Package ‘RQuantLib’

February 19, 2015

**Title**  R Interface to the QuantLib Library

**Version**  0.4.0

**Date**  2014-12-01

**Maintainer**  Dirk Eddelbuettel <edd@debian.org>

**Author**  Dirk Eddelbuettel, Khanh Nguyen (during 2009-2010)

**Description**  The RQuantLib package makes parts of QuantLib accessible from R

The QuantLib project aims to provide a comprehensive software framework for quantitative finance. The goal is to provide a standard open source library for quantitative analysis, modeling, trading, and risk management of financial assets.

**Depends**  R (>= 2.10.0)

**Suggests**  rgl, zoo, RUnit

**Imports**  methods, Rcpp (>= 0.11.0)

**LinkingTo**  Rcpp

**SystemRequirements**  QuantLib library (>= 1.4.0) from http://quantlib.org, Boost library from http://www.boost.org

**License**  GPL (>= 2)

**URL**  http://quantlib.org

http://dirk.eddelbuettel.com/code/rquantlib.html

**NeedsCompilation**  yes

**Repository**  CRAN

**Date/Publication**  2014-12-02 12:18:45

**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmericanOption</td>
<td>2</td>
</tr>
<tr>
<td>AmericanOptionImpliedVolatility</td>
<td>4</td>
</tr>
<tr>
<td>AsianOption</td>
<td>5</td>
</tr>
<tr>
<td>BarrierOption</td>
<td>7</td>
</tr>
<tr>
<td>BermudanSwaption</td>
<td>9</td>
</tr>
</tbody>
</table>
AmericanOption

Description

This function evaluates an American-style option on a common stock using finite differences. The option value as well as the common first derivatives ("Greeks") are returned.

Usage

```r
## Default S3 method:
AmericanOption(type, underlying, strike,
    dividendYield, riskFreeRate, maturity, volatility,
    timeSteps=150, gridPoints=149, engine="BaroneAdesiWhaley")
```

Arguments

- `type` A string with one of the values call or put
- `underlying` Current price of the underlying stock
- `strike` Strike price of the option
- `dividendYield` Continuous dividend yield (as a fraction) of the stock
- `riskFreeRate` Risk-free rate
- `maturity` Time to maturity (in fractional years)
- `volatility` Volatility of the underlying stock
AmericanOption

timeSteps  Time steps for the “CrankNicolson” finite differences method engine, default value is 150
gridPoints  Grid points for the “CrankNicolson” finite differences method, default value is 149
genre  String selecting pricing engine, currently supported are “BaroneAdesiWhaley” and “CrankNicolson”

Details

The Finite Differences method is used to value the American Option.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

An object of class AmericanOption (which inherits from class Option) is returned. It contains a list with the following components:

- value: Value of option
- delta: Sensitivity of the option value for a change in the underlying
- gamma: Sensitivity of the option delta for a change in the underlying
- vega: Sensitivity of the option value for a change in the underlying’s volatility
- theta: Sensitivity of the option value for a change in t, the remaining time to maturity
- rho: Sensitivity of the option value for a change in the risk-free interest rate
- dividendRho: Sensitivity of the option value for a change in the dividend yield

Note that under the new pricing framework used in QuantLib, pricers do not provide analytics for all 'Greeks'. When “CrankNicolson” is selected, then at least delta, gamma and vega are available. With the default pricing engine of “BaroneAdesiWhaley”, no greeks are returned.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

EuropeanOption
Examples

# simple call with unnamed parameters
AmericanOption("call", 100, 100, 0.02, 0.03, 0.5, 0.4)

# simple call with some explicit parameters
AmericanOption("call", strike=100, volatility=0.4, 100, 0.02, 0.03, 0.5)

# simple call with unnamed parameters, using Crank-Nicolson
AmericanOption("call", strike=100, volatility=0.4, 100, 0.02, 0.03, 0.5, engine="CrankNicolson")

AmericanOptionImpliedVolatility

Implied Volatility calculation for American Option

Description

The AmericanOptionImpliedVolatility function solves for the (unobservable) implied volatility, given an option price as well as the other required parameters to value an option.

Usage

## Default S3 method:
AmericanOptionImpliedVolatility(type, value,
underlying, strike, dividendYield, riskFreeRate, maturity, volatility,
timeSteps=150, gridPoints=151)

Arguments

type A string with one of the values call or put
value Value of the option (used only for ImpliedVolatility calculation)
underlying Current price of the underlying stock
strike Strike price of the option
dividendYield Continuous dividend yield (as a fraction) of the stock
riskFreeRate Risk-free rate
maturity Time to maturity (in fractional years)
volatility Initial guess for the volatility of the underlying stock
timeSteps Time steps for the Finite Differences method, default value is 150
gridPoints Grid points for the Finite Differences method, default value is 151

Details

The Finite Differences method is used to value the American Option. Implied volatilities are then calculated numerically.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.
AsianOption

Value

The `AmericanOptionImpliedVolatility` function returns a numeric variable with volatility implied by the given market prices and given parameters.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

`EuropeanOption`, `AmericanOption`, `BinaryOption`

Examples

```r
AmericanOptionImpliedVolatility(type="call", value=11.10, underlying=100, strike=100, dividendYield=0.01, riskFreeRate=0.03, maturity=0.5, volatility=0.4)
```

---

AsianOption

Asian Option evaluation using Closed-Form solution

Description

The `AsianOption` function evaluates an Asian-style option on a common stock using an analytic solution for continuous geometric average price. The option value, the common first derivatives ("Greeks") as well as the calling parameters are returned.

Usage

```r
## Default S3 method:
AsianOption(averageType, type, underlying, strike,
           dividendYield, riskFreeRate, maturity,
           volatility, first=0, length=11.0/12.0, fixings=26)
```
**AsianOption**

**Arguments**

- **averageType**: Specify averaging type, either “geometric” or “arithmetic”
- **type**: A string with one of the values call or put
- **underlying**: Current price of the underlying stock
- **strike**: Strike price of the option
- **dividendYield**: Continuous dividend yield (as a fraction) of the stock
- **riskFreeRate**: Risk-free rate
- **maturity**: Time to maturity (in fractional years)
- **volatility**: Volatility of the underlying stock
- **first** (Only for arithmetic averaging): Time step to first average, can be zero
- **length** (Only for arithmetic averaging): Total time length for averaging period
- **fixings** (Only for arithmetic averaging): Total number of averaging fixings

**Details**

When "arithmetic" evaluation is used, only the NPV() is returned.

The well-known closed-form solution derived by Black, Scholes and Merton is used for valuation. Implied volatilities are calculated numerically.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

**Value**

The AsianOption function returns an object of class AsianOption (which inherits from class Option). It contains a list with the following components:

- **value**: Value of option
- **delta**: Sensitivity of the option value for a change in the underlying
- **gamma**: Sensitivity of the option delta for a change in the underlying
- **vega**: Sensitivity of the option value for a change in the underlying’s volatility
- **theta**: Sensitivity of the option value for a change in t, the remaining time to maturity
- **rho**: Sensitivity of the option value for a change in the risk-free interest rate
- **dividendRho**: Sensitivity of the option value for a change in the dividend yield

**Note**

The interface might change in future release as QuantLib stabilises its own API.

**Author(s)**

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib
References

http://quantlib.org for details on QuantLib.

Examples

# simple call with some explicit parameters, and slightly increased vol:
AsianOption("geometric", "put", underlying=80, strike=85, div=-0.03,
            riskFree=0.05, maturity=0.25, vol=0.2)

Description

This function evaluates an Barrier option on a common stock using a closed-form solution. The option value as well as the common first derivatives ("Greeks") are returned.

Usage

## Default S3 method:
BarrierOption(barrType, type, underlying, strike,
              dividendYield, riskFreeRate, maturity,
              volatility, barrier, rebate=0.0)

Arguments

barrType A string with one of the values downin, downout, upin or upout
type A string with one of the values call or put
underlying Current price of the underlying stock
strike Strike price of the option
dividendYield Continuous dividend yield (as a fraction) of the stock
riskFreeRate Risk-free rate
maturity Time to maturity (in fractional years)
volatility Volatility of the underlying stock
barrier Option barrier value
rebate Optional option rebate, defaults to 0.0

Details

A closed-form solution is used to value the Barrier Option. In the case of Barrier options, the calculations are from Haug’s "Option pricing formulas" book (McGraw-Hill).

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.
BarrierOption

Value

An object of class BarrierOption (which inherits from class Option) is returned. It contains a list with the following components:

- **value**: Value of option
- **delta**: Sensitivity of the option value for a change in the underlying
- **gamma**: Sensitivity of the option delta for a change in the underlying
- **vega**: Sensitivity of the option value for a change in the underlying’s volatility
- **theta**: Sensitivity of the option value for a change in t, the remaining time to maturity
- **rho**: Sensitivity of the option value for a change in the risk-free interest rate
- **dividendRho**: Sensitivity of the option value for a change in the dividend yield

Note that under the new pricing framework used in QuantLib, binary pricers do not provide analytics for 'Greeks'. This is expected to be addressed in future releases of QuantLib.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

AmericanOption, EuropeanOption

Examples

BarrierOption(barrType="downin", type="call", underlying=100, strike=100, dividendYield=0.02, riskFreeRate=0.03, maturity=0.5, volatility=0.4, barrier=90)
BermudanSwaption

Description

BermudanSwaption prices a Bermudan swaption with specified strike and maturity (in years), after calibrating the selected short-rate model to an input swaption volatility matrix. Swaption maturities are in years down the rows, and swap tenors are in years along the columns, in the usual fashion. It is assumed that the Bermudan swaption is exercisable on each reset date of the underlying swaps.

Usage

BermudanSwaption(params, tsQuotes, swaptionMaturities, swapTenors, volMatrix)

Arguments

- **params**: A list specifying the tradeDate (month/day/year), settlementDate, payFixed flag, strike, pricing method, and curve construction options (see Examples section below). Curve construction options are interpWhat (possible values are discount, forward, and zero) and interpHow (possible values are linear, loglinear, and spline). Both interpWhat and interpHow are ignored when a flat yield curve is requested, but they must be present nevertheless. The pricing method can be one of the following (all short-rate models):
  - G2Analytic: G2 2-factor Gaussian model using analytic formulas.
  - HWAnalytic: Hull-White model using analytic formulas.
  - HWTree: Hull-White model using a tree.
  - BKTree: Black-Karasinski model using a tree.

- **tsQuotes**: Market observables needed to construct the spot term structure of interest rates. A list of name/value pairs. See the help page for DiscountCurve for details.

- **swaptionMaturities**: A vector containing the swaption maturities associated with the rows of the swaption volatility matrix.

- **swapTenors**: A vector containing the underlying swap tenors associated with the columns of the swaption volatility matrix.

- **volMatrix**: The swaption volatility matrix. Must be a 2D matrix stored by rows. See the example below.

Details

This function is based on QuantLib Version 0.3.10. It introduces support for fixed-income instruments in RQuantLib.

