef n_cubic_splint< cubic_splint_03 > **QuantLib::detail::cubic_splint_04**
- typedef n_cubic_splint< cubic_splint_04 > **QuantLib::detail::cubic_splint_05**
- typedef n_cubic_splint< cubic_splint_05 > **QuantLib::detail::cubic_splint_06**
- typedef n_cubic_splint< cubic_splint_06 > **QuantLib::detail::cubic_splint_07**
- typedef n_cubic_splint< cubic_splint_07 > **QuantLib::detail::cubic_splint_08**
- typedef n_cubic_splint< cubic_splint_08 > **QuantLib::detail::cubic_splint_09**
- typedef n_cubic_splint< cubic_splint_09 > **QuantLib::detail::cubic_splint_10**
- typedef n_cubic_splint< cubic_splint_10 > **QuantLib::detail::cubic_splint_11**
- typedef n_cubic_splint< cubic_splint_11 > **QuantLib::detail::cubic_splint_12**
- typedef n_cubic_splint< cubic_splint_12 > **QuantLib::detail::cubic_splint_13**
- typedef n_cubic_splint< cubic_splint_13 > **QuantLib::detail::cubic_splint_14**
- typedef n_cubic_splint< cubic_splint_14 > **QuantLib::detail::cubic_splint_15**
- typedef detail::SplineGrid **QuantLib::SplineGrid**

8.179 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/normaldistribution.hp File Reference

8.179.1 Detailed Description

normal, cumulative and inverse cumulative distributions

`#include <ql/Math/errorfunction.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for normaldistribution.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class NormalDistribution
  Normal distribution function.
- class CumulativeNormalDistribution
  Cumulative normal distribution function.
- class InverseCumulativeNormal
  Inverse cumulative normal distribution function.
- class MoroInverseCumulativeNormal
  Moro Inverse cumulative normal distribution class.

Typedefs

- typedef NormalDistribution **QuantLib::GaussianDistribution**
- typedef InverseCumulativeNormal **QuantLib::InvCumulativeNormalDistribution**

8.180 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/poissondistribution.h File Reference

8.180.1 Detailed Description

Poisson distribution.

```
#include <ql/Math/factorial.hpp>
#include <ql/Math/incompletegamma.hpp>
```

Include dependency graph for poissondistribution.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class PoissonDistribution
  Normal distribution function.
- class CumulativePoissonDistribution
  Cumulative Poisson distribution function.
- class InverseCumulativePoisson
  Inverse cumulative Poisson distribution function.

8.181 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/primenumbers.hpp File Reference

8.181.1 Detailed Description

Prime numbers calculator.

`#include <ql/types.hpp>`

`#include <vector>`

Include dependency graph for primenumbers.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class PrimeNumbers

  Prime numbers calculator.

8.182 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/pseudosqrt.hpp File Reference

8.182.1 Detailed Description

pseudo square root of a real symmetric matrix

`#include <ql/Math/matrix.hpp>`

Include dependency graph for pseudosqrt.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct SalvagingAlgorithm

  algorithm used for matricial pseudo square root

8.183 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/riskstatistics.hpp File Reference

8.183.1 Detailed Description

empirical-distribution risk measures

`#include <ql/Math/functional.hpp>`

`#include <ql/Math/generalstatistics.hpp>`

`#include <ql/Math/gaussianstatistics.hpp>`

Include dependency graph for riskstatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GenericRiskStatistics

  empirical-distribution risk measures

Typedefs

- typedef GaussianStatistics< GenericRiskStatistics< GeneralStatistics > > QuantLib::RiskStatistics

  default risk measures tool

8.184 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/rounding.hpp File Reference

8.184.1 Detailed Description

Rounding implementation.

`#include <ql/types.hpp>`

Include dependency graph for rounding.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Rounding
  basic rounding class
- class UpRounding
  Up-rounding.
- class DownRounding
  Down-rounding.
- class ClosestRounding
  Closest rounding.
- class CeilingTruncation
  Ceiling truncation.
- class FloorTruncation
  Floor truncation.

8.185 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/sampledcurve.hpp File Reference

8.185.1 Detailed Description

a class that contains a sampled curve

`#include <ql/Math/array.hpp>`

`#include <ql/types.hpp>`

Include dependency graph for sampledcurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SampledCurve

  This class contains a sampled curve.

Typedefs

- typedef SampledCurve **QuantLib::SampledCurveSet**

8.186 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/segmentintegral.hpp File Reference

8.186.1 Detailed Description

Integral of a one-dimensional function.

`#include <ql/types.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for segmentintegral.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SegmentIntegral

  Integral of a one-dimensional function.

8.187 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/sequencestatistics.hpp File Reference

8.187.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

`#include <ql/Math/statistics.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for sequencestatistics.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SequenceStatistics

  Statistics analysis of N-dimensional (sequence) data.

Defines

- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID**(METHOD)
- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE**(METHOD)

8.187.2 Define Documentation

8.187.2.1 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)

Value:

```
template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }
```

8.187.2.2 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)

Value:

```
template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD(Real x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
```

```
        return results_; \
    }
```

8.188 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/simpsonintegral.hpp File Reference

8.188.1 Detailed Description

integral of a one-dimensional function

`#include <ql/Math/trapezoidintegral.hpp>`

Include dependency graph for simpsonintegral.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SimpsonIntegral

  Integral of a one-dimensional function.

8.189 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/statistics.hpp File Reference

8.189.1 Detailed Description

statistics tool with risk measures

`#include <ql/Math/generalstatistics.hpp>`

Include dependency graph for statistics.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef GeneralStatistics QuantLib::Statistics
  default statistics tool

8.190 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/svd.hpp File Reference

8.190.1 Detailed Description

singular value decomposition

`#include <ql/Math/matrix.hpp>`

Include dependency graph for svd.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SVD

  Singular value decomposition.

8.191 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/symmetriceigenvalues File Reference

8.191.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

`#include <ql/Math/symmetricschurdecomposition.hpp>`

Include dependency graph for symmetriceigenvalues.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Disposable< Array > **QuantLib::SymmetricEigenvalues** (Matrix &s)
- Disposable< Matrix > **QuantLib::SymmetricEigenvectors** (Matrix &s)

8.192 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/symmetricschurdecom File Reference

8.192.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

`#include <ql/Math/matrix.hpp>`

Include dependency graph for symmetricschurdecomposition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SymmetricSchurDecomposition
  symmetric threshold Jacobi algorithm.

8.193 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/tqreigendecompositio File Reference

8.193.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

`#include <ql/Math/array.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for tqreigendecomposition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TqrEigenDecomposition

  tridiag. QR eigen decomposition with explicite shift aka Wilkinson

8.194 builddir/build/BUILD/QuantLib-0.3.11/ql/Math/trapezoidintegral.hpp File Reference

8.194.1 Detailed Description

integral of a one-dimensional function

`#include <ql/Utilities/null.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for trapezoidintegral.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TrapezoidIntegral

  Integral of a one-dimensional function.

8.195 builddir/build/BUILD/QuantLib-0.3.11/ql/money.hpp File Reference

8.195.1 Detailed Description

cash amount in a given currency

`#include <ql/currency.hpp>`

Include dependency graph for money.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Money
  amount of cash

8.196 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/brownianbridge.hpp File Reference

8.196.1 Detailed Description

Browian bridge.

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

`#include <ql/stochasticprocess.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for brownianbridge.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BrownianBridge

  Builds Wiener process paths using Gaussian variates.

8.197 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/getcovariance.hpp File Reference

8.197.1 Detailed Description

Covariance matrix calculation.

`#include <ql/Math/matrix.hpp>`

`#include <ql/Utilities/dataformatters.hpp>`

Include dependency graph for getcovariance.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CovarianceDecomposition

Functions

- template<class DataIterator> Disposable< Matrix > QuantLib::getCovariance (DataIterator volBegin, DataIterator volEnd, const Matrix &corr, Real tolerance=1.0e-12)

8.198 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctraits.hpp File Reference

8.198.1 Detailed Description

Monte Carlo policies.

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/MonteCarlo/multipathgenerator.hpp>
#include <ql/MonteCarlo/pathpricer.hpp>
#include <ql/RandomNumbers/rngtraits.hpp>
```

Include dependency graph for mctraits.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct SingleVariate
  default Monte Carlo traits for single-variate models
- struct SingleAsset
- struct MultiVariate
  default Monte Carlo traits for multi-variate models
- struct MultiAsset

8.199 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/mctypedefs.hpp File Reference

8.199.1 Detailed Description

Default choices for template instantiations.

`#include <ql/MonteCarlo/montecarlomodel.hpp>`

Include dependency graph for mctypedefs.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef MonteCarloModel< SingleVariate< PseudoRandom > > QuantLib::OneFactorMonteCarloOption

  default choice for one-factor Monte Carlo model.

- typedef MonteCarloModel< MultiVariate< PseudoRandom > > QuantLib::MultiFactorMonteCarloOption

  default choice for multi-factor Monte Carlo model.

8.200 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/montecarlomodel.hpp File Reference

8.200.1 Detailed Description

General purpose Monte Carlo model.

```
#include <ql/MonteCarlo/mctraits.hpp>
#include <ql/Math/statistics.hpp>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for montecarlomodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MonteCarloModel

  General purpose Monte Carlo model for path samples.

8.201 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/multipath.hpp File Reference

8.201.1 Detailed Description

Correlated multiple asset paths.

`#include <ql/MonteCarlo/path.hpp>`

Include dependency graph for multipath.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MultiPath

  Correlated multiple asset paths.

8.202 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/multipathgenerator.hpp File Reference

8.202.1 Detailed Description

Generates a multi path from a random-array generator.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/Processes/stochasticprocessarray.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

`#include <ql/Math/pseudosqrt.hpp>`

Include dependency graph for multipathgenerator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MultiPathGenerator

  Generates a multipath from a random number generator.

8.203 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/path.hpp File Reference

8.203.1 Detailed Description

single factor random walk

`#include <ql/timegrid.hpp>`

`#include <ql/Math/array.hpp>`

Include dependency graph for path.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Path

8.204 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/pathgenerator.hpp File Reference

8.204.1 Detailed Description

Generates random paths using a sequence generator.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/MonteCarlo/brownianbridge.hpp>`

Include dependency graph for pathgenerator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class PathGenerator

  Generates random paths using a sequence generator.

8.205 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/pathpricer.hpp File Reference

8.205.1 Detailed Description

base class for single-path pricers

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <functional>
```

Include dependency graph for pathpricer.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class PathPricer

  base class for path pricers

8.206 builddir/build/BUILD/QuantLib-0.3.11/ql/MonteCarlo/sample.hpp File Reference

8.206.1 Detailed Description

weighted sample

`#include <ql/types.hpp>`

Include dependency graph for sample.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct Sample
  weighted sample

8.207 builddir/build/BUILD/QuantLib-0.3.11/ql/numericalmethod.hpp File Reference

8.207.1 Detailed Description

Numerical method class.

`#include <ql/timegrid.hpp>`

`#include <ql/Math/array.hpp>`

Include dependency graph for numericalmethod.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class NumericalMethod

  Numerical method (tree, finite-differences) base class.

8.208 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/armijo.hpp File Reference

8.208.1 Detailed Description

Armijo line-search class.

`#include <ql/Optimization/linesearch.hpp>`

Include dependency graph for armijo.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ArmijoLineSearch
  Armijo line search.

8.209 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/conjugategrad File Reference

8.209.1 Detailed Description

Conjugate gradient optimization method.

`#include <ql/Optimization/armijo.hpp>`

Include dependency graph for conjugategradient.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ConjugateGradient

  Multi-dimensional Conjugate Gradient class.

8.210 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/constraint.hpp File Reference

8.210.1 Detailed Description

Abstract constraint class.

`#include <ql/Math/array.hpp>`

`#include <ql/Patterns/bridge.hpp>`

Include dependency graph for constraint.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ConstraintImpl

  Base class for constraint implementations.

- class Constraint

  Base constraint class.

- class NoConstraint

  No constraint.

- class PositiveConstraint

  Constraint imposing positivity to all arguments

- class BoundaryConstraint

  Constraint imposing all arguments to be in [low,high]

- class CompositeConstraint

  Constraint enforcing both given sub-constraints

8.211 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/costfunction.h File Reference

8.211.1 Detailed Description

Optimization cost function class.

`#include <ql/Math/array.hpp>`

Include dependency graph for costfunction.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CostFunction

  Cost function abstract class for optimization problem.

8.212 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/criteria.hpp File Reference

8.212.1 Detailed Description

Optimization criteria class.

`#include <ql/types.hpp>`

Include dependency graph for criteria.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class EndCriteria

  Criteria to end optimization process.

8.213 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/leastsquare.hpp File Reference

8.213.1 Detailed Description

Least square cost function.

`#include <ql/Math/matrix.hpp>`

`#include <ql/Optimization/conjugategradient.hpp>`

Include dependency graph for leastsquare.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LeastSquareProblem

  Base class for least square problem.

- class LeastSquareFunction

  Cost function for least-square problems.

- class NonLinearLeastSquare

  Non-linear least-square method.

8.214 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/linesearch.hp File Reference

8.214.1 Detailed Description

Line search abstract class.

`#include <ql/Optimization/problem.hpp>`

Include dependency graph for linesearch.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LineSearch

  Base class for line search.

8.215 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/method.hpp File Reference

8.215.1 Detailed Description

Abstract optimization method class.

`#include <ql/Optimization/constraint.hpp>`

`#include <ql/Optimization/costfunction.hpp>`

`#include <ql/Optimization/criteria.hpp>`

Include dependency graph for method.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OptimizationMethod

  Abstract class for constrained optimization method.

8.216 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/problem.hpp File Reference

8.216.1 Detailed Description

Abstract optimization class.

`#include <ql/Optimization/method.hpp>`

Include dependency graph for problem.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Problem

  Constrained optimization problem.

8.217 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/simplex.hpp File Reference

8.217.1 Detailed Description

Simplex optimization method.

`#include <ql/Optimization/problem.hpp>`

`#include <vector>`

Include dependency graph for simplex.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Simplex

  Multi-dimensional simplex class.

8.218 builddir/build/BUILD/QuantLib-0.3.11/ql/Optimization/steepestdesce File Reference

8.218.1 Detailed Description

Steepest descent optimization method.

`#include <ql/Optimization/armijo.hpp>`

Include dependency graph for steepestdescent.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SteepestDescent

  Multi-dimensional steepest-descent class.

8.219 builddir/build/BUILD/QuantLib-0.3.11/ql/option.hpp File Reference

8.219.1 Detailed Description

Base option class.

`#include <ql/instrument.hpp>`

`#include <ql/payoff.hpp>`

`#include <ql/exercise.hpp>`

Include dependency graph for option.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Option
  base option class
- class Option::arguments
- class Greeks
  additional option results
- class MoreGreeks
  more additional option results

Functions

- std::ostream & **QuantLib::operator**<< (std::ostream &out, Option::Type type)

8.220 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/bridge.hpp File Reference

8.220.1 Detailed Description

bridge pattern (a.k.a. handle-body idiom)

`#include <ql/qldefines.hpp>`

`#include <boost/shared_ptr.hpp>`

Include dependency graph for bridge.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Bridge

  The Bridge pattern made explicit.

8.221 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/composite.hpp File Reference

8.221.1 Detailed Description

composite pattern

`#include <ql/qldefines.hpp>`

`#include <boost/shared_ptr.hpp>`

`#include <list>`

Include dependency graph for composite.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Composite

  Composite pattern.

8.222 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/curiouslyrecurring. File Reference

8.222.1 Detailed Description

Curiously recurring template pattern.

`#include <ql/qldefines.hpp>`

Include dependency graph for curiouslyrecurring.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CuriouslyRecurringTemplate
  Support for the curiously recurring template pattern.

8.223 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/lazyobject.hpp File Reference

8.223.1 Detailed Description

framework for calculation on demand and result caching

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for lazyobject.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LazyObject

  Framework for calculation on demand and result caching.

8.224 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/observable.hpp File Reference

8.224.1 Detailed Description

observer/observable pattern

```
#include <ql/qldefines.hpp>
#include <boost/shared_ptr.hpp>
#include <list>
```

Include dependency graph for observable.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Observable

  Object that notifies its changes to a set of observables.

- class Observer

  Object that gets notified when a given observable changes.

8.225 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/singleton.hpp File Reference

8.225.1 Detailed Description

basic support for the singleton pattern

```
#include <ql/types.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/noncopyable.hpp>
#include <map>
```

Include dependency graph for singleton.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Singleton

  Basic support for the singleton pattern.

8.226 builddir/build/BUILD/QuantLib-0.3.11/ql/Patterns/visitor.hpp File Reference

8.226.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

`#include <ql/qldefines.hpp>`

Include dependency graph for visitor.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AcyclicVisitor

  degenerate base class for the Acyclic Visitor pattern

- class Visitor

  Visitor for a specific class

8.227 builddir/build/BUILD/QuantLib-0.3.11/ql/payoff.hpp File Reference

8.227.1 Detailed Description

Option payoff classes.

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for payoff.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Payoff

  Base class for option payoffs.

8.228 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/discretegeometricaso File Reference

8.228.1 Detailed Description

Discrete Geometric Average Strike Option.

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for discretegeometricaso.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DiscreteGeometricASO

  Discrete geometric average-strike Asian option (European style).

8.229 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mccliquetoption.hpp File Reference

8.229.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mccliquetoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McCliquetOption
  simple example of Monte Carlo pricer

8.230 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcdiscretearithmeti File Reference

8.230.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McDiscreteArithmeticASO

  example of Monte Carlo pricer using a control variate.

8.231 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mceverest.hpp File Reference

8.231.1 Detailed Description

Everest-type option pricer

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for mceverest.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McEverest

  Everest-type option pricer.

8.232 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mchimalaya.hpp File Reference

8.232.1 Detailed Description

Himalayan-type option pricer.

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for mchimalaya.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McHimalaya

  Himalayan-type option pricer.

8.233 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcmaxbasket.hpp File Reference

8.233.1 Detailed Description

Max Basket Monte Carlo pricer.

