# Package ‘fmbasics’

January 6, 2018

**Type** Package  
**Title** Financial Market Building Blocks  
**Version** 0.3.0  
**Description** Implements basic financial market objects like currencies, currency pairs, interest rates and interest rate indices. You will be able to use Benchmark instances of these objects which have been defined using their most common conventions or those defined by International Swap Dealer Association (ISDA, [https://www.isda.org](https://www.isda.org)) legal documentation.  
**License** GPL-2  
**URL** [https://github.com/imanuelcostigan/fmbasics](https://github.com/imanuelcostigan/fmbasics),  
[https://imanuelcostigan.github.io/fmbasics/](https://imanuelcostigan.github.io/fmbasics/)  
**BugReports** [https://github.com/imanuelcostigan/fmbasics/issues](https://github.com/imanuelcostigan/fmbasics/issues)  
**Imports** assertthat, fmdates (>= 0.1.2), lubridate (>= 1.6.0), methods, stats, tibble, utils  
**Suggests** covr, knitr, rmarkdown, testthat  
**VignetteBuilder** knitr  
**Encoding** UTF-8  
**LazyData** true  
**RoxygenNote** 6.0.1  
**NeedsCompilation** no  
**Author** Imanuel Costigan [aut, cre]  
**Maintainer** Imanuel Costigan <i.costigan@me.com>  
**Repository** CRAN  
**Date/Publication** 2018-01-06 04:19:05 UTC

---

### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_DiscountFactor</td>
<td>3</td>
</tr>
<tr>
<td>as_InterestRate</td>
<td>3</td>
</tr>
</tbody>
</table>

---


<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_tibble.ZeroCurve</td>
<td>4</td>
</tr>
<tr>
<td>build_zero_curve</td>
<td>5</td>
</tr>
<tr>
<td>CashFlow</td>
<td>5</td>
</tr>
<tr>
<td>CashIndex</td>
<td>6</td>
</tr>
<tr>
<td>Currency</td>
<td>7</td>
</tr>
<tr>
<td>CurrencyConstructors</td>
<td>7</td>
</tr>
<tr>
<td>CurrencyPair</td>
<td>8</td>
</tr>
<tr>
<td>CurrencyPairConstructors</td>
<td>9</td>
</tr>
<tr>
<td>CurrencyPairMethods</td>
<td>10</td>
</tr>
<tr>
<td>DiscountFactor</td>
<td>12</td>
</tr>
<tr>
<td>DiscountFactor-operators</td>
<td>12</td>
</tr>
<tr>
<td>fmbasics</td>
<td>13</td>
</tr>
<tr>
<td>IborIndex</td>
<td>13</td>
</tr>
<tr>
<td>iborindices</td>
<td>14</td>
</tr>
<tr>
<td>indexcheckers</td>
<td>15</td>
</tr>
<tr>
<td>indexshifters</td>
<td>16</td>
</tr>
<tr>
<td>InterestRate</td>
<td>17</td>
</tr>
<tr>
<td>InterestRate-operators</td>
<td>18</td>
</tr>
<tr>
<td>interpolate</td>
<td>19</td>
</tr>
<tr>
<td>interpolate.ZeroCurve</td>
<td>19</td>
</tr>
<tr>
<td>interpolate_dfs</td>
<td>20</td>
</tr>
<tr>
<td>interpolate_zeros</td>
<td>21</td>
</tr>
<tr>
<td>Interpolation</td>
<td>21</td>
</tr>
<tr>
<td>is.CashFlow</td>
<td>22</td>
</tr>
<tr>
<td>is.Currency</td>
<td>23</td>
</tr>
<tr>
<td>is.CurrencyPair</td>
<td>23</td>
</tr>
<tr>
<td>is.DiscountFactor</td>
<td>24</td>
</tr>
<tr>
<td>is.InterestRate</td>
<td>24</td>
</tr>
<tr>
<td>is.Interpolation</td>
<td>25</td>
</tr>
<tr>
<td>is.MultiCurrencyMoney</td>
<td>25</td>
</tr>
<tr>
<td>is.SingleCurrencyMoney</td>
<td>26</td>
</tr>
<tr>
<td>is.ZeroCurve</td>
<td>27</td>
</tr>
<tr>
<td>iso.CurrencyPair</td>
<td>27</td>
</tr>
<tr>
<td>is_valid_compounding</td>
<td>28</td>
</tr>
<tr>
<td>MultiCurrencyMoney</td>
<td>29</td>
</tr>
<tr>
<td>oniaindices</td>
<td>30</td>
</tr>
<tr>
<td>SingleCurrencyMoney</td>
<td>31</td>
</tr>
<tr>
<td>ZeroCurve</td>
<td>32</td>
</tr>
</tbody>
</table>

Index 33
as_DiscountFactor  

Coerce to DiscountFactor

Description
You can coerce objects to the DiscountFactor class using this method.