At present only a small number of the many parameters that can be set in QuantLib are exposed by this function. Some of the hard-coded parameters that apply to the current version include:
day-count conventions, fixing days (2), index (Euribor), fixed leg frequency (annual), and floating leg frequency (semi-annual). Also, it is assumed that the swaption volatility matrix corresponds to expiration dates and tenors that are measured in years (a 6-month expiration date is not currently supported, for example).

Given the number of parameters that must be specified and the care with which they must be specified (with no defaults), it is not practical to use this function in the usual interactive fashion.

The simplest approach is simply to save the example below to a file, edit as desired, and source the result. Alternatively, the input commands can be kept in a script file (under Windows) or an Emacs/ESS session (under Linux), and selected parts of the script can be executed in the usual way.

Fortunately, the C++ exception mechanism seems to work well with the R interface, and QuantLib exceptions are propagated back to the R user, usually with a message that indicates what went wrong. (The first part of the message contains technical information about the precise location of the problem in the QuantLib code. Scroll to the end to find information that is meaningful to the R user.)

Value

BermudanSwaption returns a list containing calibrated model parameters (what parameters are returned depends on the model selected) along with:

- price: Price of swaption in basis points (actual price equals price times notional divided by 10,000)
- ATMStrike: At-the-money strike
- params: Input parameter list

Author(s)

Dominick Samperi

References


For information about QuantLib see http://quantlib.org.


See Also

- DiscountCurve

Examples

# This data is taken from sample code shipped with QuantLib 0.3.10.
params <- list(tradeDate=as.Date('2002-2-15'),
              settleDate=as.Date('2002-2-19'),
              payFixed=TRUE,
              strike=0.06,
              ...)

C this data is taken from sample code shipped with QuantLib PNSNQPN
params \M listHtradedate\]asNdateH
G
RPPRMRMQYG
G
IL
settledate\]asNdateH
G
RPPRMRMQYG
G
IL
payfixed\]trueL
strike\]NPVL
BinaryOption

Description

This function evaluates an Binary option on a common stock using a closed-form solution. The option value as well as the common first derivatives ("Greeks") are returned.
Usage

```r
## Default S3 method:
BinaryOption(binType, type, excType, underlying, 
  strike, dividendYield, 
  riskFreeRate, maturity, volatility, cashPayoff)
```

Arguments

- `binType` A string with one of the values cash, asset or gap to select CashOrNothing, AssetOrNothing or Gap payoff profiles
- `type` A string with one of the values call or put
- `excType` A string with one of the values european or american to denote the exercise type
- `underlying` Current price of the underlying stock
- `strike` Strike price of the option
- `dividendYield` Continuous dividend yield (as a fraction) of the stock
- `riskFreeRate` Risk-free rate
- `maturity` Time to maturity (in fractional years)
- `volatility` Volatility of the underlying stock
- `cashPayoff` Payout amount

Details

A closed-form solution is used to value the Binary Option.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

An object of class BinaryOption (which inherits from class `Option`) is returned. It contains a list with the following components:

- `value` Value of option
- `delta` Sensitivity of the option value for a change in the underlying
- `gamma` Sensitivity of the option delta for a change in the underlying
- `vega` Sensitivity of the option value for a change in the underlying’s volatility
- `theta` Sensitivity of the option value for a change in t, the remaining time to maturity
- `rho` Sensitivity of the option value for a change in the risk-free interest rate
- `dividendRho` Sensitivity of the option value for a change in the dividend yield

Note

The interface might change in future release as QuantLib stabilises its own API.
The `binaryoptionimpliedvolatility` function solves for the (unobservable) implied volatility, given an option price as well as the other required parameters to value an option.

### Examples

```r
BinaryOption(binType="asset", type="call", excType="european",
      underlying=100, strike=100, dividendYield=0.02,
      riskFreeRate=0.03, maturity=0.5, volatility=0.4, cashPayoff=1)
```

---

**Description**

The `binaryoptionimpliedvolatility` function solves for the (unobservable) implied volatility, given an option price as well as the other required parameters to value an option.

**Usage**

```r
## Default S3 method:
BinaryOptionImpliedVolatility(type, value, underlying,
       strike, dividendYield, riskFreeRate, maturity, volatility,
       cashPayoff=1)
```

**Arguments**

- `type`: A string with one of the values call, put or straddle
- `value`: Value of the option (used only for ImpliedVolatility calculation)
- `underlying`: Current price of the underlying stock
- `strike`: Strike price of the option
- `dividendYield`: Continuous dividend yield (as a fraction) of the stock
- `riskFreeRate`: Risk-free rate
- `maturity`: Time to maturity (in fractional years)
- `volatility`: Initial guess for the volatility of the underlying stock
- `cashPayoff`: Binary payout if options is exercised, default is 1
Details

The Finite Differences method is used to value the Binary Option. Implied volatilities are then calculated numerically.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The BinaryOptionImpliedVolatility function returns an numeric variable with volatility implied by the given market prices.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

EuropeanOption, AmericanOption, BinaryOption

Examples

BinaryOptionImpliedVolatility("call", value=4.50, strike=100, 100, 0.02, 0.03, 0.5, 0.4, 10)

---

Bond

Base class for Bond price evaluation

Description

This class forms the basis from which the more specific classes are derived.

Usage

```r
## S3 method for class 'Bond'
print(x, digits=5, ...)
## S3 method for class 'FixedRateBond'
print(x, digits=5, ...)
## S3 method for class 'Bond'
plot(x, ...)
## S3 method for class 'Bond'
summary(object, digits=5, ...)
```
**Bond**

**Arguments**

- **x**  
  Any Bond object derived from this base class
- **object**  
  Any Bond object derived from this base class
- **digits**  
  Number of digits of precision shown
- **...**  
  Further arguments

**Details**

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

**Value**

None, but side effects of displaying content.

**Note**

The interface might change in future release as QuantLib stabilises its own API.

**Author(s)**

Khanh Nguyen <knguyen@cs.umb.edu>; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

**References**

http://quantlib.org for details on QuantLib.

**Examples**

```r
## This data is taken from sample code shipped with QuantLib 0.9.7
## from the file Examples/Swap/swapvaluation
params <- list(tradeDate=as.Date('2004-09-20'),
               settleDate=as.Date('2004-09-22'),
               dt=.25,
               interpWhat="discount",
               interpHow="loglinear")
setEvaluationDate(as.Date("2004-09-20"))

## We got numerical issues for the spline interpolation if we add
## any on of these three extra futures, at least with QuantLib 0.9.7
## The curve data comes from QuantLib's Examples/Swap/swapvaluation.cpp
## Removing s2y helps, as kindly pointed out by Luigi Ballabio
## This data is taken from sample code shipped with QuantLib 0.9.7
## from the file Examples/Swap/swapvaluation
params <- list(tradeDate=as.Date('2004-09-20'),
               settleDate=as.Date('2004-09-22'),
               dt=.25,
               interpWhat="discount",
               interpHow="loglinear")
setEvaluationDate(as.Date("2004-09-20"))
```

```r
tsQuotes <- list(d1w = 0.0382,
                 d1m = 0.0372,
                 fut1=96.2875,
                 fut2=96.7875,
                 fut3=96.9875,
                 fut4=96.6875,
                 ```
fut5=96.4875,
fut6=96.3875,
fut7=96.2875,
fut8=96.0875,
# s2y = 0.037125,  ## s2y perturbs
s3y = 0.0398,
s5y = 0.0443,
s10y = 0.05165,
s15y = 0.055175)
times <- seq(0,10,.1)

setEvaluationDate(params$tradeDate)
discountCurve <- DiscountCurve(params, tsQuotes, times)

# price a zero coupon bond
bondparams <- list(faceAmount=100, issueDate=as.Date("2004-11-30"),
maturityDate=as.Date("2008-11-30"), redemption=100 )
dateparams <-list(settlementDays=1,
    calendar="UnitedStates/GovernmentBond",
    businessDayConvention=4)
ZeroCouponBond(bondparams, discountCurve, dateparams)

# price a fixed rate coupon bond
bond <- list(settlementDays=1, issueDate=as.Date("2004-11-30"),
    faceAmount=100, accrualDayCounter='Thirty360',
    paymentConvention='Unadjusted')
schedule <- list(effectiveDate=as.Date("2004-11-30"),
    maturityDate=as.Date("2008-11-30"),
    period='Semiannual',
    calendar='UnitedStates/GovernmentBond',
    businessDayConvention='Unadjusted',
    terminationDateConvention='Unadjusted',
    dateGeneration='Forward',
    endOfMonth=1)
calc=list(dayCounter='Actual360', compounding='Compounded',
    freq='Annual', durationType='Modified')
rates <- c(0.02875)
FixedRateBond(bond, rates, schedule, calc, discountCurve=discountCurve)

# price a fixed rate coupon bond from yield
yield <- 0.050517
FixedRateBond(bond, rates, schedule, calc, yield=yield) 

# calculate the same bond from the clean price

price <- 92.167
FixedRateBond(bond, rates, schedule, calc, price=price)

# price a floating rate bond
bondparams <- list(faceAmount=100, issueDate=as.Date("2004-11-30"),
maturityDate=as.Date("2008-11-30"), redemption=100,
effectiveDate=as.Date("2004-12-01")

dateparams <- list(settlementDays=1, calendar="UnitedStates/GovernmentBond", dayCounter = 1, period=3, businessDayConvention = 1, terminationDateConvention=1, dateGeneration=0, endOfMonth=0, fixingDays = 1)

gearings <- spreads <- caps <- floors <- vector()

iborCurve <- DiscountCurve(params, list(flat=0.05), times)

ibor <- list(type="USDLibor", length=6, inTermOf="Month", term=iborCurve)

FloatingRateBond(bondparams, gearings, spreads, caps, floors, ibor, discountCurve, dateparams)

**BondUtilities**

**Bond parameter conversion utilities**

**Description**

These functions are using internally to convert from the characters at the R level to the enum types used at the C++ level. They are documented here mostly to provide a means to look up some of the possible values—the user is not expected to call these functions directly.

**Usage**

matchBDC(bdc = c("Following", "ModifiedFollowing", "Preceding", "ModifiedPreceding", "Unadjusted"))

matchCompounding(cp = c("Simple", "Compounded", "Continuous", "SimpleThenCompounded"))


matchDateGen(dg = c("Backward", "Forward", "Zero", "ThirdWednesday", "Twentieth", "TwentiethIMM"))


matchParams(params)

**Arguments**

- **bdc** A string identifying one of the possible business day convention values.
- **cp** A string identifying one of the possible compounding frequency values.
- **daycounter** A string identifying one of the possible day counter scheme values.
- **dg** A string identifying one of the possible date generation scheme values.
- **freq** A string identifying one of the possible (dividend) frequency values.
- **params** A named vector containing the other parameters as components.
Details

The QuantLib documentation should be consulted for details.

Value

Each function converts the given character value into a corresponding numeric entry. For matchParams, an named vector of strings is converted into a named vector of numerics.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

---

**Calendars**

**Calendar functions from QuantLib**

Description

The isBusinessDay function evaluates the given dates in the context of the given calendar, and returns a vector of booleans indicating business day status. BusinessDay is also recognised (but may be deprecated one day).

The isHoliday function evaluates the given dates in the context of the given calendar, and returns a vector of booleans indicating holiday day status.

The isWeekend function evaluates the given dates in the context of the given calendar, and returns a vector of booleans indicating weekend status.

The isEndOfMonth function evaluates the given dates in the context of the given calendar, and returns a vector of booleans indicating end of month status.

The getEndOfMonth function evaluates the given dates in the context of the given calendar, and returns a vector that corresponds to the end of month. endOfMonth is a deprecated form for this function.

The getHolidayList function returns the holidays between the given dates, with an option to exclude weekends. holidayList is a deprecated form for this function.

The adjust function evaluates the given dates in the context of the given calendar, and returns a vector that adjusts each input dates to the appropriate near business day with respect to the given convention.

The advance function evaluates the given dates in the context of the given calendar, and returns a vector that advances the given dates of the given number of business days and returns the result. This functions gets called either with both argument n and TimeUnit, or with argument period.
The `businessDaysBetween` function evaluates two given dates in the context of the given calendar, and returns a vector that gives the number of business days between.

The `dayCount` function returns the number of day between two dates given a day counter, see `Enum`.

The `yearFraction` function returns year fraction between two dates given a day counter, see `Enum`.

The `setCalendarContext` function sets three values to a singleton instance at the C++ layer.

The `setEvaluationDate` function sets the evaluation date used by the QuantLib pricing engines.

The `advanceDate` function advances the given date by the given number of days in the current calendar instance.

**Usage**