```
#include <ql/Pricers/mcpricer.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcmaxbasket.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McMaxBasket

  simple example of multi-factor Monte Carlo pricer

8.234 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcpagoda.hpp File Reference

8.234.1 Detailed Description

Roofed multi asset Asian option.

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for mcpagoda.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McPagoda
  roofed Asian option

8.235 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcperformanceoptio File Reference

8.235.1 Detailed Description

Performance option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcperformanceoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McPerformanceOption

  Performance option computed using Monte Carlo simulation.

8.236 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/mcpricer.hpp File Reference

8.236.1 Detailed Description

base class for Monte Carlo pricers

`#include <ql/MonteCarlo/montecarlomodel.hpp>`

Include dependency graph for mcpricer.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McPricer
  base class for Monte Carlo pricers

8.237 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricers/singleassetoption.hp File Reference

8.237.1 Detailed Description

common code for option evaluation

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for singleassetoption.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SingleAssetOption

  Black-Scholes-Merton option.

8.238 builddir/build/BUILD/QuantLib-0.3.11/ql/pricingengine.hpp File Reference

8.238.1 Detailed Description

Base class for pricing engines.

`#include <ql/argsandresults.hpp>`

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for pricingengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class PricingEngine
  interface for pricing engines
- class GenericEngine
  template base class for option pricing engines

8.239 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/americanpayoffatexpiry.hpp File Reference

8.239.1 Detailed Description

Analytical formulae for american exercise with payoff at expiry.

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for americanpayoffatexpiry.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AmericanPayoffAtExpiry

8.240 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/americanpayoffathit.hpp File Reference

8.240.1 Detailed Description

Analytical formulae for american exercise with payoff at hit.

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for americanpayoffathit.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AmericanPayoffAtHit

8.241 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference

8.241.1 Detailed Description

Analytic engine for continuous geometric average price Asian.

`#include <ql/Instruments/asianoption.hpp>`

Include dependency graph for analytic_cont_geom_av_price.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticContinuousGeometricAveragePriceAsianEngine

  Pricing engine for European continuous geometric average price Asian.

8.242 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference

8.242.1 Detailed Description

Analytic engine for discrete geometric average price Asian.

`#include <ql/Instruments/asianoption.hpp>`

Include dependency graph for analytic_discr_geom_av_price.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticDiscreteGeometricAveragePriceAsianEngine

  Pricing engine for European discrete geometric average price Asian.

8.243 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference

8.243.1 Detailed Description

Monte Carlo engine for discrete arithmetic average price Asian.

`#include <ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>`

`#include <ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp>`

Include dependency graph for mc_discr_arith_av_price.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCDiscreteArithmeticAPEngine

  Monte Carlo pricing engine for discrete arithmetic average price Asian.

8.244 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference

8.244.1 Detailed Description

Monte Carlo engine for discrete geometric average price Asian.

`#include <ql/PricingEngines/Asian/mcdiscreteasianengine.hpp>`

`#include <ql/Volatilities/blackconstantvol.hpp>`

`#include <ql/Volatilities/blackvariancecurve.hpp>`

Include dependency graph for mc_discr_geom_av_price.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCDiscreteGeometricAPEngine

  Monte Carlo pricing engine for discrete geometric average price Asian.

8.245 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Asian/mcdiscreteasianengine.hpp File Reference

8.245.1 Detailed Description

Monte Carlo pricing engine for discrete average Asians.

```
#include <ql/PricingEngines/mcsimulation.hpp>
#include <ql/Instruments/asianoption.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdiscreteasianengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCDiscreteAveragingAsianEngine

  Pricing engine for discrete average Asians using Monte Carlo simulation.

8.246 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Barrier/analyticbarrierengine.hpp File Reference

8.246.1 Detailed Description

Analytic barrier option engines.

`#include <ql/Instruments/barrieroption.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for analyticbarrierengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticBarrierEngine

  Pricing engine for barrier options using analytical formulae.

8.247 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

8.247.1 Detailed Description

Monte Carlo barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
#include <ql/PricingEngines/mcsimulation.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcbarrierengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCBarrierEngine

  Pricing engine for barrier options using Monte Carlo simulation.

8.248 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Basket/mcamericanbasketengine.hpp File Reference

8.248.1 Detailed Description

Least-square Monte Carlo engines.

`#include <ql/Instruments/basketoption.hpp>`

`#include <ql/MonteCarlo/mctraits.hpp>`

Include dependency graph for mcamericanbasketengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCAmericanBasketEngine
  least-square Monte Carlo engine

8.249 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

8.249.1 Detailed Description

European basket MC Engine.

```
#include <ql/Instruments/basketoption.hpp>
#include <ql/PricingEngines/mcsimulation.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcbasketengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCBasketEngine

  Pricing engine for basket options using Monte Carlo simulation.

8.250 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Basket/stulzengine.hpp File Reference

8.250.1 Detailed Description

2D European Basket formulae, due to Stulz (1982)

`#include <ql/Instruments/basketoption.hpp>`

Include dependency graph for stulzengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class StulzEngine

  Pricing engine for 2D European Baskets.

8.251 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/blackformula.hpp File Reference

8.251.1 Detailed Description

Black formula.

`#include <ql/Instruments/payoffs.hpp>`

Include dependency graph for blackformula.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackFormula

  Black-formula calculator.

8.252 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/blackmodel.hpp File Reference

8.252.1 Detailed Description

Abstract class for Black-type models (market models).

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/quote.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for blackmodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackModel

  Black-model for vanilla interest-rate derivatives.

8.253 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference

8.253.1 Detailed Description

Analytic engine for caps/floors.

`#include <ql/Instruments/capfloor.hpp>`

`#include <ql/PricingEngines/genericmodelengine.hpp>`

`#include <ql/ShortRateModels/model.hpp>`

Include dependency graph for analyticcapfloorengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticCapFloorEngine

  Analytic engine for cap/floor.

8.254 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference

8.254.1 Detailed Description

Black-formula cap/floor engine.

`#include <ql/Instruments/capfloor.hpp>`

`#include <ql/PricingEngines/blackmodel.hpp>`

`#include <ql/PricingEngines/genericmodelengine.hpp>`

Include dependency graph for blackcapfloorengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackCapFloorEngine

  Black-formula cap/floor engine.

8.255 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference

8.255.1 Detailed Description

discretized cap/floor

`#include <ql/Instruments/capfloor.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for discretizedcapfloor.hpp:

Namespaces

- namespace **QuantLib**

8.256 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference

8.256.1 Detailed Description

Numerical lattice engine for cap/floors.

`#include <ql/Instruments/capfloor.hpp>`

`#include <ql/PricingEngines/latticeshortratemodelengine.hpp>`

Include dependency graph for treecapfloorengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TreeCapFloorEngine

  Numerical lattice engine for cap/floors.

8.257 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference

8.257.1 Detailed Description

Analytic Cliquet engine.

`#include <ql/Instruments/cliquetoption.hpp>`

Include dependency graph for analyticcliquetengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticCliquetEngine

  Pricing engine for Cliquet options using analytical formulae.

8.258 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference

8.258.1 Detailed Description

Analytic performance engine.

`#include <ql/Instruments/cliquetoption.hpp>`

Include dependency graph for analyticperformanceengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticPerformanceEngine

  Pricing engine for performance options using analytical formulae.

8.259 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Forward/forwardengine.hpp File Reference

8.259.1 Detailed Description

Forward (strike-resetting) option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/Volatilities/impliedvoltermstructure.hpp>
#include <ql/TermStructures/impliedtermstructure.hpp>
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for forwardengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ForwardOptionArguments

  Arguments for forward (strike-resetting) option calculation

- class ForwardEngine

  Forward engine base class.

8.260 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

8.260.1 Detailed Description

Forward (strike-resetting) performance option engines.

`#include <ql/PricingEngines/Forward/forwardengine.hpp>`

`#include <ql/stochasticprocess.hpp>`

Include dependency graph for forwardperformanceengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ForwardPerformanceEngine

  Forward performance engine.

8.261 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/genericmodelengine.hpp File Reference

8.261.1 Detailed Description

Generic option engine based on a model.

`#include <ql/pricingengine.hpp>`

Include dependency graph for genericmodelengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GenericModelEngine

  Base class for some pricing engine on a particular model.

8.262 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/greeks.hpp File Reference

8.262.1 Detailed Description

default greek calculations

`#include <ql/Processes/blackscholesprocess.hpp>`

Include dependency graph for greeks.hpp:

Namespaces

- namespace **QuantLib**

Functions

- Real QuantLib::blackScholesTheta (const boost::shared_ptr< BlackScholesProcess > &, Real value, Real delta, Real gamma)

  default theta calculation for Black-Scholes options

- Real QuantLib::defaultThetaPerDay (Real theta)

  default theta-per-day calculation

8.263 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

8.263.1 Detailed Description

Engine for a short-rate model specialized on a lattice.

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/PricingEngines/genericmodelengine.hpp>`

Include dependency graph for latticeshortratemodelengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LatticeShortRateModelEngine

  Engine for a short-rate model specialized on a lattice.

8.264 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/mcsimulation.hpp File Reference

8.264.1 Detailed Description

framework for Monte Carlo engines

`#include <ql/grid.hpp>`

`#include <ql/MonteCarlo/montecarlomodel.hpp>`

Include dependency graph for mcsimulation.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class McSimulation

  base class for Monte Carlo engines

8.265 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Quanto/quantoengine.hpp File Reference

8.265.1 Detailed Description

Quanto option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/TermStructures/quantotermstructure.hpp>
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for quantoengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class QuantoOptionArguments
  Arguments for quanto option calculation
- class QuantoOptionResults
  Results from quanto option calculation
- class QuantoEngine
  Quanto engine base class.

8.266 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference

8.266.1 Detailed Description

Black-formula swaption engine.

`#include <ql/Instruments/swaption.hpp>`

`#include <ql/PricingEngines/blackmodel.hpp>`

`#include <ql/PricingEngines/genericmodelengine.hpp>`

Include dependency graph for blackswaptionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackSwaptionEngine

  Black-formula swaption engine.

8.267 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/discretizedswaption.hpp File Reference

8.267.1 Detailed Description

Discretized swaption class.

`#include <ql/Instruments/swaption.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for discretizedswaption.hpp:

Namespaces

- namespace **QuantLib**

8.268 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference

8.268.1 Detailed Description

Swaption pricing engine for two-factor additive Gaussian Model G2++.

`#include <ql/PricingEngines/blackmodel.hpp>`

`#include <ql/PricingEngines/genericmodelengine.hpp>`

`#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>`

Include dependency graph for g2swaptionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class G2SwaptionEngine

  Swaption priced by means of the Black formula

8.269 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Swaption/jamshidianswaptionengine.hpp File Reference

8.269.1 Detailed Description

Swaption engine using Jamshidian's decomposition.

```
#include <ql/qldefines.hpp>
#include <ql/Instruments/swaption.hpp>
#include <ql/ShortRateModels/onefactormodel.hpp>
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for jamshidianswaptionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class JamshidianSwaptionEngine

  Jamshidian swaption engine.

8.270 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference

8.270.1 Detailed Description

Numerical lattice engine for swaptions.

`#include <ql/Instruments/swaption.hpp>`

`#include <ql/PricingEngines/latticeshortratemodelengine.hpp>`

Include dependency graph for treeswaptionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TreeSwaptionEngine

  Numerical lattice engine for swaptions.

8.271 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticdigitalamericanengine.hpp File Reference

8.271.1 Detailed Description

analytic digital American option engine

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for analyticdigitalamericanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticDigitalAmericanEngine

8.272 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticdividendeuropeanengine.hpp File Reference

8.272.1 Detailed Description

Analytic discrete-dividend European engine.

`#include <ql/Instruments/dividendvanillaoption.hpp>`

Include dependency graph for analyticdividendeuropeanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticDividendEuropeanEngine

  Analytic pricing engine for European options with discrete dividends.

8.273 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/analyticeuropeanengine.hpp File Reference

8.273.1 Detailed Description

Analytic European engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for analyticeuropeanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticEuropeanEngine

  Pricing engine for European vanilla options using analytical formulae.

8.274 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference

8.274.1 Detailed Description

analytic Heston-model engine

```
#include <ql/qldefines.hpp>
#include <ql/PricingEngines/genericmodelengine.hpp>
#include <ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>
#include <ql/Instruments/vanillaoption.hpp>
#include <ql/Math/gaussianquadratures.hpp>
#include <complex>
```

Include dependency graph for analytichestonengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AnalyticHestonEngine
  analytic Heston-model engine based on Fourier transform

8.275 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/baroneadesiwhaleyengine.hpp File Reference

8.275.1 Detailed Description

Barone-Adesi and Whaley approximation engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for baroneadesiwhaleyengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BaroneAdesiWhaleyApproximationEngine

8.276 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/batesengine.hpp File Reference

8.276.1 Detailed Description

analytic Bates model engine

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/TwoFactorModels/batesmodel.hpp>`

`#include <ql/PricingEngines/Vanilla/analytichestonengine.hpp>`

Include dependency graph for batesengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BatesEngine

  Bates model engines based on Fourier transform.

8.277 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

8.277.1 Detailed Description

Binomial option engine.

```
#include <ql/Lattices/binomialtree.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/TermStructures/flatforward.hpp>
#include <ql/Volatilities/blackconstantvol.hpp>
```

Include dependency graph for binomialengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BinomialVanillaEngine

  Pricing engine for vanilla options using binomial trees.

8.278 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/bjerksundstenslandengine.hpp File Reference

8.278.1 Detailed Description

Bjerksund and Stensland approximation engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for bjerksundstenslandengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BjerksundStenslandApproximationEngine

8.279 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/discretizedvanillaoption.hpp File Reference

8.279.1 Detailed Description

discretized vanilla option

```
#include <ql/discretizedasset.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Pricers/singleassetoption.hpp>
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:

Namespaces

- namespace **QuantLib**

8.280 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference

8.280.1 Detailed Description

Finite-differences American option engine.

`#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/FiniteDifferences/americancondition.hpp>`

Include dependency graph for fdamericanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDAmericanEngine

  Finite-differences pricing engine for American one asset options.

8.281 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdbermudanengine.hpp File Reference

8.281.1 Detailed Description

finite-difference Bermudan engine

`#include <ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp>`

Include dependency graph for fdbermudanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDBermudanEngine

  Finite-differences Bermudan engine.

8.282 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference

8.282.1 Detailed Description

american engine with discrete deterministic dividends

`#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>`

`#include <ql/FiniteDifferences/americancondition.hpp>`

Include dependency graph for fddividendamericanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDDividendAmericanEngine

  Finite-differences pricing engine for dividend American options.

8.283 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference

8.283.1 Detailed Description

base engine for option with dividends

`#include <ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp>`

Include dependency graph for fddividendengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDDividendEngine

  Base finite-differences pricing engine for dividend options.

8.284 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference

8.284.1 Detailed Description

finite-differences engine for European option with dividends

`#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>`

Include dependency graph for fddividendeuropeanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDDividendEuropeanEngine

  Finite-differences pricing engine for dividend European options.

8.285 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fddividendshoutengine.hpp File Reference

8.285.1 Detailed Description

base class for shout engine with dividends

`#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>`

`#include <ql/FiniteDifferences/shoutcondition.hpp>`

Include dependency graph for fddividendshoutengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDDividendShoutEngine

  Finite-differences shout engine with dividends.

8.286 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference

8.286.1 Detailed Description

Finite-difference European engine.

`#include <ql/Instruments/oneassetoption.hpp>`

`#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>`

Include dependency graph for fdeuropeanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDEuropeanEngine

  Pricing engine for European options using finite-differences.

8.287 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdmultiperiodengine.hpp File Reference

8.287.1 Detailed Description

base engine for options with events happening at specific times

`#include <ql/Instruments/dividendvanillaoption.hpp>`

`#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/Instruments/dividendschedule.hpp>`

Include dependency graph for fdmultiperiodengine.hpp:

Namespaces

- namespace **QuantLib**

8.288 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference

8.288.1 Detailed Description

Finite-differences shout engine.

`#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/FiniteDifferences/shoutcondition.hpp>`

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for fdshoutengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDShoutEngine

  Finite-differences pricing engine for shout vanilla options.

8.289 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdstepconditionengine.hpp File Reference

8.289.1 Detailed Description

Finite-differences step-condition engine.