```cpp
isBusinessDay(calendar, dates)
businessDay(calendar="TARGET", dates=Sys.Date()) # deprecated form
isHoliday(calendar, dates)
isWeekend(calendar, dates)
isEndOfMonth(calendar, dates)
getEndOfMonth(calendar, dates)
endOfMonth(calendar="TARGET", dates=Sys.Date())
getHolidayList(calendar, from, to, includeWeekends=FALSE)
holidayList(calendar="TARGET", from=Sys.Date(), to = Sys.Date() + 5,
includeWeekends = FALSE)
adjust(calendar, dates, bdc = 0L)
advance(calendar="TARGET", dates=Sys.Date(), n, timeUnit, period, bdc = 0, emr =0)

businessDaysBetween(calendar, from, to, includeFirst = TRUE, includeLast = FALSE)
dayCount(startDates, endDates, dayCounters)
yearFraction(startDates, endDates, dayCounters)
setCalendarContext(calendar, fixingDays, settleDate)
setEvaluationDate(evalDate)
```

**Arguments**

- `calendar` A string identifying one of the supported QuantLib calendars, see Details for more
- `dates` A vector (or scalar) of Date types.
- `from` A vector (or scalar) of Date types.
- `to` A vector (or scalar) of Date types.
- `includeWeekends` boolean that indicates whether the calculation should include the weekends. Default = false
- `fixingDays` An integer for the fixing day period, defaults to 2.
- `settleDate` A date on which trades settles, defaults to two days after the current day.
- `n` an integer number
- `timeUnit` A value of 0,1,2,3 that corresponds to Days, Weeks, Months, and Year; for more detail, see the QuantLib documentation at [http://quantlib.org/reference/group__datetime.html](http://quantlib.org/reference/group__datetime.html)
Calendars

- **Period**: See Enum
- **BDC**: Business day convention. By default, this value is 0 and correspond to Following convention
- **EMR**: End Of Month rule, default is false
- **IncludeFirst**: boolean that indicates whether the calculation should include the first day. Default = true
- **IncludeLast**: Default = false
- **StartDates**: A vector of Date type.
- **EndDates**: A vector of Date type.
- **DayCounters**: A vector of numeric type. See Enum
- **EvalDate**: A single date used for the pricing valuations.

**Details**

The calendars are coming from QuantLib, and the QuantLib documentation should be consulted for details.

Currently, the following strings are recognised: TARGET (a default calendar), Argentina, Australia, Brazil, Canada and Canada/Settlement, Canada/TSX, China, CzechRepublic, Denmark, Finland, Germany and Germany/FrankfurtStockExchange, Germany/Settlement, Germany/Xetra, Germany/Eurex, HongKong, Hungary, Iceland, India, Indonesia, Italy and Italy/Settlement, Italy/Exchange, Japan, Mexico, NewZealand, Norway, Poland, Russia, SaudiArabia, Singapore, Slovakia, SouthAfrica, SouthKorea, SouthKorea/KRX, Sweden, Switzerland, Taiwan, Turkey, Ukraine, UnitedKingdom and UnitedKingdom/Settlement, UnitedKingdom/Xetra, UnitedKingdom/Eurex, UnitedKingdom/Exchange, UnitedKingdom/Metals, UnitedStates and UnitedStates/Settlement, UnitedStates/NYSE, UnitedStates/GovernmentBond, UnitedStates/NERC and WeekendsOnly.

(In case of multiples entries per country, the country default is listed right after the country itself. Using the shorter form is equivalent.)

**Value**

A named vector of booleans each of which is true if the corresponding date is a business day (or holiday or weekend) in the given calendar. The element names are the dates (formatted as text in yyyy-mm-dd format).

For setCalendarContext, a boolean or NULL in case of error.

**Note**

The interface might change in future release as QuantLib stabilises its own API.

**Author(s)**

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

**References**

http://quantlib.org for details on QuantLib.
**Examples**

dates <- seq(from=as.Date("2009-04-07"), to=as.Date("2009-04-14"), by=1)
isBusinessDay("UnitedStates", dates)
isBusinessDay("UnitedStates/Settlement", dates)  ## same as previous
isBusinessDay("UnitedStates/NYSE", dates)  ## stocks
isBusinessDay("UnitedStates/GovernmentBond", dates)  ## bonds
isBusinessDay("UnitedStates/NERC", dates)  ## energy

isHoliday("UnitedStates", dates)
isHoliday("UnitedStates/Settlement", dates)  ## same as previous
isHoliday("UnitedStates/NYSE", dates)  ## stocks
isHoliday("UnitedStates/GovernmentBond", dates)  ## bonds
isHoliday("UnitedStates/NERC", dates)  ## energy

isWeekend("UnitedStates", dates)
isWeekend("UnitedStates/Settlement", dates)  ## same as previous
isWeekend("UnitedStates/NYSE", dates)  ## stocks
isWeekend("UnitedStates/GovernmentBond", dates)  ## bonds
isWeekend("UnitedStates/NERC", dates)  ## energy

isEndOfMonth("UnitedStates", dates)
isEndOfMonth("UnitedStates/Settlement", dates)  ## same as previous
isEndOfMonth("UnitedStates/NYSE", dates)  ## stocks
isEndOfMonth("UnitedStates/GovernmentBond", dates)  ## bonds
isEndOfMonth("UnitedStates/NERC", dates)  ## energy

getEndOfMonth("UnitedStates", dates)
getEndOfMonth("UnitedStates/Settlement", dates)  ## same as previous
getEndOfMonth("UnitedStates/NYSE", dates)  ## stocks
getEndOfMonth("UnitedStates/GovernmentBond", dates)  ## bonds
getEndOfMonth("UnitedStates/NERC", dates)  ## energy

from <- as.Date("2009-04-07")
to <- as.Date("2009-04-14")
getHolidayList("UnitedStates", from, to)
to <- as.Date("2009-10-7")
getHolidayList("UnitedStates", from, to)

dates <- seq(from=as.Date("2009-04-07"), to=as.Date("2009-04-14"), by=1)

adjust("UnitedStates", dates)
adjust("UnitedStates/Settlement", dates)  ## same as previous
adjust("UnitedStates/NYSE", dates)  ## stocks
adjust("UnitedStates/GovernmentBond", dates)  ## bonds
adjust("UnitedStates/NERC", dates)  ## energy

advance("UnitedStates", dates, 10, 0)
advance("UnitedStates/Settlement", dates, 10, 1)  ## same as previous
advance("UnitedStates/NYSE", dates, 10, 2)  ## stocks
advance("UnitedStates/GovernmentBond", dates, 10, 3)  ## bonds
advance("UnitedStates/NERC", dates, period = 3)  ## energy
CallableBond

CallableBond evaluation

Description

The CallableBond function sets up and evaluates a callable fixed rate bond using Hull-White model and a TreeCallableFixedBondEngine pricing engine. For more detail, see the source codes in quantlib's example folder, Examples/CallableBond/CallableBond.cpp

Usage

## Default S3 method:
CallableBond(bondparams, hullWhite, coupon, dateparams)

Arguments

bondparams a named list whose elements are:

- **issueDate** a Date, the bond’s issue date
- **maturityDate** a Date, the bond’s maturity date
- **faceAmount** (Optional) a double, face amount of the bond. Default value is 100.
- **redemption** (Optional) a double, percentage of the initial face amount that will be returned at maturity date. Default value is 100.
- **callSch** (Optional) a data frame whose columns are "Price", "Type" and "Date" corresponding to QuantLib’s CallabilitySchedule. Default is an empty frame, or no callability.

hullWhite a named list whose elements are parameters needed to set up a HullWhite pricing engine in QuantLib:

- **term** a double, to set up a flat rate yield term structure
- **alpha** a double, Hull-White model’s alpha value
- **sigma** a double, Hull-White model’s sigma value
- **gridIntervals.** a double, time intervals parameter to set up the TreeCallableFixedBondEngine
CallableBond

Currently, the codes only support a flat rate yield term structure. For more detail, see QuantLib's doc on HullWhite and TreeCallableFixedBondEngine.

coupon

a numeric vector of coupon rates

dateparams

(Optional) a named list, QuantLib’s date parameters of the bond.

settlementDays

(Optional) a double, settlement days. Default value is 1.

calendar

(Optional) a string, either 'us' or 'uk' corresponding to US Goverment Bond calendar and UK Exchange calendar. Default value is 'us'.

dayCounter

(Optional) a number or string, day counter convention. See Enum. Default value is 'Thirty360'

period

(Optional) a number or string, interest compounding interval. See Enum. Default value is 'Semiannual'.

businessDayConvention

(Optional) a number or string, business day convention. See Enum. Default value is 'Following'.

terminationDateConvention

(Optional) a number or string, termination day convention. See Enum. Default value is 'Following'.

See example below.

Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The CallableBond function returns an object of class CallableBond (which inherits from class Bond). It contains a list with the following components:

NPV        net present value of the bond
cleanPrice  price price of the bond
dirtyPrice  dirty price of the bond
accruedAmount  accrued amount of the bond
yield       yield of the bond
cashFlows   cash flows of the bond
Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

Examples

# set-up a HullWhite according to example from QuantLib
HullWhite <- list(term = 0.055, alpha = 0.03, sigma = 0.01, 
gridIntervals = 40)

# callability schedule dataframe
Price <- rep(as.double(100), 24)
Type <- rep(as.character("C"), 24)
Date <- seq(as.Date("2006-09-15"), by = '3 months', length = 24)
callSch <- data.frame(Price, Type, Date)
callSch$Type <- as.character(callSch$Type)

bondparams <- list(faceAmount=100, issueDate = as.Date("2004-09-16"), 
maturityDate=as.Date("2012-09-16"), redemption=100, 
callSch = callSch)
dateparams <- list(settlementDays=3, calendar="UnitedStates/GovernmentBond", 
dayCounter = "ActualActual", 
period="Quarterly", 
businessDayConvention = "Unadjusted", 
terminationDateConvention= "Unadjusted")
coupon <- c(0.0465)

CallableBond(bondparams, HullWhite, coupon, dateparams)
# examples using default values
CallableBond(bondparams, HullWhite, coupon)
dateparams <- list( 
    period="Quarterly", 
    businessDayConvention = "Unadjusted", 
    terminationDateConvention= "Unadjusted")
CallableBond(bondparams, HullWhite, coupon, dateparams)

bondparams <- list(issueDate = as.Date("2004-09-16"), 
maturityDate=as.Date("2012-09-16")
)
CallableBond(bondparams, HullWhite, coupon, dateparams)
Description

The ConvertibleFixedCouponBond function setups and evaluates a ConvertibleFixedCouponBond using QuantLib's BinomialConvertibleEngine and BlackScholesMertonProcess.

The NPV, clean price, dirty price, accrued interest, yield and cash flows of the bond is returned. For detail, see test-suite/convertiblebond.cpp.

The ConvertibleFloatingCouponBond function setups and evaluates a ConvertibleFixedCouponBond using QuantLib's BinomialConvertibleEngine and BlackScholesMertonProcess.

The NPV, clean price, dirty price, accrued interest, yield and cash flows of the bond is returned. For detail, see test-suite/convertiblebond.cpp.

The ConvertibleZeroCouponBond function setups and evaluates a ConvertibleFixedCouponBond using QuantLib's BinomialConvertibleEngine and BlackScholesMertonProcess.

The NPV, clean price, dirty price, accrued interest, yield and cash flows of the bond is returned. For detail, see test-suite/convertiblebond.cpp.

Usage

```c++
ConvertedBond default S3 method:
ConvertibleFloatingCouponBond(bondparams, iborindex, spread, process, dateparams)

ConvertibleFixedCouponBond(bondparams, coupon, process, dateparams)

ConvertibleZeroCouponBond(bondparams, process, dateparams)
```

Arguments

- `bondparams`: bond parameters, a named list whose elements are:
  - `issueDate`: a Date, the bond's issue date
  - `maturityDate`: a Date, the bond's maturity date
  - `creditSpread`: a double, credit spread parameter in the constructor of the bond.
  - `conversitionRatio`: a double, conversion ratio parameter in the constructor of the bond.
  - `exercise`: (Optional) a string, either "eu" for European option, or "am" for American option. Default value is 'am'.
  - `faceAmount`: (Optional) a double, face amount of the bond.
Default value is 100.

redeemption (Optional) a double, percentage of the initial face amount that will be returned at maturity date. Default value is 100.

divSch (Optional) a data frame whose columns are "Type", "Amount", "Rate", and "Date" corresponding to QuantLib's DividendSchedule. Default value is an empty frame, or no dividend.

callSch (Optional) a data frame whose columns are "Price", "Type" and "Date" corresponding to QuantLib's CallabilitySchedule. Default is an empty frame, or no callability.

iborindex a DiscountCurve object, represents an IborIndex

spread a double vector, represents parameter 'spreads' in ConvertibleFloatingBond's constructor.

coupon a double vector of coupon rate

process arguments to construct a BlackScholes process and set up the binomial pricing engine for this bond.

underlying a double, flat underlying term structure

volatility a double, flat volatility term structure

dividendYield a DiscountCurve object

riskFreeRate a DiscountCurve object

dateparams (Optional) a named list, QuantLib's date parameters of the bond.

settlementDays (Optional) a double, settlement days.

Default value is 1.

calendar (Optional) a string, either 'us' or 'uk' corresponding to US Government Bond calendar and UK Exchange calendar.

Default value is 'us'.

dayCounter (Optional) a number or string, day counter convention.

See Enum. Default value is 'Thirty360'

period (Optional) a number or string, interest compounding interval. See Enum.

Default value is 'Semiannual'.

businessDayConvention (Optional) a number or string, business day convention.

See Enum. Default value is 'Following'.

See the examples below.
Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The ConvertibleFloatingCouponBond function returns an object of class ConvertibleFloatingCouponBond (which inherits from class Bond). It contains a list with the following components:

- **NPV**: net present value of the bond
- **cleanPrice**: price price of the bond
- **dirtyPrice**: dirty price of the bond
- **accruedAmount**: accrued amount of the bond
- **yield**: yield of the bond
- **cashFlows**: cash flows of the bond

The ConvertibleFixedCouponBond function returns an object of class ConvertibleFixedCouponBond (which inherits from class Bond). It contains a list with the following components:

- **NPV**: net present value of the bond
- **cleanPrice**: price price of the bond
- **dirtyPrice**: dirty price of the bond
- **accruedAmount**: accrued amount of the bond
- **yield**: yield of the bond
- **cashFlows**: cash flows of the bond

The ConvertibleZeroCouponBond function returns an object of class ConvertibleZeroCouponBond (which inherits from class Bond). It contains a list with the following components:

- **NPV**: net present value of the bond
- **cleanPrice**: price price of the bond
- **dirtyPrice**: dirty price of the bond
- **accruedAmount**: accrued amount of the bond
- **yield**: yield of the bond
- **cashFlows**: cash flows of the bond

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org/ for details on QuantLib.
Examples

```r
# this follow an example in test-suite/convertiblebond.cpp
params <- list(tradeDate=Sys.Date()-2,
               settleDate=Sys.Date(),
               dt=.25,
               interpWhat="discount",
               interpHow="loglinear")

dividendYield <- DiscountCurve(params, list(flat=0.02))
riskFreeRate <- DiscountCurve(params, list(flat=0.05))

dividendSchedule <- data.frame(Type=character(0), Amount=numeric(0),
                               Rate = numeric(0), Date = as.Date(character(0)))
callabilitySchedule <- data.frame(Price = numeric(0), Type=character(0),
                                  Date = as.Date(character(0)))

process <- list(underlying=50, divYield = dividendYield,
                 rff = riskFreeRate, volatility=0.15)

today <- Sys.Date()
bondparams <- list(exercise="am", faceAmount=100,
                    divSch = dividendSchedule,
                    callSch = callabilitySchedule,
                    redemption=100,
                    creditSpread=0.005,
                    conversionRatio = 0.0000000001,
                    issueDate=as.Date(today+2),
                    maturityDate=as.Date(today+3650))

dateparams <- list(settlementDays=3,
                    dayCounter="ActualActual",
                    period = "Semiannual", calendar = "UnitedStates/GovernmentBond",
                    businessDayConvention="Following")

lengths <- c(2,4,6,8,10,12,14,16,18,20,22,24,26,28,30)
coupons <- c( 0.0200, 0.0225, 0.0250, 0.0275, 0.0300, 0.0325, 0.0350, 0.0375, 0.0400, 0.0425, 0.0450, 0.0475, 0.0500, 0.0525, 0.0550 )
marketQuotes <- rep(100, length(lengths))
curvedateparams <- list(settlementDays=0, period="Annual",
                         dayCounter="ActualActual",
                         businessDayConvention="Unadjusted")
curveparams <- list(method="ExponentialSplinesFitting",
                     origDate = Sys.Date())
curve <- FittedBondCurve(curveparams, lengths, coupons, marketQuotes, curvedateparams)
iborindex <- list(type="USDLibor", length=6,
                   inTermQf="Month", term=curve)

spreads <- c()
#ConvertibleFloatingCouponBond(bondparams, iborindex, spreads, process, dateparams)

# example using default values
ConvertibleBond

#ConvertibleFloatingCouponBond(bondparams, iborindex, spreads, process)

dateparams <- list(settlementDays=3,
                  period = "Semiannual",
                  businessDayConvention="Unadjusted")

bondparams <- list(
                  creditSpread=0.005, conversionRatio = 0.000000001,
                  issueDate=as.Date(today+2),
                  maturityDate=as.Date(today+3650))

#ConvertibleFloatingCouponBond(bondparams, iborindex, spreads, process, dateparams)

#this follow an example in test-suite/convertiblebond.cpp
#for ConvertibleFixedCouponBond

#set up arguments to build a pricing engine.
params <- list(tradeDate=Sys.Date()-2,
               settleDate=Sys.Date(),
               dt=.25,
               interpWhat="discount",
               interpHow="loglinear")
times <- seq(0,10,.1)

dividendYield <- DiscountCurve(params, list(flat=0.02), times)
riskFreeRate <- DiscountCurve(params, list(flat=0.05), times)

dividendSchedule <- data.frame(Type=character(0), Amount=numeric(0),
                                Rate = numeric(0), Date = as.Date(character(0)))
callabilitySchedule <- data.frame(Price = numeric(0), Type=character(0),
                                     Date = as.Date(character(0)))

process <- list(underlying=50, divYield = dividendYield,
                 rff = riskFreeRate, volatility=0.15)

today <- Sys.Date()
bondparams <- list(exercise="am", faceAmount=100, divSch = dividendSchedule,
                   callSch = callabilitySchedule, redemption=100,
                   creditSpread=0.005, conversionRatio = 0.000000001,
                   issueDate=as.Date(today+2),
                   maturityDate=as.Date(today+3650))
dateparams <- list(settlementDays=3,
                   dayCounter="Actual360",
                   period = "Once", calendar = "UnitedStates/GovernmentBond",
                   businessDayConvention="Following")
coupon <- c(0.05)
ConvertibleFixedCouponBond(bondparams, coupon, process, dateparams)

#example with default value
ConvertibleFixedCouponBond(bondparams, coupon, process)
```r
dateparams <- list(settlementDays=3,
          dayCounter="Actual360")
ConvertibleFixedCouponBond(bondparams, coupon, process, dateparams)

bondparams <- list(creditSpread=0.005, conversionRatio = 0.000000001,
          issueDate=as.Date(today+2),
          maturityDate=as.Date(today+3650))
ConvertibleFixedCouponBond(bondparams, coupon, process, dateparams)

#this follow an example in test-suite/convertiblebond.cpp
params <- list(tradeDate=Sys.Date()-2,
          settleDate=Sys.Date(),
          dt=.25,
          interpWhat="discount",
          interpHow="loglinear")
times <- seq(0,10,.1)

dividendYield <- DiscountCurve(params, list(flat=0.02), times)
riskFreeRate <- DiscountCurve(params, list(flat=0.05), times)

dividendSchedule <- data.frame(Type=character(0), Amount=numeric(0),
          Rate = numeric(0), Date = as.Date(character(0)))
callabilitySchedule <- data.frame(Price = numeric(0), Type=character(0),
          Date = as.Date(character(0)))

process <- list(underlying=50, divYield = dividendYield,
          rff = riskFreeRate, volatility=0.15)

today <- Sys.Date()
bondparams <- list(exercise="am", faceAmount=100, divSch = dividendSchedule, 
          callSch = callabilitySchedule, redemption=100, 
          creditSpread=0.005, conversionRatio = 0.000000001, 
          issueDate=as.Date(today+2),
          maturityDate=as.Date(today+3650))
dateparams <- list(settlementDays=3,
          dayCounter="Actual360",
          period = "Once", calendar = "UnitedStates/GovernmentBond",
          businessDayConvention="Following"
)

ConvertibleZeroCouponBond(bondparams, process, dateparams)

#example with default values
ConvertibleZeroCouponBond(bondparams, process)

bondparams <- list(creditSpread=0.005,
          conversionRatio=0.000000001,
          issueDate=as.Date(today+2),
```
DiscountCurve

maturityDate=as.Date(today+3650))
dateparams <- list(settlementDays=3, dayCounter='Actual360')
ConvertibleZeroCouponBond(bondparams, process, dateparams)
ConvertibleZeroCouponBond(bondparams, process)

---

DiscountCurve | Returns the discount curve (with zero rates and forwards) given times

### Description

DiscountCurve constructs the spot term structure of interest rates based on input market data including the settlement date, deposit rates, futures prices, FRA rates, or swap rates, in various combinations. It returns the corresponding discount factors, zero rates, and forward rates for a vector of times that is specified as input.