`#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/FiniteDifferences/boundarycondition.hpp>`

Include dependency graph for fdstepconditionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDStepConditionEngine

  Finite-differences pricing engine for American-style vanilla options.

8.290 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/fdvanillaengine.hpp File Reference

8.290.1 Detailed Description

Finite-differences vanilla-option engine.

```
#include <ql/Instruments/oneassetoption.hpp>
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
#include <ql/FiniteDifferences/boundarycondition.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for fdvanillaengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FDVanillaEngine

  Finite-differences pricing engine for BSM one asset options.

8.291 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/integralengine.hpp File Reference

8.291.1 Detailed Description

Integral option engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for integralengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class IntegralEngine

8.292 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/jumpdiffusionengine.hpp File Reference

8.292.1 Detailed Description

Jump diffusion (Merton 1976) engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for jumpdiffusionengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class JumpDiffusionEngine

  Jump-diffusion engine for vanilla options.

8.293 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference

8.293.1 Detailed Description

Ju quadratic (1999) approximation engine.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for juquadraticengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class JuQuadraticApproximationEngine

8.294 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

8.294.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
#include <ql/MonteCarlo/mctraits.hpp>
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdigitalengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCDigitalEngine
  Pricing engine for digital options using Monte Carlo simulation.
- class MakeMCDigitalEngine
  Monte Carlo digital engine factory.

8.295 builddir/build/BUILD/QuantLib-0.3.11/ql/Pricing-Engines/Vanilla/mceuropeanengine.hpp File Reference

8.295.1 Detailed Description

Monte Carlo European option engine.

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/Volatilities/blackconstantvol.hpp>
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mceuropeanengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCEuropeanEngine

  European option pricing engine using Monte Carlo simulation.

- class MakeMCEuropeanEngine

  Monte Carlo European engine factory.

8.296 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference

8.296.1 Detailed Description

Monte Carlo Heston-model engine for European options.

`#include <ql/PricingEngines/Vanilla/mchestonengine.hpp>`

Include dependency graph for mceuropeanhestonengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCEuropeanHestonEngine

  Monte Carlo Heston-model engine for European options.

- class MakeMCEuropeanHestonEngine

  Monte Carlo Heston European engine factory.

8.297 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/mchestonengine.hpp File Reference

8.297.1 Detailed Description

Monte Carlo Heston-model engine.

`#include <ql/PricingEngines/mcsimulation.hpp>`

`#include <ql/Instruments/vanillaoption.hpp>`

`#include <ql/Processes/hestonprocess.hpp>`

Include dependency graph for mchestonengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCHestonEngine

  Monte Carlo Heston-model engine.

8.298 builddir/build/BUILD/QuantLib-0.3.11/ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

8.298.1 Detailed Description

Monte Carlo vanilla option engine.

`#include <ql/PricingEngines/mcsimulation.hpp>`

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for mcvanillaengine.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MCVanillaEngine

  Pricing engine for vanilla options using Monte Carlo simulation.

8.299 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/blackscholesproce File Reference

8.299.1 Detailed Description

Black-Scholes processes.

```
#include <ql/stochasticprocess.hpp>
#include <ql/Processes/eulerdiscretization.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for blackscholesprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackScholesProcess

  Black-Scholes stochastic process.

8.300 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/capletlmmprocess File Reference

8.300.1 Detailed Description

stochastic process of a (cap) libor market model

```
#include <ql/stochasticprocess.hpp>
#include <ql/capvolstructures.hpp>
#include <ql/Indexes/xibor.hpp>
#include <map>
```

Include dependency graph for capletlmmprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CapletLiborMarketModelProcess
  caplet libor-market-model process

8.301 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/eulerdiscretizatio File Reference

8.301.1 Detailed Description

Euler discretization for stochastic processes.

`#include <ql/stochasticprocess.hpp>`

Include dependency graph for eulerdiscretization.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class EulerDiscretization

  Euler discretization for stochastic processes.

8.302 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/geometricbrownia File Reference

8.302.1 Detailed Description

Geometric Brownian-motion process.

`#include <ql/stochasticprocess.hpp>`

Include dependency graph for geometricbrownianprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class GeometricBrownianMotionProcess

  Geometric brownian-motion process.

8.303 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/hestonprocess.hpp File Reference

8.303.1 Detailed Description

Heston stochastic process.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for hestonprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HestonProcess

  Square-root stochastic-volatility Heston process.

8.304 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/merton76process.hpp File Reference

8.304.1 Detailed Description

Merton-76 process.

`#include <ql/Processes/blackscholesprocess.hpp>`

`#include <ql/Processes/eulerdiscretization.hpp>`

Include dependency graph for merton76process.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Merton76Process

  Merton-76 jump-diffusion process.

8.305 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/ornsteinuhlenbec File Reference

8.305.1 Detailed Description

Ornstein-Uhlenbeck process.

`#include <ql/stochasticprocess.hpp>`

Include dependency graph for ornsteinuhlenbeckprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OrnsteinUhlenbeckProcess

  Ornstein-Uhlenbeck process class.

8.306 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/squarerootproces File Reference

8.306.1 Detailed Description

square-root process

```
#include <ql/stochasticprocess.hpp>
#include <ql/Processes/eulerdiscretization.hpp>
```

Include dependency graph for squarerootprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SquareRootProcess

  Square-root process class.

8.307 builddir/build/BUILD/QuantLib-0.3.11/ql/Processes/stochasticprocessa File Reference

8.307.1 Detailed Description

Array of correlated 1-D stochastic processes.

`#include <ql/stochasticprocess.hpp>`

`#include <vector>`

Include dependency graph for stochasticprocessarray.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class StochasticProcessArray

  Array of correlated 1-D stochastic processes.

8.308 builddir/build/BUILD/QuantLib-0.3.11/ql/qldefines.hpp File Reference

8.308.1 Detailed Description

Global definitions and compiler switches.

```
#include <boost/config.hpp>
#include <boost/version.hpp>
#include <ql/config.bcc.hpp>
#include <cmath>
#include <boost/limits.hpp>
```

Include dependency graph for qldefines.hpp:

Defines

- #define **BOOST_ENABLE_ASSERT_HANDLER**
- #define **QL_INTEGER** int
- #define **QL_BIG_INTEGER** long
- #define **QL_REAL** double
- #define QL_VERSION "0.3.11"
  version string
- #define QL_HEX_VERSION 0x000311f0
  version hexadecimal number
- #define QL_LIB_VERSION "0_3_11"
  version string for output lib name
- #define QL_DUMMY_RETURN(x)
  Is a dummy return statement required?
- #define QL_IO_INIT
  I/O initialization.
- #define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())
- #define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())
- #define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())
- #define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())
- #define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())
- #define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())
- #define **QL_NULL_INTEGER** ((std::numeric_limits<int>::max)())
- #define **QL_NULL_REAL** ((std::numeric_limits<float>::max)())
- #define QL_TYPENAME typename
- #define QL_FULL_ITERATOR_SUPPORT

8.309 builddir/build/BUILD/QuantLib-0.3.11/ql/quote.hpp File Reference

8.309.1 Detailed Description

purely virtual base class for market observables

`#include <ql/types.hpp>`

`#include <ql/handle.hpp>`

Include dependency graph for quote.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Quote
  purely virtual base class for market observables
- class SimpleQuote
  market element returning a stored value
- class DerivedQuote
  market element whose value depends on another market element
- class CompositeQuote
  market element whose value depends on two other market element

8.310 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

8.310.1 Detailed Description

Box-Muller Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for boxmullergaussianrng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BoxMullerGaussianRng
  Gaussian random number generator.

8.311 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

8.311.1 Detailed Description

Central limit Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for centrallimitgaussianrng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CLGaussianRng

  Gaussian random number generator.

8.312 builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/faurersg.hpp File Reference

8.312.1 Detailed Description

Faure low-discrepancy sequence generator.

```
#include <ql/Math/matrix.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <vector>
```

Include dependency graph for faurersg.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FaureRsg

  Faure low-discrepancy sequence generator.

8.313 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/haltonrsg.hpp File Reference

8.313.1 Detailed Description

Halton low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <vector>
```

Include dependency graph for haltonrsg.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HaltonRsg

  Halton low-discrepancy sequence generator.

8.314 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/inversecumulativerng.hpp File Reference

8.314.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for inversecumulativerng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InverseCumulativeRng
  Inverse cumulative random number generator.

8.315 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/inversecumulativersg.hpp File Reference

8.315.1 Detailed Description

Inverse cumulative random sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativersg.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InverseCumulativeRsg

  Inverse cumulative random sequence generator.

8.316 builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/knuthuniformrng.hpp File Reference

8.316.1 Detailed Description

Knuth uniform random number generator.

`#include <ql/MonteCarlo/sample.hpp>`

`#include <vector>`

Include dependency graph for knuthuniformrng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class KnuthUniformRng

  Uniform random number generator.

8.317 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

8.317.1 Detailed Description

L'Ecuyer uniform random number generator.

`#include <ql/MonteCarlo/sample.hpp>`

`#include <vector>`

Include dependency graph for lecuyeruniformrng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LecuyerUniformRng
  Uniform random number generator.

8.318 builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/mt19937uniformrng.hpp File Reference

8.318.1 Detailed Description

Mersenne Twister uniform random number generator.

`#include <ql/MonteCarlo/sample.hpp>`

`#include <vector>`

Include dependency graph for mt19937uniformrng.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class MersenneTwisterUniformRng
  Uniform random number generator.

8.319 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/randomizedlds.hpp File Reference

8.319.1 Detailed Description

Randomized low-discrepancy sequence.

`#include <ql/RandomNumbers/randomsequencegenerator.hpp>`

`#include <ql/RandomNumbers/mt19937uniformrng.hpp>`

Include dependency graph for randomizedlds.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class RamdomizedLDS

  Randomized (random shift) low-discrepancy sequence.

8.320 builddir/build/BUILD/QuantLib-0.3.11/ql/Random-Numbers/randomsequencegenerator.hpp File Reference

8.320.1 Detailed Description

Random sequence generator based on a pseudo-random number generator.

`#include <ql/Math/array.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

`#include <vector>`

Include dependency graph for randomsequencegenerator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class RandomSequenceGenerator

  Random sequence generator based on a pseudo-random number generator.

8.321 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/rngtraits.hpp File Reference

8.321.1 Detailed Description

random-number generation policies

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumulativerng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumulativersg.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/Math/poissondistribution.hpp>
```

Include dependency graph for rngtraits.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativeNormal > QuantLib::PseudoRandom

  default traits for pseudo-random number generation

- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativePoisson > QuantLib::PoissonPseudoRandom

  traits for Poisson-distributed pseudo-random number generation

- typedef GenericLowDiscrepancy< SobolRsg, InverseCumulativeNormal > QuantLib::LowDiscrepancy

  default traits for low-discrepancy sequence generation

8.322 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/seedgenerator.hpp File Reference

8.322.1 Detailed Description

Random seed generator.

`#include <ql/RandomNumbers/mt19937uniformrng.hpp>`

`#include <ql/Patterns/singleton.hpp>`

Include dependency graph for seedgenerator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SeedGenerator

  Random seed generator.

8.323 builddir/build/BUILD/QuantLib-0.3.11/ql/RandomNumbers/sobolrsg.hpp File Reference

8.323.1 Detailed Description

Sobol low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <vector>
```

Include dependency graph for sobolrsg.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SobolRsg

  Sobol low-discrepancy sequence generator.

8.324 builddir/build/BUILD/QuantLib-0.3.11/ql/schedule.hpp File Reference

8.324.1 Detailed Description

date schedule

`#include <ql/calendar.hpp>`

`#include <ql/Calendars/nullcalendar.hpp>`

`#include <ql/Utilities/null.hpp>`

`#include <vector>`

Include dependency graph for schedule.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Schedule

  Payment schedule.

- class MakeSchedule

  helper class

8.325 builddir/build/BUILD/QuantLib-0.3.11/ql/settings.hpp File Reference

8.325.1 Detailed Description

global repository for run-time library settings

`#include <ql/date.hpp>`

`#include <ql/Patterns/singleton.hpp>`

`#include <ql/Utilities/observablevalue.hpp>`

Include dependency graph for settings.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Settings

  global repository for run-time library settings

Functions

- std::ostream & **QuantLib::operator**<< (std::ostream &out, const Settings::DateProxy &p)

8.326 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/calibrationhelper.hpp File Reference

8.326.1 Detailed Description

Calibration helper class.

`#include <ql/grid.hpp>`

`#include <ql/pricingengine.hpp>`

`#include <ql/PricingEngines/blackmodel.hpp>`

Include dependency graph for calibrationhelper.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CalibrationHelper

  liquid market instrument used during calibration

8.327 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

8.327.1 Detailed Description

CapHelper calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
#include <ql/Instruments/capfloor.hpp>
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for caphelper.hpp:

Namespaces

- namespace **QuantLib**

8.328 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference

8.328.1 Detailed Description

Heston-model calibration helper.

`#include <ql/ShortRateModels/calibrationhelper.hpp>`

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for hestonmodelhelper.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HestonModelHelper
  calibration helper for Heston model

8.329 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

8.329.1 Detailed Description

Swaption calibration helper.

`#include <ql/ShortRateModels/calibrationhelper.hpp>`

`#include <ql/Instruments/swaption.hpp>`

Include dependency graph for swaptionhelper.hpp:

Namespaces

- namespace **QuantLib**

8.330 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/model.hpp File Reference

8.330.1 Detailed Description

Abstract interest rate model class.

`#include <ql/option.hpp>`

`#include <ql/Lattices/lattice.hpp>`

`#include <ql/ShortRateModels/parameter.hpp>`

`#include <ql/ShortRateModels/calibrationhelper.hpp>`

`#include <ql/Optimization/problem.hpp>`

Include dependency graph for model.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AffineModel

  Affine model class.

- class TermStructureConsistentModel

  Term-structure consistent model class.

- class ShortRateModel

  Abstract short-rate model class.

8.331 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/onefactormodel.hpp File Reference

8.331.1 Detailed Description

Abstract one-factor interest rate model class.

```
#include <ql/qldefines.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/ShortRateModels/model.hpp>
#include <ql/Lattices/lattice1d.hpp>
#include <ql/Lattices/trinomialtree.hpp>
```

Include dependency graph for onefactormodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class OneFactorModel

  Single-factor short-rate model abstract class.

- class OneFactorModel::ShortRateDynamics

  Base class describing the short-rate dynamics.

- class OneFactorModel::ShortRateTree

  Recombining trinomial tree discretizing the state variable.

- class OneFactorAffineModel

  Single-factor affine base class.

8.332 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

8.332.1 Detailed Description

Black-Karasinski model.

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/onefactormodel.hpp>`

`#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>`

Include dependency graph for blackkarasinski.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackKarasinski

  Standard Black-Karasinski model class.

- class BlackKarasinski::Dynamics

  Short-rate dynamics in the Black-Karasinski model.

8.333 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

8.333.1 Detailed Description

Cox-Ingersoll-Ross model.

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for coxingersollross.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CoxIngersollRoss

  Cox-Ingersoll-Ross model class.

- class CoxIngersollRoss::Dynamics

  Dynamics of the short-rate under the Cox-Ingersoll-Ross model

8.334 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

8.334.1 Detailed Description

Extended Cox-Ingersoll-Ross model.

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>`

Include dependency graph for extendedcoxingersollross.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ExtendedCoxIngersollRoss
  Extended Cox-Ingersoll-Ross model class.
- class ExtendedCoxIngersollRoss::Dynamics
  Short-rate dynamics in the extended Cox-Ingersoll-Ross model.
- class ExtendedCoxIngersollRoss::FittingParameter
  Analytical term-structure fitting parameter $\varphi(t)$.

8.335 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRate-Models/OneFactorModels/hullwhite.hpp File Reference

8.335.1 Detailed Description

Hull & White (HW) model.

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>`

Include dependency graph for hullwhite.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HullWhite

  Single-factor Hull-White (extended Vasicek) model class.

- class HullWhite::Dynamics

  Short-rate dynamics in the Hull-White model.

- class HullWhite::FittingParameter

  Analytical term-structure fitting parameter $\varphi(t)$.

8.336 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

8.336.1 Detailed Description

Vasicek model class.

`#include <ql/qldefines.hpp>`

`#include <ql/ShortRateModels/onefactormodel.hpp>`

`#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>`

Include dependency graph for vasicek.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Vasicek

  Vasicek model class

- class Vasicek::Dynamics

  Short-rate dynamics in the Vasicek model.

8.337 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/parameter.hpp File Reference

8.337.1 Detailed Description

Model parameter classes.