### Usage

DiscountCurve(params, tsQuotes, times)

### Arguments

- **params**: A list specifying the tradeDate (month/day/year), settleDate, forward rate time span dt, and two curve construction options: interpWhat (with possible values discount, forward, and zero) and interpHow (with possible values linear, loglinear, and spline). spline here means cubic spline interpolation of the interpWhat value.
- **tsQuotes**: Market quotes used to construct the spot term structure of interest rates. Must be a list of name/value pairs, where the currently recognized names are:

  - flat: rate for a flat yield curve
  - d1w: 1-week deposit rate
  - d1m: 1-month deposit rate
  - d3m: 3-month deposit rate
  - d6m: 6-month deposit rate
  - d9m: 9-month deposit rate
  - d1y: 1-year deposit rate
  - s2y: 2-year swap rate
  - s3y: 3-year swap rate
  - s5y: 5-year swap rate
  - s10y: 10-year swap rate
  - s15y: 15-year swap rate
  - s20y: 20-year swap rate
  - s30y: 30-year swap rate
  - fut1–fut8: 3-month futures contracts
  - fra3x6: 3x6 FRA
  - fra6x9: 6x9 FRA
  - fra6x12: 6x12 FRA
Here rates are expected as fractions (so 5% means .05). If flat is specified it must be the first and only item in the list. The eight futures correspond to the first eight IMM dates. The maturity dates of the instruments specified need not be ordered, but they must be distinct.

**times**  
A vector of times at which to return the discount factors, forward rates, and zero rates. Times must be specified such that the largest time plus dt does not exceed the longest maturity of the instruments used for calibration (no extrapolation).

**Details**

This function is based on QuantLib Version 0.3.10. It introduces support for fixed-income instruments in RQuantLib.

Forward rates and zero rates are computed assuming continuous compounding, so the forward rate \( f \) over the period from \( t_1 \) to \( t_2 \) is determined by the relation

\[ d_1/d_2 = e^{f(t_2-t_1)}, \]

where \( d_1 \) and \( d_2 \) are discount factors corresponding to the two times. In the case of the zero rate \( t_1 \) is the current time (the spot date).

Curve construction can be a delicate problem and the algorithms may fail for some input data sets and/or some combinations of the values for interpwhat and interphow. Fortunately, the C++ exception mechanism seems to work well with the R interface, and QuantLib exceptions are propagated back to the R user, usually with a message that indicates what went wrong. (The first part of the message contains technical information about the precise location of the problem in the QuantLib code. Scroll to the end to find information that is meaningful to the R user.)

**Value**

DiscountCurve returns a list containing:

- **times**  
  Vector of input times

- **discounts**  
  Corresponding discount factors

- **forwards**  
  Corresponding forward rates with time span \( dt \)

- **zerorates**  
  Corresponding zero coupon rates

- **flatQuotes**  
  True if a flat quote was used, False otherwise

- **params**  
  The input parameter list

**Author(s)**

Dominick Samperi

**References**


For information about QuantLib see [http://quantlib.org](http://quantlib.org).

See Also

BermudanSwaption

Examples

```r
savepar <- par(mfrow=c(3,3), mar=c(4,4,2,0.5))

## This data is taken from sample code shipped with QuantLib 0.9.7
## from the file Examples/Swap/swapvaluation
params <- list(tradeDate=as.Date('2004-09-20'),
                settleDate=as.Date('2004-09-22'),
                dt=.25,
                interpWhat="discount",
                interpHow="loglinear")
setEvaluationDate(as.Date("2004-09-20"))

## We get numerical issue for the spline interpolation if we add
## any on of these three extra futures -- the original example
## creates different curves based on different deposit, fra, futures
## and swap data
## Removing s2y helps, as kindly pointed out by Luigi Ballabio

if (exists("s2y", environment(params))) {
  tsQuotes <- list(d1w = 0.0382,
                   d3m = 0.0363,
                   d6m = 0.0353,
                   d9m = 0.0348,
                   d1y = 0.0345,
                   fut1=96.2875,
                   fut2=96.7875,
                   fut3=96.9875,
                   fut4=96.6875,
                   fut5=96.4875,
                   fut6=96.3875,
                   fut7=96.2875,
                   fut8=96.0875,
                   #
                   s2y = 0.037125,
                   s3y = 0.0398,
                   s5y = 0.0443,
                   s10y = 0.05165,
                   s15y = 0.055175)
}

times <- seq(0,10,.1)

# Loglinear interpolation of discount factors
curves <- DiscountCurve(params, tsQuotes, times)
plot(curves, setpar=FALSE)

# Linear interpolation of discount factors
params$interpHow="linear"
curves <- DiscountCurve(params, tsQuotes, times)
plot(curves, setpar=FALSE)
```
# Spline interpolation of discount factors
params$interpHow="spline"
curves <- DiscountCurve(params, tsQuotes, times)
plot(curves,setpar=FALSE)
par(savepar)

Enum Documentation for parameters

Description
Reference for parameters when constructing a bond

Arguments

<table>
<thead>
<tr>
<th>DayCounter</th>
<th>an int value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Actual360</td>
</tr>
<tr>
<td>1</td>
<td>Actual360FixEd</td>
</tr>
<tr>
<td>2</td>
<td>ActualActual</td>
</tr>
<tr>
<td>3</td>
<td>ActualBusiness252</td>
</tr>
<tr>
<td>4</td>
<td>OneDayCounter</td>
</tr>
<tr>
<td>5</td>
<td>SimpleDayCounter</td>
</tr>
<tr>
<td>6</td>
<td>Thirty360</td>
</tr>
<tr>
<td>7</td>
<td>Actual365NoLeap</td>
</tr>
<tr>
<td>8</td>
<td>ActualActual.ISMA</td>
</tr>
<tr>
<td>9</td>
<td>ActualActual.Bond</td>
</tr>
<tr>
<td>10</td>
<td>ActualActual.ISDA</td>
</tr>
<tr>
<td>11</td>
<td>ActualActual.Historical</td>
</tr>
<tr>
<td>12</td>
<td>ActualActual.AFB</td>
</tr>
<tr>
<td>anything else</td>
<td>ActualActual.Euro</td>
</tr>
</tbody>
</table>

businessDayConvention

<table>
<thead>
<tr>
<th>an int value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>anything else</td>
</tr>
</tbody>
</table>

compounding

<table>
<thead>
<tr>
<th>an int value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Enum

2  Continuous
3  SimpleThenCompounded

period or frequency
    an int value

-1  NoFrequency
0   Once
1   Annual
2   Semiannual
3   EveryFourthMonth
4   Quarterly
6   BiMonthly
12  Monthly
13  EveryFourthWeek
26  BiWeekly
52  Weekly
365 Daily
anything else  OtherFrequency

date generation
    an int value to specify date generation rule

0   Backward
1   Forward
2   Zero
3   ThirdWednesday
4   Twentieth
anything else  TwentiethIMM

durationType    an int value to specify duration type

0   Simple
1   Macaulay
2   Modified

Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation, particularly the datetime classes.

Value

None

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu>
EuropeanOption

References

http://quantlib.org for details on QuantLib.

EuropeanOption European Option evaluation using Closed-Form solution

Description

The EuropeanOption function evaluates an European-style option on a common stock using the Black-Scholes-Merton solution. The option value, the common first derivatives ("Greeks") as well as the calling parameters are returned.

Usage

```r
## Default S3 method:
EuropeanOption(type, underlying, strike,
      dividendYield, riskFreeRate, maturity, volatility)
```

Arguments

- `type`: A string with one of the values call or put
- `underlying`: Current price of the underlying stock
- `strike`: Strike price of the option
- `dividendYield`: Continuous dividend yield (as a fraction) of the stock
- `riskFreeRate`: Risk-free rate
- `maturity`: Time to maturity (in fractional years)
- `volatility`: Volatility of the underlying stock

Details

The well-known closed-form solution derived by Black, Scholes and Merton is used for valuation. Implied volatilities are calculated numerically.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The EuropeanOption function returns an object of class EuropeanOption (which inherits from class Option). It contains a list with the following components:

- `value`: Value of option
- `delta`: Sensitivity of the option value for a change in the underlying
- `gamma`: Sensitivity of the option delta for a change in the underlying
- `vega`: Sensitivity of the option value for a change in the underlying’s volatility
- `theta`: Sensitivity of the option value for a change in t, the remaining time to maturity
- `rho`: Sensitivity of the option value for a change in the risk-free interest rate
- `dividendRho`: Sensitivity of the option value for a change in the dividend yield
Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

EuropeanOptionImpliedVolatility, EuropeanOptionArrays, AmericanOption, BinaryOption

Examples

# simple call with unnamed parameters
EuropeanOption("call", 100, 100, 0.01, 0.03, 0.5, 0.4)
# simple call with some explicit parameters, and slightly increased vol:
EuropeanOption(type="call", underlying=100, strike=100, dividendYield=0.01, riskFreeRate=0.03, maturity=0.5, volatility=0.5)

EuropeanOptionArrays  European Option evaluation using Closed-Form solution

Description

The EuropeanOptionArrays function allows any two of the numerical input parameters to be a vector, and a list of matrices is returned for the option value as well as each of the 'greeks'. For each of the returned matrices, each element corresponds to an evaluation under the given set of parameters.