```
#include <ql/qldefines.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/Optimization/constraint.hpp>
#include <vector>
```

Include dependency graph for parameter.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ParameterImpl

  Base class for model parameter implementation.

- class Parameter

  Base class for model arguments.

- class ConstantParameter

  Standard constant parameter $a(t) = a$.

- class NullParameter

  Parameter which is always zero $a(t) = 0$

- class PiecewiseConstantParameter

  Piecewise-constant parameter.

- class TermStructureFittingParameter

  Deterministic time-dependent parameter used for yield-curve fitting.

8.338 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/twofactormodel.hpp File Reference

8.338.1 Detailed Description

Abstract two-factor interest rate model class.

`#include <ql/stochasticprocess.hpp>`

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/Lattices/lattice2d.hpp>`

Include dependency graph for twofactormodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TwoFactorModel

  Abstract base-class for two-factor models.

- class TwoFactorModel::ShortRateDynamics

  Class describing the dynamics of the two state variables.

- class TwoFactorModel::ShortRateTree

  Recombining two-dimensional tree discretizing the state variable.

8.339 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference

8.339.1 Detailed Description

extended versions of the Heston model

`#include <ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>`

Include dependency graph for batesmodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BatesModel

8.340 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

8.340.1 Detailed Description

Two-factor additive Gaussian Model G2++.

`#include <ql/ShortRateModels/twofactormodel.hpp>`

`#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>`

`#include <ql/Instruments/swaption.hpp>`

Include dependency graph for g2.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class G2

  Two-additive-factor gaussian model class.

- class G2::FittingParameter

  Analytical term-structure fitting parameter $\varphi(t)$.

8.341 builddir/build/BUILD/QuantLib-0.3.11/ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference

8.341.1 Detailed Description

Heston model for the stochastic volatility of an asset.

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/Processes/hestonprocess.hpp>`

Include dependency graph for hestonmodel.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class HestonModel

  Heston model for the stochastic volatility of an asset.

8.342 builddir/build/BUILD/QuantLib-0.3.11/ql/solver1d.hpp File Reference

8.342.1 Detailed Description

Abstract 1-D solver class.

`#include <ql/Utilities/null.hpp>`

`#include <ql/Patterns/curiouslyrecurring.hpp>`

`#include <ql/errors.hpp>`

`#include <iomanip>`

Include dependency graph for solver1d.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Solver1D

  Base class for 1-D solvers.

Defines

- #define **MAX_FUNCTION_EVALUATIONS** 100

8.343 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/bisection.hpp File Reference

8.343.1 Detailed Description

bisection 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for bisection.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Bisection

  Bisection 1-D solver

8.344 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/brent.hpp File Reference

8.344.1 Detailed Description

Brent 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for brent.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Brent
  Brent 1-D solver

8.345 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/falseposition.hpp File Reference

8.345.1 Detailed Description

false-position 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for falseposition.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FalsePosition

  False position 1-D solver.

8.346 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/newton.hpp File Reference

8.346.1 Detailed Description

Newton 1-D solver.

`#include <ql/Solvers1D/newtonsafe.hpp>`

Include dependency graph for newton.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Newton
  Newton 1-D solver

8.347 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/newtonsafe.hpp File Reference

8.347.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for newtonsafe.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class NewtonSafe
  safe Newton 1-D solver

8.348 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/ridder.hpp File Reference

8.348.1 Detailed Description

Ridder 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for ridder.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Ridder
  Ridder 1-D solver

8.349 builddir/build/BUILD/QuantLib-0.3.11/ql/Solvers1D/secant.hpp File Reference

8.349.1 Detailed Description

secant 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for secant.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Secant
  Secant 1-D solver

8.350 builddir/build/BUILD/QuantLib-0.3.11/ql/stochasticprocess.hpp File Reference

8.350.1 Detailed Description

stochastic processes

`#include <ql/date.hpp>`

`#include <ql/Patterns/observable.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for stochasticprocess.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class StochasticProcess
  multi-dimensional stochastic process class.
- class StochasticProcess::discretization
  discretization of a stochastic process over a given time interval
- class StochasticProcess1D
  1-dimensional stochastic process
- class StochasticProcess1D::discretization
  discretization of a 1-D stochastic process

Typedefs

- typedef StochasticProcess QuantLib::GenericStochasticProcess

8.351 builddir/build/BUILD/QuantLib-0.3.11/ql/swaptionvolstructure.hpp File Reference

8.351.1 Detailed Description

Swaption volatility structure.

`#include <ql/termstructure.hpp>`

`#include <ql/Math/extrapolation.hpp>`

Include dependency graph for swaptionvolstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SwaptionVolatilityStructure
  Swaption-volatility structure

8.352 builddir/build/BUILD/QuantLib-0.3.11/ql/termstructure.hpp File Reference

8.352.1 Detailed Description

base class for term structures

`#include <ql/calendar.hpp>`

`#include <ql/daycounter.hpp>`

`#include <ql/settings.hpp>`

`#include <ql/Utilities/null.hpp>`

Include dependency graph for termstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TermStructure

  Basic term-structure functionality.

8.353 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/affinetermstructure.hpp File Reference

8.353.1 Detailed Description

Affine term structure.

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/Optimization/method.hpp>`

`#include <ql/TermStructures/ratehelpers.hpp>`

`#include <ql/Patterns/lazyobject.hpp>`

Include dependency graph for affinetermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class AffineTermStructure

  Term-structure implied by an affine model.

8.354 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bondhelpers.hpp File Reference

8.354.1 Detailed Description

bond rate helpers

`#include <ql/Instruments/fixedcouponbond.hpp>`

`#include <ql/TermStructures/ratehelpers.hpp>`

Include dependency graph for bondhelpers.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FixedCouponBondHelper
  fixed-coupon bond helper

8.355 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/bootstraptraits.hpp File Reference

8.355.1 Detailed Description

bootstrap traits

`#include <ql/TermStructures/discountcurve.hpp>`

`#include <ql/TermStructures/zerocurve.hpp>`

`#include <ql/TermStructures/forwardcurve.hpp>`

Include dependency graph for bootstraptraits.hpp:

Namespaces

- namespace **QuantLib**

Classes

- struct Discount

  Discount-curve traits.

- struct ZeroYield

  Zero-curve traits.

- struct ForwardRate

  Forward-curve traits.

8.356 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/compoundforward.hpp File Reference

8.356.1 Detailed Description

compounded forward term structure

`#include <ql/TermStructures/forwardstructure.hpp>`

`#include <ql/TermStructures/extendeddiscountcurve.hpp>`

`#include <ql/Math/loglinearinterpolation.hpp>`

Include dependency graph for compoundforward.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CompoundForward
  compound-forward structure

8.357 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/discountcurve.hpp File Reference

8.357.1 Detailed Description

interpolated discount factor structure

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/Math/loglinearinterpolation.hpp>`

`#include <vector>`

Include dependency graph for discountcurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InterpolatedDiscountCurve

  Term structure based on interpolation of discount factors.

Typedefs

- typedef InterpolatedDiscountCurve< LogLinear > QuantLib::DiscountCurve

  Term structure based on log-linear interpolation of discount factors.

8.358 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/drifttermstructure.hpp File Reference

8.358.1 Detailed Description

Drift term structure.

`#include <ql/TermStructures/zeroyieldstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for drifttermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class DriftTermStructure

  Drift term structure.

8.359 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/extendeddiscountcurve.hpp File Reference

8.359.1 Detailed Description

discount factor structure with detailed compound-forward calculation

`#include <ql/TermStructures/discountcurve.hpp>`

`#include <map>`

Include dependency graph for extendeddiscountcurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ExtendedDiscountCurve

  Term structure based on loglinear interpolation of discount factors.

8.360 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/flatforward.hpp File Reference

8.360.1 Detailed Description

flat forward rate term structure

`#include <ql/yieldtermstructure.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for flatforward.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class FlatForward

  Flat interest-rate curve.

8.361 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardcurve.hpp File Reference

8.361.1 Detailed Description

interpolated forward-rate structure

`#include <ql/TermStructures/forwardstructure.hpp>`

`#include <ql/Math/backwardflatinterpolation.hpp>`

`#include <vector>`

Include dependency graph for forwardcurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InterpolatedForwardCurve

  Term structure based on interpolation of forward rates.

Typedefs

- typedef InterpolatedForwardCurve< BackwardFlat > QuantLib::ForwardCurve

  Term structure based on flat interpolation of forward rates.

8.362 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

8.362.1 Detailed Description

Forward-spreaded term structure.

`#include <ql/TermStructures/forwardstructure.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for forwardspreadedtermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ForwardSpreadedTermStructure

  Term structure with added spread on the instantaneous forward rate.

8.363 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/forwardstructure.hpp File Reference

8.363.1 Detailed Description

Forward-based yield term structure.

`#include <ql/yieldtermstructure.hpp>`

Include dependency graph for forwardstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ForwardRateStructure

  Forward rate term structure.

8.364 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/impliedtermstructure.hpp File Reference

8.364.1 Detailed Description

Implied term structure.

`#include <ql/yieldtermstructure.hpp>`

Include dependency graph for impliedtermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ImpliedTermStructure

  Implied term structure at a given date in the future.

8.365 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/piecewiseflatforward.hpp File Reference

8.365.1 Detailed Description

piecewise flat forward term structure

`#include <ql/qldefines.hpp>`

`#include <ql/TermStructures/piecewiseyieldcurve.hpp>`

Include dependency graph for piecewiseflatforward.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef PiecewiseYieldCurve< Discount, LogLinear > QuantLib::PiecewiseFlatForward

  Piecewise flat-forward term structure.

8.366 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/piecewiseyieldcurve.hpp File Reference

8.366.1 Detailed Description

piecewise-interpolated term structure

`#include <ql/TermStructures/discountcurve.hpp>`

`#include <ql/TermStructures/ratehelpers.hpp>`

`#include <ql/TermStructures/bootstraptraits.hpp>`

`#include <ql/Math/linearinterpolation.hpp>`

`#include <ql/Solvers1D/brent.hpp>`

Include dependency graph for piecewiseyieldcurve.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class PiecewiseYieldCurve

  Piecewise yield term structure.

8.367 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/quantotermstructure.hpp File Reference

8.367.1 Detailed Description

Quanto term structure.

`#include <ql/TermStructures/zeroyieldstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for quantotermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class QuantoTermStructure

  Quanto term structure.

8.368 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/ratehelpers.hpp File Reference

8.368.1 Detailed Description

rate helpers base class

`#include <ql/quote.hpp>`

`#include <ql/Instruments/simpleswap.hpp>`

Include dependency graph for ratehelpers.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class RateHelper
  Base class for rate helpers.
- class DepositRateHelper
  Deposit rate.
- class FraRateHelper
  Forward rate agreement.
- class FuturesRateHelper
  Interest-rate futures.
- class SwapRateHelper
  Swap rate

8.369 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zerocurve.hpp File Reference

8.369.1 Detailed Description

interpolated zero-rates structure

`#include <ql/TermStructures/zeroyieldstructure.hpp>`

`#include <ql/Math/linearinterpolation.hpp>`

`#include <vector>`

Include dependency graph for zerocurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class InterpolatedZeroCurve

  Term structure based on interpolation of zero yields.

Typedefs

- typedef InterpolatedZeroCurve< Linear > QuantLib::ZeroCurve

  Term structure based on linear interpolation of zero yields.

8.370 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zerospreadedtermstructure.hpp File Reference

8.370.1 Detailed Description

Zero spreaded term structure.

`#include <ql/TermStructures/zeroyieldstructure.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for zerospreadedtermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ZeroSpreadedTermStructure

  Term structure with an added spread on the zero yield rate.

8.371 builddir/build/BUILD/QuantLib-0.3.11/ql/TermStructures/zeroyieldstructure.hpp File Reference

8.371.1 Detailed Description

Zero-yield based term structure.

`#include <ql/yieldtermstructure.hpp>`

Include dependency graph for zeroyieldstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ZeroYieldStructure

  Zero-yield term structure.

8.372 builddir/build/BUILD/QuantLib-0.3.11/ql/timegrid.hpp File Reference

8.372.1 Detailed Description

discrete time grid

```
#include <ql/types.hpp>
#include <ql/errors.hpp>
#include <ql/Math/comparison.hpp>
#include <vector>
#include <numeric>
```

Include dependency graph for timegrid.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class TimeGrid
  time grid class

8.373 builddir/build/BUILD/QuantLib-0.3.11/ql/types.hpp File Reference

8.373.1 Detailed Description

Custom types.

`#include <ql/qldefines.hpp>`

`#include <cstddef>`

Include dependency graph for types.hpp:

Namespaces

- namespace **QuantLib**

Typedefs

- typedef QL_INTEGER QuantLib::Integer
  integer number
- typedef QL_BIG_INTEGER QuantLib::BigInteger
  large integer number
- typedef unsigned QL_INTEGER QuantLib::Natural
  positive integer
- typedef unsigned QL_BIG_INTEGER QuantLib::BigNatural
  large positive integer
- typedef QL_REAL QuantLib::Real
  real number
- typedef Real QuantLib::Decimal
  decimal number
- typedef std::size_t QuantLib::Size
  size of a container
- typedef Real QuantLib::Time
  continuous quantity with 1-year units
- typedef Real QuantLib::DiscountFactor
  discount factor between dates
- typedef Real QuantLib::Rate
  interest rates
- typedef Real QuantLib::Spread

spreads on interest rates

- typedef Real QuantLib::Volatility
  volatility

8.374 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/dataformatters.hpp File Reference

8.374.1 Detailed Description

output manipulators

`#include <ql/Utilities/null.hpp>`

`#include <ostream>`

Include dependency graph for dataformatters.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Functions

- template<typename T> std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const null_checker< T > &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const ordinal_holder &)
- template<typename T> std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const power_of_two_holder< T > &)
- std::ostream & **QuantLib::detail::operator**<< (std::ostream &, const percent_holder &)
- template<typename T> detail::null_checker< T > QuantLib::io::checknull (T)

  check for nulls before output

- detail::ordinal_holder QuantLib::io::ordinal (Size)

  outputs naturals as 1st, 2nd, 3rd...

- template<typename T> detail::power_of_two_holder< T > QuantLib::io::power_of_two (T)

  output integers as powers of two

- detail::percent_holder QuantLib::io::percent (Real)

  output reals as percentages

- detail::percent_holder QuantLib::io::rate (Rate)

  output rates and spreads as percentages

- detail::percent_holder QuantLib::io::volatility (Volatility)

  output volatilities as percentages

8.375 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/dataparsers.hpp File Reference

8.375.1 Detailed Description

Classes used to parse data for input.

`#include <ql/date.hpp>`

`#include <vector>`

Include dependency graph for dataparsers.hpp:

Namespaces

- namespace **QuantLib**

8.376 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/disposable.hpp File Reference

8.376.1 Detailed Description

generic disposable object with move semantics

`#include <ql/qldefines.hpp>`

Include dependency graph for disposable.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Disposable

  generic disposable object with move semantics

8.377 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/null.hpp File Reference

8.377.1 Detailed Description

null values

`#include <ql/types.hpp>`

Include dependency graph for null.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class Null
  template class providing a null value for a given type.

8.378 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/observablevalue.hp File Reference

8.378.1 Detailed Description

observable and assignable proxy to concrete value

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for observablevalue.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ObservableValue

  observable and assignable proxy to concrete value

8.379 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/steppingiterator.hp File Reference

8.379.1 Detailed Description

Iterator advancing in constant steps.

`#include <ql/types.hpp>`

`#include <boost/iterator/iterator_adaptor.hpp>`

Include dependency graph for steppingiterator.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class step_iterator

  Iterator advancing in constant steps.

8.380 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/strings.hpp File Reference

8.380.1 Detailed Description

string utilities

`#include <ql/qldefines.hpp>`

`#include <string>`

Include dependency graph for strings.hpp:

Namespaces

- namespace **QuantLib**

Functions

- std::string **QuantLib::lowercase** (const std::string &)
- std::string **QuantLib::uppercase** (const std::string &)

8.381 builddir/build/BUILD/QuantLib-0.3.11/ql/Utilities/tracing.hpp File Reference

8.381.1 Detailed Description

tracing facilities

```
#include <ql/types.hpp>
#include <ql/errors.hpp>
#include <ql/Patterns/singleton.hpp>
#include <boost/current_function.hpp>
#include <iosfwd>
```

Include dependency graph for tracing.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Defines

- #define **QL_DEFAULT_TRACER**
- #define QL_TRACE_ENABLE
  enable tracing
- #define QL_TRACE_DISABLE
  disable tracing
- #define QL_TRACE_ON(out)
  set tracing stream
- #define QL_TRACE(message)
  output tracing information
- #define QL_TRACE_ENTER_FUNCTION
  output tracing information
- #define QL_TRACE_EXIT_FUNCTION
  output tracing information
- #define QL_TRACE_LOCATION
  output tracing information
- #define QL_TRACE_VARIABLE(variable)
  output tracing information

8.382 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackconstantvo File Reference

8.382.1 Detailed Description

Black constant volatility, no time dependence, no strike dependence.

`#include <ql/voltermstructure.hpp>`

`#include <ql/DayCounters/actual365fixed.hpp>`

Include dependency graph for blackconstantvol.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackConstantVol

  Constant Black volatility, no time-strike dependence.

8.383 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackvariancecu File Reference

8.383.1 Detailed Description

Black volatility curve modelled as variance curve.

```
#include <ql/voltermstructure.hpp>
#include <ql/Math/interpolation.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancecurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackVarianceCurve

  Black volatility curve modelled as variance curve.

8.384 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/blackvariancesu File Reference

8.384.1 Detailed Description

Black volatility surface modelled as variance surface.

`#include <ql/voltermstructure.hpp>`

`#include <ql/Math/matrix.hpp>`

`#include <ql/Math/interpolation2D.hpp>`

`#include <ql/DayCounters/actual365fixed.hpp>`

Include dependency graph for blackvariancesurface.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackVarianceSurface

  Black volatility surface modelled as variance surface.