Usage

EuropeanOptionArrays(type, underlying, strike, dividendYield, riskFreeRate, maturity, volatility)
oldEuropeanOptionArrays(type, underlying, strike, dividendYield, riskFreeRate, maturity, volatility)
plotOptionSurface(E0res, ylable="", xlabell="", zlabell="", fov=60)

Arguments

type  A string with one of the values call or put
underlying  (Scalar or list) current price(s) of the underlying stock
strike  (Scalar or list) strike price(s) of the option
dividendYield  (Scalar or list) continuous dividend yield(s) (as a fraction) of the stock
EuropeanOptionArrays

riskFreeRate (Scalar or list) risk-free rate(s)
maturity (Scalar or list) time(s) to maturity (in fractional years)
volatility (Scalar or list) volatility(ies) of the underlying stock
E0res result matrix produced by EuropeanOptionArrays
ylabel label for y-axis
xlabel label for x-axis
zlabel label for z-axis
fov viewpoint for 3d rendering

Details
The well-known closed-form solution derived by Black, Scholes and Merton is used for valuation. Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value
The EuropeanOptionArrays function allows any two of the numerical input parameters to be a vector or sequence. A list of two-dimensional matrices is returned. Each cell corresponds to an evaluation under the given set of parameters.
For these functions, the following components are returned:
value (matrix) value of option
delta (matrix) change in value for a change in the underlying
gamma (matrix) change in value for a change in delta
vega (matrix) change in value for a change in the underlying’s volatility
theta (matrix) change in value for a change in delta
rho (matrix) change in value for a change in time to maturity
dividendRho (matrix) change in value for a change in delta
parameters List with parameters with which object was created

The oldEuropeanOptionArrays function is an older implementation which vectorises this at the R level instead but allows more general multidimensional arrays.

Note
The interface might change in future release as QuantLib stabilises its own API.

Author(s)
Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References
http://quantlib.org for details on QuantLib.
EuropeanOptionImpliedVolatility

Implied Volatility calculation for European Option

Description

The EuropeanOptionImpliedVolatility function solves for the (unobservable) implied volatility, given an option price as well as the other required parameters to value an option.
Usage

```r
## Default S3 method:
EuropeanOptionImpliedVolatility(type, value, underlying, strike, dividendYield, riskFreeRate, maturity, volatility)
```

Arguments

- **type**: A string with one of the values call or put
- **value**: Value of the option (used only for ImpliedVolatility calculation)
- **underlying**: Current price of the underlying stock
- **strike**: Strike price of the option
- **dividendYield**: Continuous dividend yield (as a fraction) of the stock
- **riskFreeRate**: Risk-free rate
- **maturity**: Time to maturity (in fractional years)
- **volatility**: Initial guess for the volatility of the underlying stock

Details

The well-known closed-form solution derived by Black, Scholes and Merton is used for valuation. Implied volatilities are then calculated numerically.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The `EuropeanOptionImpliedVolatility` function returns an numeric variable with volatility implied by the given market prices and given parameters.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

[http://quantlib.org](http://quantlib.org) for details on QuantLib.

See Also

- `EuropeanOption`
- `AmericanOption`
- `BinaryOption`
**FittedBondCurve**

**Examples**

```python
europeanOptionImpliedVolatility(type="call", value=11.10, underlying=100, strike=100, dividendYield=0.01, riskFreeRate=0.03, maturity=0.5, volatility=0.4)
```

---

**FittedBondCurve**

Returns the discount curve (with zero rates and forwards) given set of bonds

---

**Description**

FittedBondCurve fits a term structure to a set of bonds using three different fitting methodologies. For more detail, see QuantLib/Example/FittedBondCurve.

**Usage**

FittedBondCurve(curveparams, lengths, coupons, marketQuotes, dateparams)

**Arguments**

- **curveparams** curve parameters
  - **method** a string, fitting methods: "ExponentialSplinesFitting", "SimplePolynomialFitting", "NelsonSiegelFitting"
  - **origDate** a Date, starting date of the curve
- **lengths** an numeric vector, length of the bonds in year
- **coupons** a numeric vector, coupon rate of the bonds
- **marketQuotes** a numeric vector, market price of the bonds
- **dateparams** (Optional) a named list, QuantLib’s date parameters of the bond.
  - **settlementDays** (Optional) a double, settlement days. Default value is 1.
  - **dayCounter** (Optional) a number or string, day counter convention. See Enum. Default value is ‘Thirty360’
  - **period** (Optional) a number or string, interest compounding interval. See Enum. Default value is ‘Semiannual’
  - **businessDayConvention** (Optional) a number or string, business day convention. See Enum. Default value is ‘Following’.

See example below.
Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

table, a three columns "date - zeroRate - discount" data frame

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org/ for details on QuantLib.

Examples

```r
lengths <- c(2,4,6,8,10,12,14,16,18,20,22,24,26,28,30)
coupons <- c( 0.0200, 0.0225, 0.0250, 0.0275, 0.0300, 0.0325, 0.0350, 0.0375, 0.0400, 0.0425, 0.0450, 0.0475, 0.0500, 0.0525, 0.0550 )
marketQuotes <- rep(100, length(lengths))
dateparams <- list(settlementDays=0, period="Annual",
dayCounter="ActualActual",
businessDayConvention ="Unadjusted")
curveparams <- list(method="ExponentialSplinesFitting",
origDate = Sys.Date())
curve <- FittedBondCurve(curveparams, lengths, coupons, marketQuotes, dateparams)
if (require(zoo)) {
  z <- zoo(curve$table$zeroRates, order.by=curve$table$date)
  plot(z)
}
```

Description

The `FixedRateBond` function evaluates a fixed rate bond using discount curve, the yield or the clean price. More specifically, when a discount curve is provided the calculation is done by DiscountingBondEngine from QuantLib. The NPV, clean price, dirty price, accrued interest, yield, duration, actual settlement date and cash flows of the bond is returned. When a yield is provided instead, no engine is provided to the bond class and prices are computed from yield. In the latter case, NPV is set to NA. Same situation when the clean price is given instead of discount curve or yield. For more detail, see the source codes in QuantLib's file test-suite/bond.cpp.
The `FixedRateBondPriceByYield` function calculates the theoretical price of a fixed rate bond from its yield.

The `FixedRateBondYield` function calculates the theoretical yield of a fixed rate bond from its price.

**Usage**

```
## Default S3 method:
FixedRateBond(bond, rates, schedule,
              calc=list(dayCounter='ActualActual.ISMA',
                        compounding='Compounded',
                        freq='Annual',
                        durationType='Modified'),
              discountCurve = NULL, yield = NA, price = NA)

## Default S3 method:
FixedRateBondPriceByYield(settlementDays=1, yield, faceAmount=100,
                           effectiveDate, maturityDate,
                           period, calendar="UnitedStates/GovernmentBond",
                           rates, dayCounter=2,
                           businessDayConvention=0, compound = 0, redemption=100,
                           issueDate)

## Default S3 method:
FixedRateBondYield(settlementDays=1, price, faceAmount=100,
                   effectiveDate, maturityDate,
                   period, calendar="UnitedStates/GovernmentBond",
                   rates, dayCounter=2,
                   businessDayConvention=0,
                   compound = 0, redemption=100,
                   issueDate)
```

**Arguments**

<table>
<thead>
<tr>
<th>Bond</th>
<th>(Optional) bond parameters, a named list whose elements are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>settlementDays</td>
<td>(Optional) a double, settlement days. Default value is 1.</td>
</tr>
<tr>
<td>faceAmount</td>
<td>(Optional) a double, face amount of the bond. Default value is 100.</td>
</tr>
<tr>
<td>dayCounter</td>
<td>(Optional) a number or string, day counter convention. Defaults to 'Thirty360'</td>
</tr>
<tr>
<td>issueDate</td>
<td>(Optional) a Date, the bond’s issue date Defaults to QuantLib default.</td>
</tr>
<tr>
<td>paymentConvention</td>
<td>(Optional) a number or string, the bond payment convention. Defaults to QuantLib default.</td>
</tr>
<tr>
<td>redemption</td>
<td>(Optional) a double, the redemption amount.</td>
</tr>
</tbody>
</table>
FixedRateBond

- paymentCalendar: (Optional) a string, the name of the calendar. Defaults to QuantLib default.
- exCouponPeriod: (Optional) a number, the number of days when the coupon goes ex relative to the coupon date. Defaults to QuantLib default.
- exCouponCalendar: (Optional) a string, the name of the ex-coupon calendar. Defaults to QuantLib default.
- exCouponConvention: (Optional) a number or string, the coupon payment convention. Defaults to QuantLib default.
- exCouponEndOfMonth: (Optional) 1 or 0, use End of Month rule for ex-coupon dates. Defaults to 0 (false).

- rates: a numeric vector, bond’s coupon rates
- schedule: (Optional) a named list, QuantLib’s parameters of the bond’s schedule.
  - effectiveDate: a Date, when the schedule becomes effective.
  - maturityDate: a Date, when the schedule matures.
  - period: (Optional) a number or string, the frequency of the schedule. Default value is ‘Semiannual’.
  - calendar: (Optional) a string, the calendar name. Defaults to ‘TARGET’.
  - businessDayConvention: (Optional) a number or string, the day convention to use. Defaults to ‘Following’.
  - terminationDateConvention: (Optional) a number or string, the day convention to use for the terminal date. Defaults to ‘Following’.
  - dateGeneration: (Optional) a number or string, the date generation rule. Defaults to ‘Backward’.
  - endOfMonth: (Optional) 1 or 0, use End of Month rule for schedule dates. Defaults to 0 (false).

- calc: (Optional) a named list, QuantLib’s parameters for calculations.
  - dayCounter: (Optional) a number or string, day counter convention. Defaults to ‘ActualActual.ISMA’.
  - compounding: a string, what kind of compounding to use. Defaults to ‘Compounded’.
  - freq: (Optional) a number or string, the frequency to use. Default value is ‘Annual’.
  - durationType: (Optional) a number or string, the type of duration to calculate. Defaults to ‘Simple’.
  - accuracy: (Optional) a number, the accuracy required.