8.385 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/capflatvolvector. File Reference

8.385.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

```
#include <ql/capvolstructures.hpp>
#include <ql/Math/linearinterpolation.hpp>
#include <ql/DayCounters/thirty360.hpp>
#include <vector>
```

Include dependency graph for capflatvolvector.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CapVolatilityVector

  Cap/floor at-the-money term-volatility vector.

8.386 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/capletconstantvo File Reference

8.386.1 Detailed Description

Constant caplet volatility.

`#include <ql/capvolstructures.hpp>`

`#include <ql/quote.hpp>`

Include dependency graph for capletconstantvol.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class CapletConstantVolatility

  Constant caplet volatility, no time-strike dependence.

8.387 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/capletvariancecu File Reference

8.387.1 Detailed Description

caplet variance curve

```
#include <ql/capvolstructures.hpp>
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for capletvariancecurve.hpp:

Namespaces

- namespace **QuantLib**

8.388 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/impliedvolterms File Reference

8.388.1 Detailed Description

Implied Black Vol Term Structure.

`#include <ql/voltermstructure.hpp>`

Include dependency graph for impliedvoltermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class ImpliedVolTermStructure

  Implied vol term structure at a given date in the future.

8.389 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localconstantvol File Reference

8.389.1 Detailed Description

Local constant volatility, no time dependence, no asset dependence.

`#include <ql/Volatilities/blackconstantvol.hpp>`

Include dependency graph for localconstantvol.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LocalConstantVol

  Constant local volatility, no time-strike dependence.

8.390 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localvolcurve.hp File Reference

8.390.1 Detailed Description

Local volatility curve derived from a Black curve.

`#include <ql/Volatilities/blackvariancecurve.hpp>`

Include dependency graph for localvolcurve.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LocalVolCurve

  Local volatility curve derived from a Black curve.

8.391 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/localvolsurface.h File Reference

8.391.1 Detailed Description

Local volatility surface derived from a Black vol surface.

`#include <ql/voltermstructure.hpp>`

`#include <ql/yieldtermstructure.hpp>`

Include dependency graph for localvolsurface.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class LocalVolSurface

  Local volatility surface derived from a Black vol surface.

8.392 builddir/build/BUILD/QuantLib-0.3.11/ql/Volatilities/swaptionvolmat File Reference

8.392.1 Detailed Description

Swaption at-the-money volatility matrix.

```
#include <ql/swaptionvolstructure.hpp>
#include <ql/Math/matrix.hpp>
#include <ql/Math/bilinearinterpolation.hpp>
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class SwaptionVolatilityMatrix

  At-the-money swaption-volatility matrix.

8.393 builddir/build/BUILD/QuantLib-0.3.11/ql/voltermstructure.hpp File Reference

8.393.1 Detailed Description

Volatility term structures.

```
#include <ql/termstructure.hpp>
#include <ql/quote.hpp>
#include <ql/Math/extrapolation.hpp>
#include <ql/Patterns/visitor.hpp>
#include <vector>
```

Include dependency graph for voltermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class BlackVolTermStructure
  Black-volatility term structure.
- class BlackVolatilityTermStructure
  Black-volatility term structure.
- class BlackVarianceTermStructure
  Black variance term structure.
- class LocalVolTermStructure
  Local-volatility term structure.

8.394 builddir/build/BUILD/QuantLib-0.3.11/ql/yieldtermstructure.hpp File Reference

8.394.1 Detailed Description

Interest-rate term structure.

```
#include <ql/termstructure.hpp>
#include <ql/interestrate.hpp>
#include <ql/handle.hpp>
#include <ql/Math/extrapolation.hpp>
#include <vector>
```

Include dependency graph for yieldtermstructure.hpp:

Namespaces

- namespace **QuantLib**

Classes

- class YieldTermStructure

  Interest-rate term structure.

Chapter 9

QuantLib Example Documentation

9.1 AmericanOption.cpp

This example calculates American options using different methods

```
/* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */

#include <ql/quantlib.hpp>
#include <iostream>

using namespace QuantLib;

#if defined(QL_ENABLE_SESSIONS)
namespace QuantLib {

    Integer sessionId() { return 0; }

}
#endif


int main(int, char* [])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Put);
        Real underlying = 36;
        Real strike = 40;
        Spread dividendYield = 0.00;
        Rate riskFreeRate = 0.06;
        Volatility volatility = 0.20;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Settings::instance().evaluationDate() = todaysDate;

        Date exerciseDate(17, May, 1999);
        DayCounter dayCounter = Actual365Fixed();
        Time maturity = dayCounter.yearFraction(settlementDate,
                                                exerciseDate);

        std::cout << "option type = "  << type << std::endl;
```

```
std::cout << "Time to maturity = "        << maturity
          << std::endl;
std::cout << "Underlying price = "        << underlying
          << std::endl;
std::cout << "Strike = "                  << strike
          << std::endl;
std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
          << std::endl;
std::cout << "Dividend yield = " << io::rate(dividendYield)
          << std::endl;
std::cout << "Volatility = " << io::volatility(volatility)
          << std::endl;
std::cout << std::endl;

std::string method;

Real value, discrepancy, rightValue, relativeDiscrepancy;
rightValue = (type == Option::Put ? 4.48667344 : 2.17372645);

std::cout << std::endl ;

// write column headings
Size widths[] = { 35, 14, 14, 14 };
std::cout << std::setw(widths[0]) << std::left << "Method"
          << std::setw(widths[1]) << std::left << "Value"
          << std::setw(widths[2]) << std::left << "Discrepancy"
          << std::setw(widths[3]) << std::left << "Rel. Discr."
          << std::endl;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

boost::shared_ptr<Exercise> exercise(
                                new EuropeanExercise(exerciseDate));
boost::shared_ptr<Exercise> amExercise(
                                new AmericanExercise(settlementDate,
                                                     exerciseDate));
boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));


Handle<Quote> underlyingH(
    boost::shared_ptr<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
Handle<YieldTermStructure> flatTermStructure(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> flatDividendTS(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> flatVolTS(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility, dayCounter)));

std::vector<Date> dates(4);
dates[0] = settlementDate + 1*Months;
dates[1] = exerciseDate;
dates[2] = exerciseDate + 6*Months;
dates[3] = exerciseDate + 12*Months;
std::vector<Real> strikes(4);
strikes[0] = underlying*0.9;
strikes[1] = underlying;
strikes[2] = underlying*1.1;
strikes[3] = underlying*1.2;
```

```
Matrix vols(4,4);
vols[0][0] = volatility*1.1; vols[0][1] = volatility;
    vols[0][2] = volatility*0.9; vols[0][3] = volatility*0.8;
vols[1][0] = volatility*1.1; vols[1][1] = volatility;
    vols[1][2] = volatility*0.9; vols[1][3] = volatility*0.8;
vols[2][0] = volatility*1.1; vols[2][1] = volatility;
    vols[2][2] = volatility*0.9; vols[2][3] = volatility*0.8;
vols[3][0] = volatility*1.1; vols[3][1] = volatility;
    vols[3][2] = volatility*0.9; vols[3][3] = volatility*0.8;
Handle<BlackVolTermStructure> blackSurface(
    boost::shared_ptr<BlackVolTermStructure>(new
        BlackVarianceSurface(settlementDate, dates,
                             strikes, vols, dayCounter)));

boost::shared_ptr<StrikedTypePayoff> payoff(new
    PlainVanillaPayoff(type, strike));

boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
    BlackScholesProcess(
        underlyingH,
        flatDividendTS,
        flatTermStructure,
        flatVolTS));

// European option
VanillaOption euroOption(stochasticProcess, payoff, exercise,
    boost::shared_ptr<PricingEngine>(new AnalyticEuropeanEngine()));

// method: Black Scholes Engine
method = "equivalent European option";
value = euroOption.NPV();
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << "N/A"
          << std::setw(widths[3]) << std::left << "N/A"
          << std::endl;

// American option
VanillaOption option(stochasticProcess, payoff, amExercise);

// target value
method = "reference value";
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << rightValue
          << std::setw(widths[2]) << std::left << "N/A"
          << std::setw(widths[3]) << std::left << "N/A"
          << std::endl;

Size timeSteps = 801;

// Finite differences
method = "Finite differences";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new FDAmericanEngine(timeSteps,timeSteps-1)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << discrepancy
          << std::scientific
          << std::setw(widths[3]) << std::left << relativeDiscrepancy
          << std::endl;
```

```
// Binomial Method (JR)
method = "Binomial Jarrow-Rudd";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << discrepancy
          << std::scientific
          << std::setw(widths[3]) << std::left << relativeDiscrepancy
          << std::endl;

// Binomial Method (CRR)
method = "Binomial Cox-Ross-Rubinstein";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << discrepancy
          << std::scientific
          << std::setw(widths[3]) << std::left << relativeDiscrepancy
          << std::endl;

// Equal Probability Additive Binomial Tree (EQP)
method = "Additive equiprobabilities";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<AdditiveEQPBinomialTree>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << discrepancy
          << std::scientific
          << std::setw(widths[3]) << std::left << relativeDiscrepancy
          << std::endl;

// Equal Jumps Additive Binomial Tree (Trigeorgis)
method = "Binomial Trigeorgis";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::setw(widths[1]) << std::left << value
          << std::setw(widths[2]) << std::left << discrepancy
          << std::scientific
          << std::setw(widths[3]) << std::left << relativeDiscrepancy
          << std::endl;

// Tian Binomial Tree (third moment matching)
method = "Binomial Tian";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
```

```
        std::cout << std::setw(widths[0]) << std::left << method
                  << std::fixed
                  << std::setw(widths[1]) << std::left << value
                  << std::setw(widths[2]) << std::left << discrepancy
                  << std::scientific
                  << std::setw(widths[3]) << std::left << relativeDiscrepancy
                  << std::endl;

        // Leisen-Reimer Binomial Tree
        method = "Binomial Leisen-Reimer";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << std::setw(widths[0]) << std::left << method
                  << std::fixed
                  << std::setw(widths[1]) << std::left << value
                  << std::setw(widths[2]) << std::left << discrepancy
                  << std::scientific
                  << std::setw(widths[3]) << std::left << relativeDiscrepancy
                  << std::endl;

        // Barone-Adesi and Whaley approximation
        method = "Barone-Adesi and Whaley approx.";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BaroneAdesiWhaleyApproximationEngine));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << std::setw(widths[0]) << std::left << method
                  << std::fixed
                  << std::setw(widths[1]) << std::left << value
                  << std::setw(widths[2]) << std::left << discrepancy
                  << std::scientific
                  << std::setw(widths[3]) << std::left << relativeDiscrepancy
                  << std::endl;

        // Bjerksund and Stensland approximation
        method = "Bjerksund and Stensland approx.";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BjerksundStenslandApproximationEngine));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << std::setw(widths[0]) << std::left << method
                  << std::fixed
                  << std::setw(widths[1]) << std::left << value
                  << std::setw(widths[2]) << std::left << discrepancy
                  << std::scientific
                  << std::setw(widths[3]) << std::left << relativeDiscrepancy
                  << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

9.2 BermudanSwaption.cpp

This is an example of using the QuantLib short rate models.

```
/* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */

#include <ql/quantlib.hpp>
#include <iostream>

using namespace QuantLib;

#if defined(QL_ENABLE_SESSIONS)
namespace QuantLib {

    Integer sessionId() { return 0; }

}
#endif


//Number of swaptions to be calibrated to...

Size numRows = 5;
Size numCols = 5;

Integer swapLenghts[] = {
      1,     2,     3,     4,     5};
Volatility swaptionVols[] = {
  0.1490, 0.1340, 0.1228, 0.1189, 0.1148,
  0.1290, 0.1201, 0.1146, 0.1108, 0.1040,
  0.1149, 0.1112, 0.1070, 0.1010, 0.0957,
  0.1047, 0.1021, 0.0980, 0.0951, 0.1270,
  0.1000, 0.0950, 0.0900, 0.1230, 0.1160};

void calibrateModel(const boost::shared_ptr<ShortRateModel>& model,
                    const std::vector<boost::shared_ptr<CalibrationHelper> >&
                                                                    helpers,
                    Real lambda) {

    Simplex om(lambda, 1e-9);
    om.setEndCriteria(EndCriteria(10000, 1e-7));
    model->calibrate(helpers, om);

    // Output the implied Black volatilities
    for (Size i=0; i<numRows; i++) {
        Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
        Size k = i*numCols + j;
        Real npv = helpers[i]->modelValue();
        Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
                                                    1000, 0.05, 0.50);
        Volatility diff = implied - swaptionVols[k];

        std::cout << i+1 << "x" << swapLenghts[j]
                  << std::setprecision(5) << std::noshowpos
                  << ": model " << std::setw(7) << io::volatility(implied)
                  << ", market " << std::setw(7)
                  << io::volatility(swaptionVols[k])
                  << " (" << std::setw(7) << std::showpos
                  << io::volatility(diff) << std::noshowpos << ")\n";
    }
}

int main(int, char* [])
{
    try {
        QL_IO_INIT
```

```
Date todaysDate(15, February, 2002);
Calendar calendar = TARGET();
Date settlementDate(19, February, 2002);
Settings::instance().evaluationDate() = todaysDate;

// flat yield term structure impling 1x5 swap at 5%
boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
boost::shared_ptr<FlatForward> myTermStructure(
              new FlatForward(settlementDate, Handle<Quote>(flatRate),
                              Actual365Fixed()));
Handle<YieldTermStructure> rhTermStructure;
rhTermStructure.linkTo(myTermStructure);

// Define the ATM/OTM/ITM swaps
Frequency fixedLegFrequency = Annual;
BusinessDayConvention fixedLegConvention = Unadjusted;
BusinessDayConvention floatingLegConvention = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
Frequency floatingLegFrequency = Semiannual;
bool payFixedRate = true;
Integer fixingDays = 2;
Rate dummyFixedRate = 0.03;
boost::shared_ptr<Xibor> indexSixMonths(new
    Euribor(6, Months, rhTermStructure));

Date startDate = calendar.advance(settlementDate,1,Years,
                                  floatingLegConvention);
Date maturity = calendar.advance(startDate,5,Years,
                                 floatingLegConvention);
Schedule fixedSchedule(calendar,startDate,maturity,
                       fixedLegFrequency,fixedLegConvention);
Schedule floatSchedule(calendar,startDate,maturity,
                       floatingLegFrequency,floatingLegConvention);
boost::shared_ptr<SimpleSwap> swap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, dummyFixedRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
Rate fixedATMRate = swap->fairRate();
Rate fixedOTMRate = fixedATMRate * 1.2;
Rate fixedITMRate = fixedATMRate * 0.8;

boost::shared_ptr<SimpleSwap> atmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedATMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
boost::shared_ptr<SimpleSwap> otmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedOTMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
boost::shared_ptr<SimpleSwap> itmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedITMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));

// defining the swaptions to be used in model calibration
std::vector<Period> swaptionMaturities;
swaptionMaturities.push_back(Period(1, Years));
swaptionMaturities.push_back(Period(2, Years));
swaptionMaturities.push_back(Period(3, Years));
swaptionMaturities.push_back(Period(4, Years));
swaptionMaturities.push_back(Period(5, Years));
```

```
std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;

// List of times that have to be included in the timegrid
std::list<Time> times;

Size i;
for (i=0; i<numRows; i++) {
    Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
    Size k = i*numCols + j;
    boost::shared_ptr<Quote> vol(new SimpleQuote(swaptionVols[k]));
    swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
        SwaptionHelper(swaptionMaturities[i],
                       Period(swapLenghts[j], Years),
                       Handle<Quote>(vol),
                       indexSixMonths,
                       indexSixMonths->frequency(),
                       indexSixMonths->dayCounter(),
                       rhTermStructure)));
    swaptions.back()->addTimesTo(times);
}

// Building time-grid
TimeGrid grid(times.begin(), times.end(), 30);


// defining the models
boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));
boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));
boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
boost::shared_ptr<BlackKarasinski> modelBK(
                                new BlackKarasinski(rhTermStructure));


// model calibrations

std::cout << "G2 (analytic formulae) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new G2SwaptionEngine(modelG2, 6.0, 16)));

calibrateModel(modelG2, swaptions, 0.05);
std::cout << "calibrated to:\n"
          << "a     = " << modelG2->params()[0] << ", "
          << "sigma = " << modelG2->params()[1] << "\n"
          << "b     = " << modelG2->params()[2] << ", "
          << "eta   = " << modelG2->params()[3] << "\n"
          << "rho   = " << modelG2->params()[4]
          << std::endl << std::endl;



std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new JamshidianSwaptionEngine(modelHW)));

calibrateModel(modelHW, swaptions, 0.05);
std::cout << "calibrated to:\n"
          << "a = " << modelHW->params()[0] << ", "
          << "sigma = " << modelHW->params()[1]
          << std::endl << std::endl;

std::cout << "Hull-White (numerical) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new TreeSwaptionEngine(modelHW2,grid)));
```

```
calibrateModel(modelHW2, swaptions, 0.05);
std::cout << "calibrated to:\n"
          << "a = " << modelHW2->params()[0] << ", "
          << "sigma = " << modelHW2->params()[1]
          << std::endl << std::endl;

std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new TreeSwaptionEngine(modelBK,grid)));

calibrateModel(modelBK, swaptions, 0.05);
std::cout << "calibrated to:\n"
          << "a = " << modelBK->params()[0] << ", "
          << "sigma = " << modelBK->params()[1]
          << std::endl << std::endl;


// ATM Bermudan swaption pricing

std::cout << "Payer bermudan swaption "
          << "struck at " << io::rate(fixedATMRate)
          << " (ATM)" << std::endl;

std::vector<Date> bermudanDates;
const std::vector<boost::shared_ptr<CashFlow> >& leg =
    swap->fixedLeg();
for (i=0; i<leg.size(); i++) {
    boost::shared_ptr<Coupon> coupon =
        boost::dynamic_pointer_cast<Coupon>(leg[i]);
    bermudanDates.push_back(coupon->accrualStartDate());
}

boost::shared_ptr<Exercise> bermudanExercise(
                                new BermudanExercise(bermudanDates));

Swaption bermudanSwaption(atmSwap, bermudanExercise, rhTermStructure,
                          boost::shared_ptr<PricingEngine>());

// Do the pricing for each model

// G2 price the European swaption here, it should switch to bermudan
bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
    TreeSwaptionEngine(modelG2, 50)));
std::cout << "G2:       " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
   new TreeSwaptionEngine(modelHW, 50)));
std::cout << "HW:       " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
    TreeSwaptionEngine(modelHW2, 50)));
std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
    TreeSwaptionEngine(modelBK, 50)));
std::cout << "BK:       " << bermudanSwaption.NPV() << std::endl;


// OTM Bermudan swaption pricing

std::cout << "Payer bermudan swaption "
          << "struck at " << io::rate(fixedOTMRate)
          << " (OTM)" << std::endl;

Swaption otmBermudanSwaption(otmSwap,bermudanExercise,rhTermStructure,
                             boost::shared_ptr<PricingEngine>());
```

```
        // Do the pricing for each model
        otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelG2, 50)));
        std::cout << "G2:       " << otmBermudanSwaption.NPV() << std::endl;

        otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelHW, 50)));
        std::cout << "HW:       " << otmBermudanSwaption.NPV() << std::endl;

        otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelHW2, 50)));
        std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;

        otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelBK, 50)));
        std::cout << "BK:       " << otmBermudanSwaption.NPV() << std::endl;


        // ITM Bermudan swaption pricing

        std::cout << "Payer bermudan swaption "
                  << "struck at " << io::rate(fixedITMRate)
                  << " (ITM)" << std::endl;

        Swaption itmBermudanSwaption(itmSwap,bermudanExercise,rhTermStructure,
                                     boost::shared_ptr<PricingEngine>());

        // Do the pricing for each model
        itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelG2, 50)));
        std::cout << "G2:       " << itmBermudanSwaption.NPV() << std::endl;

        itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelHW, 50)));
        std::cout << "HW:       " << itmBermudanSwaption.NPV() << std::endl;

        itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelHW2, 50)));
        std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;

        itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelBK, 50)));
        std::cout << "BK:       " << itmBermudanSwaption.NPV() << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

9.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```
/* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */

/*  This example computes profit and loss of a discrete interval hedging
    strategy and compares with the results of Derman & Kamal's (Goldman Sachs
    Equity Derivatives Research) Research Note: "When You Cannot Hedge
    Continuously: The Corrections to Black-Scholes"
    http://www.ederman.com/emanuelderman/GSQSpapers/when_you_cannot_hedge.pdf

    Suppose an option hedger sells an European option and receives the
    Black-Scholes value as the options premium.
    Then he follows a Black-Scholes hedging strategy, rehedging at discrete,
    evenly spaced time intervals as the underlying stock changes. At
    expiration, the hedger delivers the option payoff to the option holder,
    and unwinds the hedge. We are interested in understanding the final
    profit or loss of this strategy.

    If the hedger had followed the exact Black-Scholes replication strategy,
    re-hedging continuously as the underlying stock evolved towards its final
    value at expiration, then, no matter what path the stock took, the final
    P&L would be exactly zero. When the replication strategy deviates from
    the exact Black-Scholes method, the final P&L may deviate from zero. This
    deviation is called the replication error. When the hedger rebalances at
    discrete rather than continuous intervals, the hedge is imperfect and the
    replication is inexact. The more often hedging occurs, the smaller the
    replication error.

    We examine the range of possibilities, computing the replication error.
*/

// the only header you need to use QuantLib
#include <ql/quantlib.hpp>
#include <iostream>

using namespace QuantLib;

#if defined(QL_ENABLE_SESSIONS)
namespace QuantLib {

    Integer sessionId() { return 0; }

}
#endif


/* The ReplicationError class carries out Monte Carlo simulations to evaluate
   the outcome (the replication error) of the discrete hedging strategy over
   different, randomly generated scenarios of future stock price evolution.
*/
class ReplicationError
{
public:
    ReplicationError(Option::Type type,
                     Time maturity,
                     Real strike,
                     Real s0,
                     Volatility sigma,
                     Rate r)
    : maturity_(maturity), payoff_(type, strike), s0_(s0),
      sigma_(sigma), r_(r) {

        // value of the option
        DiscountFactor rDiscount = std::exp(-r_*maturity_);
```

```
        DiscountFactor qDiscount = 1.0;
        Real forward = s0_*qDiscount/rDiscount;
        Real variance = sigma_*sigma_*maturity_;
        boost::shared_ptr<StrikedTypePayoff> payoff(
                                            new PlainVanillaPayoff(payoff_));
        BlackFormula black(forward,rDiscount,variance,payoff);
        std::cout << "Option value: " << black.value() << std::endl;

        // store option's vega, since Derman and Kamal's formula needs it
        vega_ = black.vega(maturity_);

        std::cout << std::endl;
        std::cout <<
            "        |        | P&L  \t|  P&L    | Derman&Kamal | P&L"
            "      \t| P&L" << std::endl;

        std::cout <<
            "samples | trades | Mean \t| Std Dev | Formula      |"
            " skewness \t| kurt." << std::endl;

        std::cout << "---------------------------------"
            "-----------------------------------------------" << std::endl;
    }

    // the actual replication error computation
    void compute(Size nTimeSteps, Size nSamples);
private:
    Time maturity_;
    PlainVanillaPayoff payoff_;
    Real s0_;
    Volatility sigma_;
    Rate r_;
    Real vega_;
};

// The key for the MonteCarlo simulation is to have a PathPricer that
// implements a value(const Path& path) method.
// This method prices the portfolio for each Path of the random variable
class ReplicationPathPricer : public PathPricer<Path> {
  public:
    // real constructor
    ReplicationPathPricer(Option::Type type,
                          Real underlying,
                          Real strike,
                          Rate r,
                          Time maturity,
                          Volatility sigma)
    : type_(type), underlying_(underlying),
      strike_(strike), r_(r), maturity_(maturity), sigma_(sigma) {
        QL_REQUIRE(strike_ > 0.0, "strike must be positive");
        QL_REQUIRE(underlying_ > 0.0, "underlying must be positive");
        QL_REQUIRE(r_ >= 0.0,
                   "risk free rate (r) must be positive or zero");
        QL_REQUIRE(maturity_ > 0.0, "maturity must be positive");
        QL_REQUIRE(sigma_ >= 0.0,
                   "volatility (sigma) must be positive or zero");

    }
    // The value() method encapsulates the pricing code
    Real operator()(const Path& path) const;

  private:
    Option::Type type_;
    Real underlying_, strike_;
    Rate r_;
    Time maturity_;
    Volatility sigma_;
```

```
};


// Compute Replication Error as in the Derman and Kamal's research note
int main(int, char* [])
{
    try {
        QL_IO_INIT

        Time maturity = 1.0/12.0;   // 1 month
        Real strike = 100;
        Real underlying = 100;
        Volatility volatility = 0.20; // 20%
        Rate riskFreeRate = 0.05; // 5%
        ReplicationError rp(Option::Call, maturity, strike, underlying,
                volatility, riskFreeRate);

        Size scenarios = 50000;
        Size hedgesNum;

        hedgesNum = 21;
        rp.compute(hedgesNum, scenarios);

        hedgesNum = 84;
        rp.compute(hedgesNum, scenarios);

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}


/* The actual computation of the Profit&Loss for each single path.

   In each scenario N rehedging trades spaced evenly in time over
   the life of the option are carried out, using the Black-Scholes
   hedge ratio.
*/
Real ReplicationPathPricer::operator()(const Path& path) const {

    Size n = path.length()-1;
    QL_REQUIRE(n>0, "the path cannot be empty");

    // discrete hedging interval
    Time dt = maturity_/n;

    // For simplicity, we assume the stock pays no dividends.
    Rate stockDividendYield = 0.0;

    // let's start
    Time t = 0;

    // stock value at t=0
    Real stock = underlying_;

    // money account at t=0
    Real money_account = 0.0;

    /************************/
    /*** the initial deal ***/
    /************************/
    // option fair price (Black-Scholes) at t=0
```

```
    DiscountFactor rDiscount = std::exp(-r_*maturity_);
    DiscountFactor qDiscount = std::exp(-stockDividendYield*maturity_);
    Real forward = stock*qDiscount/rDiscount;
    Real variance = sigma_*sigma_*maturity_;
    boost::shared_ptr<StrikedTypePayoff> payoff(
                                    new PlainVanillaPayoff(type_,strike_));
    BlackFormula black(forward,rDiscount,variance,payoff);
    // sell the option, cash in its premium
    money_account += black.value();
    // compute delta
    Real delta = black.delta(stock);
    // delta-hedge the option buying stock
    Real stockAmount = delta;
    money_account -= stockAmount*stock;

    /*********************************/
    /*** hedging during option life ***/
    /*********************************/
    for (Size step = 0; step < n-1; step++){

        // time flows
        t += dt;

        // accruing on the money account
        money_account *= std::exp( r_*dt );

        // stock growth:
        stock = path[step+1];

        // recalculate option value at the current stock value,
        // and the current time to maturity
        rDiscount = std::exp(-r_*(maturity_-t));
        qDiscount = std::exp(-stockDividendYield*(maturity_-t));
        forward = stock*qDiscount/rDiscount;
        variance = sigma_*sigma_*(maturity_-t);
        BlackFormula black(forward,rDiscount,variance,payoff);

        // recalculate delta
        delta = black.delta(stock);

        // re-hedging
        money_account -= (delta - stockAmount)*stock;
        stockAmount = delta;
    }

    /*************************/
    /*** option expiration ***/
    /*************************/
    // last accrual on my money account
    money_account *= std::exp( r_*dt );
    // last stock growth
    stock = path[n];

    // the hedger delivers the option payoff to the option holder
    Real optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
    money_account -= optionPayoff;

    // and unwinds the hedge selling his stock position
    money_account += stockAmount*stock;

    // final Profit&Loss
    return money_account;
}


// The computation over nSamples paths of the P&L distribution
void ReplicationError::compute(Size nTimeSteps, Size nSamples)
```

```
{
    QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");

    // hedging interval
    // Time tau = maturity_ / nTimeSteps;

    /* Black-Scholes framework: the underlying stock price evolves
       lognormally with a fixed known volatility that stays constant
       throughout time.
    */
    Date today = Date::todaysDate();
    DayCounter dayCount = Actual365Fixed();
    Handle<Quote> stateVariable(
                          boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
    Handle<YieldTermStructure> riskFreeRate(
                          boost::shared_ptr<YieldTermStructure>(
                                      new FlatForward(today, r_, dayCount)));
    Handle<YieldTermStructure> dividendYield(
                          boost::shared_ptr<YieldTermStructure>(
                                      new FlatForward(today, 0.0, dayCount)));
    Handle<BlackVolTermStructure> volatility(
                          boost::shared_ptr<BlackVolTermStructure>(
                               new BlackConstantVol(today, sigma_, dayCount)));
    boost::shared_ptr<StochasticProcess1D> diffusion(
                         new BlackScholesProcess(stateVariable, dividendYield,
                                                 riskFreeRate, volatility));

    // Black Scholes equation rules the path generator:
    // at each step the log of the stock
    // will have drift and sigma^2 variance
    PseudoRandom::rsg_type rsg =
        PseudoRandom::make_sequence_generator(nTimeSteps, 0);

    bool brownianBridge = false;

    typedef SingleVariate<PseudoRandom>::path_generator_type generator_type;
    boost::shared_ptr<generator_type> myPathGenerator(new
        generator_type(diffusion, maturity_, nTimeSteps,
                       rsg, brownianBridge));

    // The replication strategy's Profit&Loss is computed for each path
    // of the stock. The path pricer knows how to price a path using its
    // value() method
    boost::shared_ptr<PathPricer<Path> > myPathPricer(new
        ReplicationPathPricer(payoff_.optionType(), s0_,
            payoff_.strike(), r_, maturity_, sigma_));

    // a statistics accumulator for the path-dependant Profit&Loss values
    Statistics statisticsAccumulator;

    // The OneFactorMontecarloModel generates paths using myPathGenerator
    // each path is priced using myPathPricer
    // prices will be accumulated into statisticsAccumulator
    OneFactorMonteCarloOption MCSimulation(myPathGenerator,
                                           myPathPricer,
                                           statisticsAccumulator,
                                           false);

    // the model simulates nSamples paths
    MCSimulation.addSamples(nSamples);

    // the sampleAccumulator method of OneFactorMonteCarloOption_old
    // gives access to all the methods of statisticsAccumulator
    Real PLMean  = MCSimulation.sampleAccumulator().mean();
    Real PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
    Real PLSkew  = MCSimulation.sampleAccumulator().skewness();
    Real PLKurt  = MCSimulation.sampleAccumulator().kurtosis();
```

```
    // Derman and Kamal's formula
    Real theorStD = std::sqrt(M_PI/4/nTimeSteps)*vega_*sigma_;


    std::cout << std::fixed
              << nSamples << "\t| "
              << nTimeSteps << "\t | "
              << std::setprecision(3) << PLMean << " \t| "
              << std::setprecision(2) << PLStDev << " \t  | "
              << std::setprecision(2) << theorStD << " \t | "
              << std::setprecision(2) << PLSkew << " \t| "
              << std::setprecision(2) << PLKurt << std::endl;
}
```

9.4 EuropeanOption.cpp

This example calculates European options using different methods while testing call-put parity.

```
/* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */

#include <ql/quantlib.hpp>
#include <iostream>

using namespace QuantLib;

#if defined(QL_ENABLE_SESSIONS)
namespace QuantLib {

    Integer sessionId() { return 0; }

}
#endif


// This will be included in the library after a bit of redesign
class WeightedPayoff {
    public:
        WeightedPayoff(Option::Type type,
               Time maturity,
               Real strike,
               Real s0,
               Volatility sigma,
               Rate r,
               Rate q)
        : type_(type), maturity_(maturity),
        strike_(strike),
        s0_(s0),
        sigma_(sigma),r_(r), q_(q){}

        Real operator()(Real x) const {
           Real nuT = (r_-q_-0.5*sigma_*sigma_)*maturity_;
           return std::exp(-r_*maturity_)
               *PlainVanillaPayoff(type_, strike_)(s0_*std::exp(x))
               *std::exp(-(x - nuT)*(x -nuT)/(2*sigma_*sigma_*maturity_))
               /std::sqrt(2.0*M_PI*sigma_*sigma_*maturity_);
        }
private:
    Option::Type type_;
    Time maturity_;
    Real strike_;
    Real s0_;
    Volatility sigma_;
    Rate r_,q_;
};


int main(int, char* [])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Call);
        Real underlying = 7;
        Real strike = 8;
        Spread dividendYield = 0.05;
        Rate riskFreeRate = 0.05;
```

```
Date todaysDate(15, May, 1998);
Date settlementDate(17, May, 1998);
Settings::instance().evaluationDate() = todaysDate;

Date exerciseDate(17, May, 1999);
DayCounter dayCounter = Actual365Fixed();
Time maturity = dayCounter.yearFraction(settlementDate,
                                        exerciseDate);

Volatility volatility = 0.10;
std::cout << "option type = "  << type << std::endl;
std::cout << "Time to maturity = "        << maturity
          << std::endl;
std::cout << "Underlying price = "        << underlying
          << std::endl;
std::cout << "Strike = "                  << strike
          << std::endl;
std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
          << std::endl;
std::cout << "Dividend yield = " << io::rate(dividendYield)
          << std::endl;
std::cout << "Volatility = " << io::volatility(volatility)
          << std::endl;
std::cout << std::endl;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

boost::shared_ptr<Exercise> exercise(
                                  new EuropeanExercise(exerciseDate));
boost::shared_ptr<Exercise> amExercise(
                                  new AmericanExercise(settlementDate,
                                                       exerciseDate));
boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));


Handle<Quote> underlyingH(
    boost::shared_ptr<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
Handle<YieldTermStructure> flatTermStructure(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> flatDividendTS(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> flatVolTS(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility, dayCounter)));

std::vector<Date> dates(4);
dates[0] = settlementDate + 1*Months;
dates[1] = exerciseDate;
dates[2] = exerciseDate + 6*Months;
dates[3] = exerciseDate + 12*Months;
std::vector<Real> strikes(4);
strikes[0] = underlying*0.9;
strikes[1] = underlying;
strikes[2] = underlying*1.1;
strikes[3] = underlying*1.2;