See example below.
FixedRateBond

maxEvaluations Defaults to 1.0e-8.
(Optional) a number, max number of iterations.
 Defaults to 100.

discoutCurve Can be one of the following:

a DiscountCurve a object of DiscountCurve class
For more detail, see example or the discountCurve function
A 2 items list specifies a flat curve in two
values "todayDate" and "rate"
A 3 items list specifies three values to construct a
DiscountCurve object, "params", "tsQuotes", "times".
For more detail, see example or the discountCurve function

yield yield of the bond
price clean price of the bond
settlementDays an integer, 1 for T+1, 2 for T+2, etc...
effectiveDate bond’s effective date
maturityDate bond’s maturity date
period frequency of events, 0=NoFrequency, 1=Once, 2=Annual, 3=Semiannual, 4=EveryFourthMonth, 5=Quarterly, 6=Bimonthly, 7=Monthly, 8=EveryFourthWeekly, 9=Biweekly, 10=Weekly, 11=Daily. For more information, see QuantLib’s Frequency class
calendar Business Calendar. Either us or uk
faceAmount face amount of the bond
businessDayConvention convention used to adjust a date in case it is not a valid business day. See quantlib for more detail. 0 = Following, 1 = ModifiedFollowing, 2 = Preceding, 3 = ModifiedPreceding, other = Unadjusted
dayCounter day count convention. 0 = Actual360(), 1 = Actual365Fixed(), 2 = ActualAct-
tual(), 3 = Business252(), 4 = OneDayCounter(), 5 = SimpleDayCounter(), all other = Thirty360(). For more information, see QuantLib’s DayCounter class
compound compounding type. 0=Simple, 1=Compounded, 2=Continuous, all other=SimpleThenCompounded. See QuantLib’s Compound class
redemption redemption when the bond expires
issueDate date the bond is issued

Details
A discount curve is built to calculate the bond value.
Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.
FixedRateBond

Value

The FixedRateBond function returns an object of class FixedRateBond (which inherits from class Bond). It contains a list with the following components:

- **NPV**: net present value of the bond
- **cleanPrice**: clean price of the bond
- **dirtyPrice**: dirty price of the bond
- **accruedAmount**: accrued amount of the bond
- **yield**: yield of the bond
- **duration**: the duration of the bond
- **settlementDate**: the actual settlement date used for the bond
- **cashFlows**: cash flows of the bond

The FixedRateBond::PriceByYield function returns an object of class FixedRateBond::PriceByYield (which inherits from class Bond). It contains a list with the following components:

- **price**: price of the bond

The FixedRateBond::Yield function returns an object of class FixedRateBond::Yield (which inherits from class Bond). It contains a list with the following components:

- **yield**: yield of the bond

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

[http://quantlib.org](http://quantlib.org) for details on QuantLib.

Examples

```r
# Simple call with a flat curve
bond <- list(settlementDays=1,
              issueDate=as.Date("2004-11-30"),
              faceAmount=100,
              accrualDayCounter='Thirty360',
              paymentConvention='Unadjusted')
schedule <- list(effectiveDate=as.Date("2004-11-30"),
                 maturityDate=as.Date("2008-11-30"),
                 period='Semiannual',
                 calendar='UnitedStates/GovernmentBond',
                 businessDayConvention='Unadjusted',
```
Csame bond with a discount curve constructed from market quotes
tsQuotes <- list(d1w = 0.0382,
  d1m = 0.0372,
  fut1 = 96.2875,
  fut2 = 96.7875,
  fut3 = 96.9875,
  fut4 = 96.6875,
  fut5 = 96.4875,
  fut6 = 96.3875,
  fut7 = 96.2875,
  fut8 = 96.0875,
  s3y = 0.0398,
  s5y = 0.0443,
  s10y = 0.05165,
  s15y = 0.055175)
tsQuotes <- list("flat" = 0.02) # While discount curve code is buggy
discountCurve <- DiscountCurve(params, tsQuotes)
FixedRateBond(bond,
  coupon.rate,
  schedule,
  calc,
  discountCurve=discountCurve)

#Same bond calculated from yield rather than from the discount curve
yield <- 0.02
FixedRateBond(bond,
  coupon.rate,
  schedule,
FloatingRateBond

Description

The `FloatingRateBond` function evaluates a floating rate bond using discount curve. More specifically, the calculation is done by `DiscountingBondEngine` from QuantLib. The NPV, clean price, dirty price, accrued interest, yield and cash flows of the bond is returned. For more detail, see the source codes in quantlib's test-suite: `test-suite/bond.cpp`
FloatingRateBond

Usage

## Default S3 method:
FloatingRateBond(bond, gearings, spreads, 
caps, floors, index, 
curve, dateparams )

Arguments

bond      bond parameters, a named list whose elements are:

  issueDate  a Date, the bond’s issue date
  maturityDate  a Date, the bond’s maturity date
  faceAmount  (Optional) a double, face amount of the bond.
              Default value is 100.
  redemption  (Optional) a double, percentage of the initial
              face amount that will be returned at maturity
date. Default value is 100.
  effectiveDate  (Optional) a Date, the bond’s effective date. Default value is issueDate

gearings  (Optional) a numeric vector, bond’s gearings. See quantlib’s doc on FloatingRateBond for more detail. Default value is an empty vector c().

spreads  (Optional) a numeric vector, bond’s spreads. See quantlib’s doc on FloatingRateBond for more detail. Default value is an empty vector c()

caps  (Optional) a numeric vector, bond’s caps. See quantlib’s doc on FloatingRateBond for more detail. Default value is an empty vector c()

difloors  (Optional) a numeric vector, bond’s floors. See quantlib’s doc on FloatingRateBond for more detail. Default value is an empty vector c()

curve  Can be one of the following:

  a DiscountCurve  a object of DiscountCurve class
                  For more detail, see example or
                  the discountCurve function
  A 2 items list  specifies a flat curve in two
                  values "todayDate" and "rate"
  A 3 items list  specifies three values to construct a
                  DiscountCurve object, "params", 
                  "tsQuotes", "times".
                  For more detail, see example or
                  the discountCurve function

index  a named list whose elements are parameters of an IborIndex term structure.

  type  a string, currently support only "USDLibor"
  length  an integer, length of the index
  inTermOf  a string, period unit, currently support only 'Month'
**FloatingRateBond**

**term**  
a DiscountCurve object, the term structure of the index

**dateparams**  
(Optional) a named list, QuantLib’s date parameters of the bond.

**settlementDays**  
(Optional) a double, settlement days.  
Default value is 1.

**calendar**  
(Optional) a string, either ‘us’ or ‘uk’ 
corresponding to US Government Bond 
calendar and UK Exchange calendar.  
Default value is ‘us’.

**dayCounter**  
(Optional) a number or string, 
day counter convention.  
See Enum. Default value is ‘Thirty360’

**period**  
(Optional) a number or string, 
interest compounding interval. See Enum.  
Default value is ‘Semiannual’.

**businessDayConvention**  
(Optional) a number or string, 
business day convention.  
See Enum. Default value is ‘Following’.

**terminationDateConvention**  
(Optional) a number or string,  
termination day convention.  
See Enum. Default value is ‘Following’.

**endOfMonth**  
(Optional) a numeric with value 1 or 0.  
End of Month rule. Default value is 0.

**dateGeneration**  
(Optional) a numeric, date generation method.  
See Enum. Default value is ‘Backward’

See example below.

**Details**

A discount curve is built to calculate the bond value.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

**Value**

The FloatingRateBond function returns an object of class FloatingRateBond (which inherits from class Bond). It contains a list with the following components:

- **NPV** net present value of the bond
- **cleanPrice** clean price of the bond
- **dirtyPrice** dirty price of the bond
- **accruedAmount** accrued amount of the bond
- **yield** yield of the bond
- **cashFlows** cash flows of the bond
Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Khanh Nguyen <knguyen@cs.umbn.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

Examples

```r
bond <- list(faceAmount=100, issueDate=as.Date("2004-11-30"),
maturityDate=as.Date("2008-11-30"), redemption=100,
effectiveDate=as.Date("2004-11-30"))
dateparams <- list(settlementDays=1, calendar="UnitedStates/GovernmentBond",
                   dayCounter = 'ActualActual', period=2,
businessDayConvention = 1, terminationDateConvention=1,
dateGeneration=0, endOfMonth=0, fixingDays = 1)
gearings <- spreads <- caps <- floors <- vector()

params <- list(tradeDate=as.Date('2002-2-15'),
                settleDate=as.Date('2002-2-19'),
dt=.25,
                interpWhat="discount",
                interpHow="loglinear")
setEvaluationDate(as.Date("2004-11-22"))

tsQuotes <- list(d1w = 0.0382,
                 d1m = 0.0372,
                 fut1=96.2875,
                 fut2=96.7875,
                 fut3=96.9875,
                 fut4=96.6875,
                 fut5=96.4875,
                 fut6=96.3875,
                 fut7=96.2875,
                 fut8=96.0875,
                 s3y = 0.0398,
                 s5y = 0.0443,
                 s10y = 0.05165,
                 s15y = 0.055175)
tsQuotes <- list("flat" = 0.02) ## While discount curve code is buggy

## when both discount and libor curves are flat.

discountCurve.flat <- DiscountCurve(params, list(flat=0.05))
termstructure <- DiscountCurve(params, list(flat=0.03))
```
ImpliedVolatility

**ImpliedVolatility**  
*Base class for option-price implied volatility evaluation*

**Description**

This class forms the basis from which the more specific classes are derived.

**Usage**

```r
## S3 method for class 'ImpliedVolatility'
print(x, digits=3, ...)
## S3 method for class 'ImpliedVolatility'
summary(object, digits=3, ...)
```

**Arguments**

- `x`  
  Any option-price implied volatility object derived from this base class

- `object`  
  Any option-price implied volatility object derived from this base class

- `digits`  
  Number of digits of precision shown

- `...`  
  Further arguments

**Details**

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

**Value**

None, but side effects of displaying content.
**Option**

**Note**

The interface might change in future release as QuantLib stabilises its own API.

**Author(s)**

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

**References**

[http://quantlib.org](http://quantlib.org) for details on QuantLib.

**See Also**

AmericanOptionImpliedVolatility, EuropeanOptionImpliedVolatility, AmericanOption, EuropeanOption, BinaryOption

**Examples**

```r
impVol <- EuropeanOptionImpliedVolatility("call", value=11.10, strike=100, 
volatility=0.4, 100, 0.01, 0.03, 0.5)
print(impVol)
summary(impVol)
```

---

**Description**

This class forms the basis from which the more specific classes are derived.

**Usage**

```r
## S3 method for class 'Option'
print(x, digits=4, ...)  
## S3 method for class 'Option'
plot(x, ...)  
## S3 method for class 'Option'
summary(object, digits=4, ...)
```

**Arguments**

- `x` Any option object derived from this base class
- `object` Any option object derived from this base class
- `digits` Number of digits of precision shown
- `...` Further arguments
Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

None, but side effects of displaying content.

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.

See Also

AmericanOption, EuropeanOption, BinaryOption

Examples

EO<−EuropeanOption("call", strike=100, volatility=0.4, 100, 0.01, 0.03, 0.5)
print(EO)
summary(EO)

Schedule generation

Description

The Schedule function generates a schedule of dates conformant to a given convention in a given calendar.

Usage

## Default S3 method:
Schedule(params)
Arguments

params a named list, QuantLib’s parameters of the schedule.

**effectiveDate** a Date, when the schedule becomes effective.

**maturityDate** a Date, when the schedule matures.

**period** (Optional) a number or string, the frequency of the schedule. Default value is ‘Semiannual’.

**calendar** (Optional) a string, the calendar name. Defaults to ‘TARGET’

**businessDayConvention** (Optional) a number or string, the day convention to use. Defaults to ‘Following’.

**terminationDateConvention** (Optional) a number or string, the day convention to use for the terminal date. Defaults to ‘Following’.