Matrix vols(4,4);
vols[0][0] = volatility*1.1;
             vols[0][1] = volatility;
                          vols[0][2] = volatility*0.9;
```

```
                                        vols[0][3] = volatility*0.8;
        vols[1][0] = volatility*1.1;
                    vols[1][1] = volatility;
                                vols[1][2] = volatility*0.9;
                                        vols[1][3] = volatility*0.8;
        vols[2][0] = volatility*1.1;
                    vols[2][1] = volatility;
                                vols[2][2] = volatility*0.9;
                                        vols[2][3] = volatility*0.8;
        vols[3][0] = volatility*1.1;
                    vols[3][1] = volatility;
                                vols[3][2] = volatility*0.9;
                                        vols[3][3] = volatility*0.8;

        Handle<BlackVolTermStructure> blackSurface(
            boost::shared_ptr<BlackVolTermStructure>(
                new BlackVarianceSurface(settlementDate, dates,
                                         strikes, vols, dayCounter)));


        boost::shared_ptr<StrikedTypePayoff> payoff(new
            PlainVanillaPayoff(type, strike));

        boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
            BlackScholesProcess(underlyingH, flatDividendTS,
                                flatTermStructure,
                                //  blackSurface
                                flatVolTS));

        EuropeanOption option(stochasticProcess, payoff, exercise);


        std::string method;
        Real value, discrepancy, rightValue, relativeDiscrepancy;

        std::cout << std::endl << std::endl;

        // write column headings
        std::cout << "Method\t\tValue\t\tEstimatedError\tDiscrepancy"
                     "\tRel. Discr." << std::endl;

        // method: Black-Scholes Engine
        method = "Black-Scholes";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new AnalyticEuropeanEngine()));
        rightValue = value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t\t" << relativeDiscrepancy << std::endl;


        // method: Integral
        method = "Integral";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new IntegralEngine()));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

/*
        // method: Integral
        method = "Binary Cash";
```

```
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new IntegralCashOrNothingEngine(1.0)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

        // method: Integral
        method = "Binary Asset";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new IntegralAssetOrNothingEngine()));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

*/
        Size timeSteps = 801;

        // Binomial Method (JR)
        method = "Binomial (JR)";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;


        // Binomial Method (CRR)
        method = "Binomial (CRR)";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

        // Equal Probability Additive Binomial Tree (EQP)
        method = "Additive (EQP)";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BinomialVanillaEngine<AdditiveEQPBinomialTree>(timeSteps)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

        // Equal Jumps Additive Binomial Tree (Trigeorgis)
        method = "Bin. Trigeorgis";
        option.setPricingEngine(boost::shared_ptr<PricingEngine>(
            new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;
```

```
// Tian Binomial Tree (third moment matching)
method = "Binomial Tian";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << value << "\t" << "N/A\t\t"
          << discrepancy << "\t" << relativeDiscrepancy << std::endl;

// Leisen-Reimer Binomial Tree
method = "Binomial LR";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << value << "\t" << "N/A\t\t"
          << discrepancy << "\t" << relativeDiscrepancy << std::endl;

// Finite Differences

method = "Finite Diff.";
timeSteps = 100;
Size gridPoints = 100;
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new FDEuropeanEngine(timeSteps, gridPoints)));
value = option.NPV();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << value << "\t" << "N/A\t\t"
          << discrepancy << "\t" << relativeDiscrepancy << std::endl;

// Monte Carlo Method
timeSteps = 1;

method = "MC (crude)";
Size mcSeed = 42;

boost::shared_ptr<PricingEngine> mcengine1;
mcengine1 =
    MakeMCEuropeanEngine<PseudoRandom>().withSteps(timeSteps)
                                        .withTolerance(0.02)
                                        .withSeed(mcSeed);
option.setPricingEngine(mcengine1);

value = option.NPV();
Real errorEstimate = option.errorEstimate();
discrepancy = std::fabs(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << value << "\t" << errorEstimate << "\t"
          << discrepancy << "\t" << relativeDiscrepancy << std::endl;

method = "MC (Sobol)";
timeSteps = 1;
Size nSamples = 32768;  // 2^15

boost::shared_ptr<PricingEngine> mcengine2;
mcengine2 =
    MakeMCEuropeanEngine<LowDiscrepancy>().withSteps(timeSteps)
                                          .withSamples(nSamples);
option.setPricingEngine(mcengine2);
```

```
        value = option.NPV();
        discrepancy = std::fabs(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
                  << value << "\t" << "N/A\t\t"
                  << discrepancy << "\t" << relativeDiscrepancy << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

9.5 history_iterators.cpp

This code exemplifies how to use History iterators to perform Gaussian statistic analyses on historical data.

```
// initialize a History
History h(...);

// print out the mean value and its standard deviation.

GaussianStatistics s;
s.addSequence(h.vdbegin(),h.vdend());
cout << "Historical mean: " << s.mean() << endl;
cout << "Std. deviation:  " << s.standardDeviation() << endl;

// Another possibility: print out the maximum value.

History::const_valid_iterator max = h.vbegin(), i=max, end = h.vend();
for (i++; i!=end; i++)
    if (i->value() > max->value())
        max = i;
cout << "Maximum value: " << max->value()
     << " assumed " << DateFormatter::toString(max->date()) << endl;

// or the minimum, this time the STL way:

bool lessthan(const History::Entry& i, const History::Entry& j) {
    return i.value() < j.value();
}

History::const_valid_iterator min =
    std::min_element(h.vbegin(),h.vend(),lessthan);
cout << "Minimum value: " << min->value()
     << " assumed " << DateFormatter::toString(min->date()) << endl;
```

9.6 swapvaluation.cpp

This is an example of using the QuantLib Term Structure for pricing a simple swap.

```
/* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */

/*  This example shows how to set up a Term Structure and then price a simple
    swap.
*/

// the only header you need to use QuantLib
#include <ql/quantlib.hpp>
#include <iostream>
#include <iomanip>

using namespace QuantLib;

#if defined(QL_ENABLE_SESSIONS)
namespace QuantLib {

    Integer sessionId() { return 0; }

}
#endif


int main(int, char* [])
{
    try {
        QL_IO_INIT

        /*********************
         ***  MARKET DATA  ***
         *********************/

        Calendar calendar = TARGET();
        // uncommenting the following line generates an error
        // calendar = Tokyo();
        Date settlementDate(22, September, 2004);
        // must be a business day
        settlementDate = calendar.adjust(settlementDate);

        Integer fixingDays = 2;
        Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
        // nothing to do with Date::todaysDate
        Settings::instance().evaluationDate() = todaysDate;


        todaysDate = Settings::instance().evaluationDate();
        std::cout << "Today: " << todaysDate.weekday()
                  << ", " << todaysDate << std::endl;

        std::cout << "Settlement date: " << settlementDate.weekday()
                  << ", " << settlementDate << std::endl;

        // deposits
        Rate d1wQuote=0.0382;
        Rate d1mQuote=0.0372;
        Rate d3mQuote=0.0363;
        Rate d6mQuote=0.0353;
        Rate d9mQuote=0.0348;
        Rate d1yQuote=0.0345;
        // FRAs
        Rate fra3x6Quote=0.037125;
        Rate fra6x9Quote=0.037125;
        Rate fra6x12Quote=0.037125;
```

```
// futures
Real fut1Quote=96.2875;
Real fut2Quote=96.7875;
Real fut3Quote=96.9875;
Real fut4Quote=96.6875;
Real fut5Quote=96.4875;
Real fut6Quote=96.3875;
Real fut7Quote=96.2875;
Real fut8Quote=96.0875;
// swaps
Rate s2yQuote=0.037125;
Rate s3yQuote=0.0398;
Rate s5yQuote=0.0443;
Rate s10yQuote=0.05165;
Rate s15yQuote=0.055175;


/*******************
 ***    QUOTES    ***
 *******************/

// SimpleQuote stores a value which can be manually changed;
// other Quote subclasses could read the value from a database
// or some kind of data feed.

// deposits
boost::shared_ptr<Quote> d1wRate(new SimpleQuote(d1wQuote));
boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
// FRAs
boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
// futures
boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
// swaps
boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));
boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));
boost::shared_ptr<Quote> s15yRate(new SimpleQuote(s15yQuote));


/*********************
 ***  RATE HELPERS ***
 *********************/

// RateHelpers are built from the above quotes together with
// other instrument dependant infos.  Quotes are passed in
// relinkable handles which could be relinked to some other
// data source later.

// deposits
DayCounter depositDayCounter = Actual360();

boost::shared_ptr<RateHelper> d1w(new DepositRateHelper(
    Handle<Quote>(d1wRate),
```

```
    1, Weeks, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
    Handle<Quote>(d1mRate),
    1, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
    Handle<Quote>(d3mRate),
    3, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
    Handle<Quote>(d6mRate),
    6, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
    Handle<Quote>(d9mRate),
    9, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
    Handle<Quote>(d1yRate),
    1, Years, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));


// setup FRAs
boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
    Handle<Quote>(fra3x6Rate),
    3, 6, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));
boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
    Handle<Quote>(fra6x9Rate),
    6, 9, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));
boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
    Handle<Quote>(fra6x12Rate),
    6, 12, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));


// setup futures
Integer futMonths = 3;
Date imm = Date::nextIMMdate(settlementDate);
boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(
```

```
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));
    imm = Date::nextIMMdate(imm+1);
    boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));
    imm = Date::nextIMMdate(imm+1);
    boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));
    imm = Date::nextIMMdate(imm+1);
    boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));


    // setup swaps
    Frequency swFixedLegFrequency = Annual;
    BusinessDayConvention swFixedLegConvention = Unadjusted;
    DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
    Frequency swFloatingLegFrequency = Semiannual;

    boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
        Handle<Quote>(s2yRate),
        2, Years, fixingDays,
        calendar, swFixedLegFrequency,
        swFixedLegConvention, swFixedLegDayCounter,
        swFloatingLegFrequency, ModifiedFollowing));
    boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
        Handle<Quote>(s3yRate),
        3, Years, fixingDays,
        calendar, swFixedLegFrequency,
        swFixedLegConvention, swFixedLegDayCounter,
        swFloatingLegFrequency, ModifiedFollowing));
    boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(
        Handle<Quote>(s5yRate),
        5, Years, fixingDays,
        calendar, swFixedLegFrequency,
        swFixedLegConvention, swFixedLegDayCounter,
        swFloatingLegFrequency, ModifiedFollowing));
    boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
        Handle<Quote>(s10yRate),
        10, Years, fixingDays,
        calendar, swFixedLegFrequency,
        swFixedLegConvention, swFixedLegDayCounter,
        swFloatingLegFrequency, ModifiedFollowing));
    boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
        Handle<Quote>(s15yRate),
        15, Years, fixingDays,
        calendar, swFixedLegFrequency,
        swFixedLegConvention, swFixedLegDayCounter,
        swFloatingLegFrequency, ModifiedFollowing));


    /*********************
     **  CURVE BUILDING **
     *********************/

    // Any DayCounter would be fine.
```

```
// ActualActual::ISDA ensures that 30 years is 30.0
DayCounter termStructureDayCounter =
    ActualActual(ActualActual::ISDA);


double tolerance = 1.0e-15;

// A depo-swap curve
std::vector<boost::shared_ptr<RateHelper> > depoSwapInstruments;
depoSwapInstruments.push_back(d1w);
depoSwapInstruments.push_back(d1m);
depoSwapInstruments.push_back(d3m);
depoSwapInstruments.push_back(d6m);
depoSwapInstruments.push_back(d9m);
depoSwapInstruments.push_back(d1y);
depoSwapInstruments.push_back(s2y);
depoSwapInstruments.push_back(s3y);
depoSwapInstruments.push_back(s5y);
depoSwapInstruments.push_back(s10y);
depoSwapInstruments.push_back(s15y);
boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoSwapInstruments,
                         termStructureDayCounter, tolerance));


// A depo-futures-swap curve
std::vector<boost::shared_ptr<RateHelper> > depoFutSwapInstruments;
depoFutSwapInstruments.push_back(d1w);
depoFutSwapInstruments.push_back(d1m);
depoFutSwapInstruments.push_back(fut1);
depoFutSwapInstruments.push_back(fut2);
depoFutSwapInstruments.push_back(fut3);
depoFutSwapInstruments.push_back(fut4);
depoFutSwapInstruments.push_back(fut5);
depoFutSwapInstruments.push_back(fut6);
depoFutSwapInstruments.push_back(fut7);
depoFutSwapInstruments.push_back(fut8);
depoFutSwapInstruments.push_back(s3y);
depoFutSwapInstruments.push_back(s5y);
depoFutSwapInstruments.push_back(s10y);
depoFutSwapInstruments.push_back(s15y);
boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoFutSwapInstruments,
                         termStructureDayCounter, tolerance));


// A depo-FRA-swap curve
std::vector<boost::shared_ptr<RateHelper> > depoFRASwapInstruments;
depoFRASwapInstruments.push_back(d1w);
depoFRASwapInstruments.push_back(d1m);
depoFRASwapInstruments.push_back(d3m);
depoFRASwapInstruments.push_back(fra3x6);
depoFRASwapInstruments.push_back(fra6x9);
depoFRASwapInstruments.push_back(fra6x12);
depoFRASwapInstruments.push_back(s2y);
depoFRASwapInstruments.push_back(s3y);
depoFRASwapInstruments.push_back(s5y);
depoFRASwapInstruments.push_back(s10y);
depoFRASwapInstruments.push_back(s15y);
boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoFRASwapInstruments,
                         termStructureDayCounter, tolerance));


// Term structures that will be used for pricing:
// the one used for discounting cash flows
Handle<YieldTermStructure> discountingTermStructure;
```

```
// the one used for forward rate forecasting
Handle<YieldTermStructure> forecastingTermStructure;


/*********************
* SWAPS TO BE PRICED *
**********************/

// constant nominal 1,000,000 Euro
Real nominal = 1000000.0;
// fixed leg
Frequency fixedLegFrequency = Annual;
BusinessDayConvention fixedLegConvention = Unadjusted;
BusinessDayConvention floatingLegConvention = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
Rate fixedRate = 0.04;

// floating leg
Frequency floatingLegFrequency = Semiannual;
boost::shared_ptr<Xibor> euriborIndex(new Euribor(6, Months,
    forecastingTermStructure)); // using the forecasting curve
Spread spread = 0.0;

Integer lenghtInYears = 5;
bool payFixedRate = true;

Date maturity = calendar.advance(settlementDate, lenghtInYears, Years,
                                 floatingLegConvention);
Schedule fixedSchedule(calendar, settlementDate, maturity,
                       fixedLegFrequency, fixedLegConvention);
Schedule floatSchedule(calendar, settlementDate, maturity,
                       floatingLegFrequency, floatingLegConvention);
SimpleSwap spot5YearSwap(
    payFixedRate, nominal,
    fixedSchedule, fixedRate, fixedLegDayCounter,
    floatSchedule, euriborIndex, fixingDays, spread,
    discountingTermStructure);

Date fwdStart = calendar.advance(settlementDate, 1, Years);
Date fwdMaturity = calendar.advance(fwdStart, lenghtInYears, Years,
                                    floatingLegConvention);
Schedule fwdFixedSchedule(calendar, fwdStart, fwdMaturity,
                          fixedLegFrequency, fixedLegConvention);
Schedule fwdFloatSchedule(calendar, fwdStart, fwdMaturity,
                          floatingLegFrequency, floatingLegConvention);
SimpleSwap oneYearForward5YearSwap(
    payFixedRate, nominal,
    fwdFixedSchedule, fixedRate, fixedLegDayCounter,
    fwdFloatSchedule, euriborIndex, fixingDays, spread,
    discountingTermStructure);


/***************
* SWAP PRICING *
****************/

// utilities for reporting
std::vector<std::string> headers(4);
headers[0] = "term structure";
headers[1] = "net present value";
headers[2] = "fair spread";
headers[3] = "fair fixed rate";
std::string separator = " | ";
Size width = headers[0].size() + separator.size()
           + headers[1].size() + separator.size()
           + headers[2].size() + separator.size()
           + headers[3].size() + separator.size() - 1;
```

```
std::string rule(width, '-'), dblrule(width, '=');
std::string tab(8, ' ');

// calculations

std::cout << dblrule << std::endl;
std::cout <<  "5-year market swap-rate = "
          << std::setprecision(2) << io::rate(s5yRate->value())
          << std::endl;
std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
          << io::rate(fixedRate) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

Real NPV;
Rate fairRate;
Spread fairSpread;

// Of course, you're not forced to really use different curves
forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;


// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
           "5-years swap mispriced by "
           << io::rate(std::fabs(fairRate-s5yQuote)));


forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;

QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
           "5-years swap mispriced!");
```

```
forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;

QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
           "5-years swap mispriced!");


std::cout << rule << std::endl;

// now let's price the 1Y forward 5Y swap

std::cout << tab << "5-years, 1-year forward swap paying "
          << io::rate(fixedRate) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;
std::cout << rule << std::endl;


forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;


forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
```

```
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;


forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;


// now let's say that the 5-years swap rate goes up to 4.60%.
// A smarter market element--say, connected to a data source-- would
// notice the change itself. Since we're using SimpleQuotes,
// we'll have to change the value manually--which forces us to
// downcast the handle and use the SimpleQuote
// interface. In any case, the point here is that a change in the
// value contained in the Quote triggers a new bootstrapping
// of the curve and a repricing of the swap.

boost::shared_ptr<SimpleQuote> fiveYearsRate =
    boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
fiveYearsRate->setValue(0.0460);

std::cout << dblrule << std::endl;
std::cout <<  "5-year market swap-rate = "
          << io::rate(s5yRate->value()) << std::endl;
std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
          << io::rate(fixedRate) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

// now get the updated results
forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;
```

```
QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
           "5-years swap mispriced!");


forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;

QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
           "5-years swap mispriced!");


forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
          << std::fixed << std::setprecision(2) << NPV << separator;
std::cout << std::setw(headers[2].size())
          << io::rate(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
          << io::rate(fairRate) << separator;
std::cout << std::endl;

QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
           "5-years swap mispriced!");

std::cout << rule << std::endl;

// the 1Y forward 5Y swap changes as well

std::cout << tab << "5-years, 1-year forward swap paying "
          << io::rate(fixedRate) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;
std::cout << rule << std::endl;


forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-swap" << separator;
```

```
        std::cout << std::setw(headers[1].size())
                  << std::fixed << std::setprecision(2) << NPV << separator;
        std::cout << std::setw(headers[2].size())
                  << io::rate(fairSpread) << separator;
        std::cout << std::setw(headers[3].size())
                  << io::rate(fairRate) << separator;
        std::cout << std::endl;


        forecastingTermStructure.linkTo(depoFutSwapTermStructure);
        discountingTermStructure.linkTo(depoFutSwapTermStructure);

        NPV = oneYearForward5YearSwap.NPV();
        fairSpread = oneYearForward5YearSwap.fairSpread();
        fairRate = oneYearForward5YearSwap.fairRate();

        std::cout << std::setw(headers[0].size())
                  << "depo-fut-swap" << separator;
        std::cout << std::setw(headers[1].size())
                  << std::fixed << std::setprecision(2) << NPV << separator;
        std::cout << std::setw(headers[2].size())
                  << io::rate(fairSpread) << separator;
        std::cout << std::setw(headers[3].size())
                  << io::rate(fairRate) << separator;
        std::cout << std::endl;


        forecastingTermStructure.linkTo(depoFRASwapTermStructure);
        discountingTermStructure.linkTo(depoFRASwapTermStructure);

        NPV = oneYearForward5YearSwap.NPV();
        fairSpread = oneYearForward5YearSwap.fairSpread();
        fairRate = oneYearForward5YearSwap.fairRate();

        std::cout << std::setw(headers[0].size())
                  << "depo-FRA-swap" << separator;
        std::cout << std::setw(headers[1].size())
                  << std::fixed << std::setprecision(2) << NPV << separator;
        std::cout << std::setw(headers[2].size())
                  << io::rate(fairSpread) << separator;
        std::cout << std::setw(headers[3].size())
                  << io::rate(fairRate) << separator;
        std::cout << std::endl;

        return 0;

    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

9.7 tracing_example.cpp

This code exemplifies how to insert trace statements to follow the flow of program execution. When compiler under gcc 3.3 and run, the following program will output the following trace:

```
trace[1]: Entering int main()
trace[2]: Entering int foo(int)
trace[3]: Entering int Foo::bar(int)
trace[3]: i = 21
trace[3]: At line 16 in tracing_example.cpp
trace[3]: Wrong answer
trace[3]: i = 42
trace[3]: Exiting int Foo::bar(int)
trace[3]: Entering int Foo::bar(int)
trace[3]: i = 42
trace[3]: At line 13 in tracing_example.cpp
trace[3]: Right answer, but no question
trace[3]: i = 42
trace[3]: Exiting int Foo::bar(int)
trace[2]: Exiting int foo(int)
trace[1]: Exiting int main()
```

Of course, a word of warning must be added: adding so much tracing to your code might degrade its readability, at least until we devise an Emacs macro to hide trace statements with a couple of keystrokes.

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

using namespace QuantLib;

namespace Foo {

    int bar(int i) {
        QL_TRACE_ENTER_FUNCTION;
        QL_TRACE_VARIABLE(i);

        if (i == 42) {
            QL_TRACE_LOCATION;
            QL_TRACE("Right answer, but no question");
        } else {
            QL_TRACE_LOCATION;
            QL_TRACE("Wrong answer");
            i *= 2;
        }

        QL_TRACE_VARIABLE(i);
        QL_TRACE_EXIT_FUNCTION;
        return i;
    }

}

int foo(int i) {
    using namespace Foo;
    QL_TRACE_ENTER_FUNCTION;

    int j = bar(i);
    int k = bar(j);

    QL_TRACE_EXIT_FUNCTION;
    return k;
}

int main() {
```

```
    QL_TRACE_ENABLE;

    QL_TRACE_ENTER_FUNCTION;

    int i = foo(21);

    QL_TRACE_EXIT_FUNCTION;
    return 0;
}
```

Chapter 10

Test List

Class ActualActual the correctness of the results is checked against known good values.

Class AnalyticBarrierEngine the correctness of the returned value is tested by reproducing results available in literature.

Class AnalyticCliquetEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class AnalyticContinuousGeometricAveragePriceAsianEngine
- the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class AnalyticDigitalAmericanEngine
- the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

Class AnalyticDiscreteGeometricAveragePriceAsianEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the available greeks is tested against numerical calculations.

Class AnalyticDividendEuropeanEngine the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class AnalyticEuropeanEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the implied-volatility calculation is tested by checking that it does not modify the option.
- the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Class AnalyticHestonEngine the correctness of the returned value is tested by reproducing results available in web/literature and comparison with Black pricing.

Class AnalyticPerformanceEngine the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class Array construction of arrays is checked in a number of cases

Class BaroneAdesiWhaleyApproximationEngine the correctness of the returned value is tested by reproducing results available in literature.

Class BatesEngine the correctness of the returned value is tested by reproducing results available in web/literature, testing against QuantLib's jump diffusion engine and comparison with Black pricing.

Class BatesModel calibration is tested against known values.

Class BinomialVanillaEngine the correctness of the returned value is tested by checking it against analytic results.

Class Bisection the correctness of the returned values is tested by checking them against known good results.

Class BivariateCumulativeNormalDistributionDr78 the correctness of the returned value is tested by checking it against known good results.

Class BivariateCumulativeNormalDistributionWe04DP the correctness of the returned value is tested by checking it against known good results.

Class BjerksundStenslandApproximationEngine the correctness of the returned value is tested by reproducing results available in literature.

Class Bond
- price/yield calculations are cross-checked for consistency.
- price/yield calculations are checked against known good values.

Class Brent the correctness of the returned values is tested by checking them against known good results.

Class BSMTermOperator coefficients are tested against constant BSM operator

Class Calendar the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Class CapFloor
- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

Class CapletLiborMarketModelProcess the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

Class CompositeQuote the correctness of the returned values is tested by checking them against numerical calculations.

Class CompoundForward
- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Class ConvergenceStatistics results are tested against known good values.

Class CovarianceDecomposition cross checked with getCovariance

Class CubicSpline the correctness of the returned values is tested by reproducing results available in literature.

Class CumulativePoissonDistribution the correctness of the returned value is tested by checking it against known good results.

Class Date self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Class DerivedQuote the correctness of the returned values is tested by checking them against numerical calculations.

Class DPlusDMinus the correctness of the returned values is tested by checking them against numerical calculations.

Class DZero the correctness of the returned values is tested by checking them against numerical calculations.

Class ExchangeRate application of direct and derived exchange rate is tested against calculations.

Class ExchangeRateManager lookup of direct, triangulated, and derived exchange rates is tested.

Class Factorial the correctness of the returned value is tested by checking it against numerical calculations.

Class FalsePosition the correctness of the returned values is tested by checking them against known good results.

Class FaureRsg the correctness of the returned values is tested by reproducing known good values.

Class FDAmericanEngine

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class FDDividendAmericanEngine
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

Class FDDividendEuropeanEngine
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

Class FDEuropeanEngine the correctness of the returned value is tested by checking it against analytic results.

Class FDShoutEngine the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class FixedCouponBond calculations are tested by checking results against cached values.

Class FloatingRateBond calculations are tested by checking results against cached values.

Class ForwardEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class ForwardPerformanceEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class ForwardSpreadedTermStructure
- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Class GammaFunction the correctness of the returned value is tested by checking it against known good results.

Class GaussianQuadrature the correctness of the result is tested by checking it against known good values.

Class Germany the correctness of the returned results is tested against a list of known holidays.

Class HaltonRsg
- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class HestonModel calibration is tested against known good values.

Class HullWhite calibration results are tested against cached values

Class ImpliedTermStructure
- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure is checked.

Class InArrearIndexedCoupon The class is tested by comparing the value of an in-arrear swap against a known good value.

Class Instrument observability of class instances is checked.

Class InterestRate Converted rates are checked against known good results

Class InverseCumulativePoisson the correctness of the returned value is tested by checking it against known good results.

Class Italy the correctness of the returned results is tested against a list of known holidays.

Class JointCalendar the correctness of the returned results is tested by reproducing the calculations.

Class JumpDiffusionEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class JuQuadraticApproximationEngine the correctness of the returned value is tested by reproducing results available in literature.

Class KronrodIntegral the correctness of the result is tested by checking it against known good values.

Member pseudoSqrt
- the correctness of the results is tested by reproducing known good data.

- the correctness of the results is tested by checking returned values against numerical calculations.

Class MCBarrierEngine the correctness of the returned value is tested by reproducing results available in literature.

Class MCBasketEngine the correctness of the returned value is tested by reproducing results available in literature.

Class MCDigitalEngine the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Class MCDiscreteArithmeticAPEngine the correctness of the returned value is tested by reproducing results available in literature.

Class MCDiscreteGeometricAPEngine the correctness of the returned value is tested by reproducing results available in literature.

Class MCEuropeanEngine the correctness of the returned value is tested by checking it against analytic results.

Class MCEuropeanHestonEngine the correctness of the returned value is tested by reproducing results available in web/literature

Class MersenneTwisterUniformRng the correctness of the returned values is tested by checking them against known good results.

Class Money money arithmetic is tested with and without currency conversions.

Class MultiCubicSpline interpolated values are checked against the original function.

Class MultiPathGenerator the generated paths are checked against cached results

Class Newton the correctness of the returned values is tested by checking them against known good results.

Class NewtonSafe the correctness of the returned values is tested by checking them against known good results.

Class NormalDistribution the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the CumulativeNormalDistribution and InverseCumulativeNormal classes.

Class PathGenerator the generated paths are checked against cached results

Class PiecewiseYieldCurve
- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Class PoissonDistribution the correctness of the returned value is tested by checking it against known good results.

Class QuantoEngine
- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class Quote the observability of class instances is tested.

Class RamdomizedLDS correct initialization is tested.

Class Ridder the correctness of the returned values is tested by checking them against known good results.

Class Rounding the correctness of the returned values is tested by checking them against known good results.

Class Secant the correctness of the returned values is tested by checking them against known good results.

Class SeedGenerator correct initializaion of the single instance is tested.

Class SegmentIntegral the correctness of the result is tested by checking it against known good values.

Class SequenceStatistics the correctness of the returned values is tested by checking them against numerical calculations.

Class SimpleDayCounter the correctness of the results is checked against known good values.

Class SimpleSwap
- the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- the correctness of the returned value is tested by checking it against a known good value.

Class SimpsonIntegral the correctness of the result is tested by checking it against known good values.

Class SobolRsg
- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class StulzEngine the correctness of the returned value is tested by reproducing results available in literature.

Class SVD the correctness of the returned values is tested by checking their properties.

Class Swaption
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.

Class SymmetricSchurDecomposition the correctness of the returned values is tested by checking their properties.

Class TARGET the correctness of the returned results is tested against a list of known holidays.

Class TqrEigenDecomposition the correctness of the result is tested by checking it against known good values.

Class TrapezoidIntegral the correctness of the result is tested by checking it against known good values.

Class TreeSwaptionEngine calculations are checked against cached results

Class UnitedKingdom the correctness of the returned results is tested against a list of known holidays.

Class UnitedStates the correctness of the returned results is tested against a list of known holidays.

Class YieldTermStructure observability against evaluation date changes is checked.

Class ZeroCouponBond calculations are tested by checking results against cached values.

Class ZeroSpreadedTermStructure
- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Chapter 11

Todo List

Class AmericanCondition unify the intrinsicValues/Payoff thing

Class AmericanExercise check that everywhere the American condition is applied from earliestDate and not earlier

Class AmericanPayoffAtExpiry calculate greeks

Class AmericanPayoffAtHit calculate greeks

Class AnalyticBarrierEngine rework to avoid repeated casts inside utility methods

Class AnalyticContinuousGeometricAveragePriceAsianEngine handle seasoned options

Class AnalyticDigitalAmericanEngine add more greeks (as of now only delta and rho available)

Class AnalyticDiscreteGeometricAveragePriceAsianEngine implement correct theta, rho, and dividend-rho calculation

Class BermudanExercise it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

Class BicubicSpline revise end conditions

Class BivariateCumulativeNormalDistributionDr78 check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Member QuantLib::BlackScholesProcess::drift(Time t, Real x) const revise extrapolation

Member QuantLib::BlackScholesProcess::diffusion(Time t, Real x) const revise extrapolation

Class BlackVarianceCurve check time extrapolation

Class BlackVarianceSurface check time extrapolation

Member QuantLib::BoundaryCondition::Side Generalize for n-dimensional conditions

Class CapVolatilityVector either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Class Cashflows add tests

Class Cdor check settlement days and day-count convention.

Class CliquetOption
- add local/global caps/floors
- add accrued coupon and last fixing

Class ContinuousAveragingAsianOption add running average

Class DirichletBC generalize to time-dependent conditions.

Class DiscreteGeometricASO add analytical greeks

Class EarlyExercise derive a plain American Exercise class (no earliestDate, no payoffAtExpiry)

Class ExplicitEuler add Richardson extrapolation

Class FraRateHelper convexity adjustment should be implemented.

Class GenericRiskStatistics add historical annualized volatility

Class IntegralEngine define tolerance for calculate()

Class Jibar check settlement days and day-count convention.

Class LogLinearInterpolation implement primitive, derivative, and secondDerivative functions.

Member pseudoSqrt
- implement Hypersphere decomposition:
  1. Jäckel "Monte Carlo Methods in Finance", Chapter 6
  2. Brigo "A Note on Correlation and Rank Reduction"
  3. Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
- implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Class McDiscreteArithmeticASO continous-averaging version

Class MixedScheme
- derive variable theta schemes
- introduce multi time-level schemes.

Class MultiCubicSpline
- fix it for Borland compilation
- allow extrapolation as for the other interpolations
- investigate if and how to implement Hyman filters and different boundary conditions

Class NeumannBC generalize to time-dependent conditions.

Class Option::arguments
- remove std::vector<Time> stoppingTimes
- how to handle strike-less option (asian average strike, forward, etc.)?

Class RamdomizedLDS implement the other randomization algorithms

Class ShoutCondition unify the intrinsicValues/Payoff thing

Class Solver1D
- clean up the interface so that it is clear whether the accuracy is specified for x or $f(x)$.
- add target value (now the target value is 0.0)

Class SwapRateHelper currency and day counter of Xibor should be added to obtain well-defined SwapRateHelper

Warning:
This class assumes that the settlement date does not change between calls of setTermStructure().

Class Swaption add explicit exercise lag

Class SwaptionVolatilityMatrix either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Class Tibor check settlement days.

Class TimeGrid what was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Class UnitedKingdom add LIFFE

Class YieldTermStructure add derived class ParSwapTermStructure similar to ZeroYieldTermStructure, DiscountStructure, ForwardRateStructure

Class Zibor check settlement days and day-count.

Chapter 12

Bug List

Class BlackFormula When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

Class BPSBasketCalculator this class must still be checked. It is not guaranteed to yield the right results.

Class CompoundForward swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

Class CoxIngersollRoss this class was not tested enough to guarantee its functionality.

Class ExtendedCoxIngersollRoss this class was not tested enough to guarantee its functionality.

Class FDDividendAmericanEngine method impliedVolatility() utterly fails

Class G2 This class was not tested enough to guarantee its functionality.

Class HullWhite When the term structure is relinked, the r0 parameter of the underlying Vasicek model is not updated.

Class JuQuadraticApproximationEngine test fails for Borland compiler

Class LocalVolSurface this class is untested, probably unreliable.

Class MCAmericanBasketEngine this engine does not yet work for put options. More problems might surface.

Class MultiCubicSpline
- cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- it does not compile under Borland

Member QuantLib::Swap::sensitivity(Integer basis=2) const this method must still be checked. It is not guaranteed to yield the right results.

Chapter 13

Deprecated List

Class MultiAsset use MultiVariate instead

Class SingleAsset use SingleVariate instead

Member QuantLib::YieldTermStructure::parRate(Year tenor, Time t0, Frequency freq=Annual, bool extrapolate=
use the overload taking a vector of times

Member IMMMonth use IMM::Month instead

Index