**dateGeneration** (Optional) a number or string, the date generation rule. Defaults to ‘Backward’.

**endOfMonth** (Optional) 1 or 0, use End of Month rule for schedule dates. Defaults to 0 (false).

See example below.

Details

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The Schedule function returns an object of class Schedule. It contains the list of dates in the schedule.

Author(s)

Michele Salvadore <michele.salvadore@gmail.com> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

[http://quantlib.org](http://quantlib.org) for details on QuantLib.

See Also

[FixedRateBond](#)
Examples

```r
params <- list(
  effectiveDate = as.Date("2004-11-30"),
  maturityDate = as.Date("2008-11-30"),
  period = 'Semiannual',
  calendar = 'UnitedStates/GovernmentBond',
  businessDayConvention = 'Unadjusted',
  terminationDateConvention = 'Unadjusted',
  dateGeneration = 'Forward',
  endOfMonth = 1)
Schedule(params)
```

### Description

The `ZeroCouponBond` function evaluates a zero-coupon plainly using discount curve. More specifically, the calculation is done by `DiscountingBondEngine` from QuantLib. The NPV, clean price, dirty price, accrued interest, yield and cash flows of the bond is returned. For more detail, see the source code in the QuantLib file `test-suite/bond.cpp`.

The `ZeroPriceYield` function evaluates a zero-coupon clean price based on its yield.


### Usage

#### Default S3 method:

```r
ZeroCouponBond(bond, discountCurve, dateparams)
```

#### Default S3 method:

```r
ZeroPriceByYield(yield, faceAmount,
  issueDate, maturityDate,
  dayCounter = 2, frequency = 2,
  compound = 0, businessDayConvention = 4)
```

#### Default S3 method:

```r
ZeroYield(price, faceAmount,
  issueDate, maturityDate,
  dayCounter = 2, frequency = 2,
  compound = 0, businessDayConvention = 4)
```

### Arguments

- **bond**: bond parameters, a named list whose elements are:
  - **issueDate**: a Date, the bond’s issue date
  - **maturityDate**: a Date, the bond’s maturity date
ZeroCouponBond

faceAmount  (Optional) a double, face amount of the bond.
            Default value is 100.

redemption  (Optional) a double, percentage of the initial
            face amount that will be returned at maturity
            date. Default value is 100.

discountCurve  Can be one of the following:

   a DiscountCurve       a object of DiscountCurve class
            For more detail, see example or
            the discountCurve function

   A 2 items list       specifies a flat curve in two
            values "todayDate" and "rate"

   A 3 items list       specifies three values to construct a
            DiscountCurve object, "params",  
            "tsQuotes", "times".
            For more detail, see example or
            the discountCurve function

dateparams  (Optional) a named list, QuantLib's date parameters of the bond.

settlementDays  (Optional) a double, settlement days.
            Default value is 1.

calendar  (Optional) a string, either 'us' or 'uk'
            corresponding to US Government Bond
            calendar and UK Exchange calendar.
            Default value is 'us'.

businessDayConvention  (Optional) a number or string,
            business day convention.
            See Enum. Default value is 'Following'.

yield  yield of the bond

price  price of the bond

faceAmount  face amount of the bond

issueDate  date the bond is issued

maturityDate  maturity date, an R's date type

dayCounter  day count convention. 0 = Actual360(), 1 = Actual365Fixed(), 2 = ActualActual(), 3 = Business252(), 4 = OneDayCounter(), 5 = SimpleDayCounter(), all
            other = Thirty360(). For more information, see QuantLib’s DayCounter class

frequency  frequency of events, 0 = NoFrequency, 1 = Once, 2 = Annual, 3 = Semiannual, 4 = EveryFourthMonth, 5 = Quarterly, 6 = Bimonthly, 7 = Monthly, 8 = EveryFourthWeekly, 9 = Biweekly, 10 = Weekly, 11 = Daily. For more information, see QuantLib’s Frequency class
ZeroCouponBond

compound  compounding type. 0=Simple, 1=Compounded, 2=Continuous, all other=SimpleThenCompounded. See QuantLib’s Compound class

businessDayConvention  convention used to adjust a date in case it is not a valid business day. See quantlib for more detail. 0 = Following, 1 = ModifiedFollowing, 2 = Preceding, 3 = ModifiedPreceding, other = Unadjusted

Details

A discount curve is built to calculate the bond value.

Please see any decent Finance textbook for background reading, and the QuantLib documentation for details on the QuantLib implementation.

Value

The ZeroCouponBond function returns an object of class ZeroCouponBond (which inherits from class Bond). It contains a list with the following components:

NPV  net present value of the bond
cleanPrice  clean price of the bond
dirtyPrice  dirty price of the bond
accruedAmount  accrued amount of the bond
yield  yield of the bond
cashFlows  cash flows of the bond

The ZeroPriceByYield function returns an object of class ZeroPriceByYield (which inherits from class Bond). It contains a list with the following components:

price  price of the bond

The ZeroYield function returns an object of class ZeroYield (which inherits from class Bond). It contains a list with the following components:

yield  yield of the bond

Note

The interface might change in future release as QuantLib stabilises its own API.

Author(s)

Khanh Nguyen <knguyen@cs.umb.edu> for the implementation; Dirk Eddelbuettel <edd@debian.org> for the R interface; the QuantLib Group for QuantLib

References

http://quantlib.org for details on QuantLib.
Examples

# Simple call with all parameter and a flat curve
bond <- list(faceAmount=100, issueDate=as.Date("2004-11-30"),
maturityDate=as.Date("2008-11-30"), redemption=100)

dateparams <- list(settlementDays=1, calendar="UnitedStates/GovernmentBond",
businessDayConvention='Unadjusted')

discCurve.param <- list(tradeDate=as.Date('2002-2-15'),
settleDate=as.Date('2002-2-15'),
dt=0.25,
interpWhat='discount', interpHow='loglinear')
discCurve.flat <- DiscountCurve(discCurve.param, list(flat=0.05))
ZeroCouponBond(bond, discCurve.flat, dateparams)

# The same bond with a discount curve constructed from market quotes
tsQuotes <- list(d1w =0.0382,
d1m =0.0372,
fut1=96.2875,
fut2=96.7875,
fut3=96.9875,
fut4=96.6875,
fut5=96.4875,
fut6=96.3875,
fut7=96.2875,
fut8=96.0875,
s3y =0.0398,
s5y =0.0443,
s10y =0.05165,
s15y =0.055175)
tsQuotes <- list("flat" = 0.02) # While discount curve code is buggy
discCurve <- flat = 0.02)
discCurve.param, tsQuotes)
ZeroCouponBond(bond, discCurve, dateparams)

# examples with default arguments
ZeroCouponBond(bond, discCurve)

bond <- list(issueDate=as.Date("2004-11-30"),
maturityDate=as.Date("2008-11-30"))
dateparams <- list(settlementDays=1)
ZeroCouponBond(bond, discCurve, dateparams)

ZeroPriceByYield(0.1478, 100, as.Date("1993-6-24"), as.Date("1993-11-1"))

ZeroYield(00, 100, as.Date("1993-6-24"), as.Date("1993-11-1"))
Index

*Topic misc
  AmericanOption, 2
  AmericanOptionImpliedVolatility, 4
  AsianOption, 5
  BarrierOption, 7
  BinaryOption, 11
  BinaryOptionImpliedVolatility, 13
  Bond, 14
  BondUtilities, 17
  Calendars, 18
  CallableBond, 22
  Enum, 34
  EuropeanOption, 36
  EuropeanOptionArrays, 37
  EuropeanOptionImpliedVolatility, 39
  FixedRateBond, 42
  FloatingRateBond, 48
  ImpliedVolatility, 52
  Option, 53
  Schedule, 54
  ZeroCouponBond, 56

*Topic models
  BermudanSwaption, 9
  DiscountCurve, 31
  adjust (Calendars), 18
  advance (Calendars), 18
  advanceDate (Calendars), 18
  AmericanOption, 2, 5, 8, 13, 14, 37, 39, 40, 53, 54
  AmericanOptionImpliedVolatility, 4, 53
  AsianOption, 5
  BarrierOption, 7
  BermudanSwaption, 9, 33
  BinaryOption, 5, 11, 14, 37, 39, 40, 53, 54
  BinaryOptionImpliedVolatility, 13
  Bond, 14
  BondUtilities, 17
  businessDay (Calendars), 18
  businessDaysBetween (Calendars), 18
  Calendars, 18
  CallableBond, 22
  ConvertibleBond, 25
  ConvertibleFixedCouponBond
    (ConvertibleBond), 25
  ConvertibleFloatingCouponBond
    (ConvertibleBond), 25
  ConvertibleZeroCouponBond
    (ConvertibleBond), 25
dayCount (Calendars), 18
  DiscountCurve, 9, 10, 31
  endOfMonth (Calendars), 18
  Enum, 19, 20, 23, 26, 34, 41, 50, 57
  EuropeanOption, 3, 5, 8, 13, 14, 36, 40, 53, 54
  EuropeanOptionArrays, 37, 37
  EuropeanOptionImpliedVolatility, 37, 39, 53
  FittedBondCurve, 41
  FixedRateBond, 42, 55
  FixedRateBondPriceByYield
    (FixedRateBond), 42
  FixedRateBondYield (FixedRateBond), 42
  FloatingRateBond, 48
  getEndOfMonth (Calendars), 18
  getHolidayList (Calendars), 18
  holidayList (Calendars), 18
  ImpliedVolatility, 52
  isBusinessDay (Calendars), 18
  isEndOfMonth (Calendars), 18
  isHoliday (Calendars), 18
  isWeekend (Calendars), 18
matchBDC (BondUtilities), 17
matchCompounding (BondUtilities), 17
matchDateGen (BondUtilities), 17
matchDayCounter (BondUtilities), 17
matchFrequency (BondUtilities), 17
matchParams (BondUtilities), 17

oldEuropeanOptionArrays
  (EuropeanOptionArrays), 37
  Option, 3, 6, 8, 12, 36, 53

plot.Bond (Bond), 14
plot.DiscountCurve (DiscountCurve), 31
plot.FittedBondCurve (FittedBondCurve), 41
plot.Option (Option), 53
plotOptionSurface
  (EuropeanOptionArrays), 37
print.Bond (Bond), 14
print.FixedRateBond (Bond), 14
print.ImpliedVolatility
  (ImpliedVolatility), 52
print.Option (Option), 53

Schedule, 54
setCalendarContext (Calendars), 18
setEvaluationDate (Calendars), 18
summary.BKTree (BermudanSwaption), 9
summary.Bond (Bond), 14
summary.G2Analytic (BermudanSwaption), 9
summary.HWAnalytic (BermudanSwaption), 9
summary.HWTree (BermudanSwaption), 9
summary.ImpliedVolatility
  (ImpliedVolatility), 52
summary.Option (Option), 53

yearFraction (Calendars), 18

ZeroCouponBond, 56
ZeroPriceByYield (ZeroCouponBond), 56
ZeroYield (ZeroCouponBond), 56