Usage
as_DiscountFactor(x, ...)

## S3 method for class 'InterestRate'
as_DiscountFactor(x, d1, d2, ...)

Arguments
- x: object to coerce
- ...: other parameters passed to methods
- d1: a Date vector containing the as of date
- d2: a Date vector containing the date to which the discount factor applies

Value
a DiscountFactor object

Examples
library("lubridate")
as_DiscountFactor(InterestRate(c(0.04, 0.05), c(2, 4), 'act/365'), ymd(20140101), ymd(20150101))

as_InterestRate  

Coerce to InterestRate

Description
You can coerce objects to the InterestRate class using this method.

Usage
as_InterestRate(x, ...)

## S3 method for class 'DiscountFactor'
as_InterestRate(x, compounding, day_basis, ...)

## S3 method for class 'InterestRate'
as_InterestRate(x, compounding = NULL, day_basis = NULL, ...)
Arguments

x       object to coerce
...     other parameters passed to methods
compounding  a numeric vector representing the compounding frequency.
day_basis  a character vector representing the day basis associated with the interest rate (see `fmdates::year_frac()`)

Value

an InterestRate object

Examples

```r
library("lubridate")
as_InterestRate(DiscountFactor(0.95, ymd(20130101), ymd(20140101)),
    compounding = 2, day_basis = "act/365")
```

Description

Create a tibble that contains the pillar point maturities in years (using the act/365 convention) and the corresponding continuously compounded zero rates.

Usage

```r
## S3 method for class 'ZeroCurve'
as_tibble(x, ...)
```

Arguments

x       a ZeroCurve object
...     other parameters that are not used by this methods

Value

a tibble with two columns named Years and Zeros.

See Also

`tibble::tibble()`
Examples

library(tibble)
zc <- build_zero_curve()
as_tibble(zc)

Description

This creates a ZeroCurve object from the example data set zerocurve.csv.

Usage

build_zero_curve(interpolation = NULL)

Arguments

interpolation an Interpolation object

Value

a ZeroCurve object using data from zerocurve.csv

Examples

build_zero_curve(LogDFInterpolation())

CashFlow

Create a CashFlow

Description

This allows you to create a CashFlow object.

Usage

CashFlow(dates, monies)

Arguments

dates a Date vector with either the same length as monies or a vector of length one that is recycled
monies a MultiCurrencyMoney object
Value

a CashFlow object that extends tibble::tibble()

See Also


Examples

```r
CashFlow(as.Date("2017-11-15"),
    MultiCurrencyMoney(list(SingleCurrencyMoney(1, AUD()))))
```

---

**CashIndex**

**CashIndex class**

Description

This can be used to represent ONIA like indices (e.g. AONIA, FedFunds) and extends the InterestRateIndex class.

Usage

CashIndex(name, currency, spot_lag, calendar, day_basis, day_convention)

Arguments

- **name**: the name of the index as a string
- **currency**: the currency associated with the index as a Currency object
- **spot_lag**: the period between the index’s fixing and the start of the index’s term
- **calendar**: the calendar used to determine whether the index fixes on a given date as a Calendar
- **day_basis**: the day basis associated with the index (e.g. "act/365")
- **day_convention**: the day convention associated with the index (e.g. "mf")

Value

an object of class CashIndex that inherits from Index

Examples

```r
library(lubridate)
library(fmdates)
# RBA cash overnight rate
CashIndex("AONIA", AUD(), days(0), c(AUSYCalendar()), "act/365", "f")
```
Description

A currency refers to money in any form when in actual use or circulation, as a medium of exchange, especially circulating paper money. This package includes handy constructors for common currencies.

Usage

Currency(iso, calendar)

Arguments

iso a three letter code representing the currency (see ISO4217)
calendar a JointCalendar

Value

an object of class Currency

References


See Also

CurrencyConstructors

Examples

library("fmdates")
Currency("AUD", c(AUSYCalendar()))

Description

These constructors use the following conventions:
Usage

AUD()

EUR()

GBP()

JPY()

NZD()

USD()

CHF()

HKD()

NOK()

Details

<table>
<thead>
<tr>
<th>Creator</th>
<th>Joint calendars</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD()</td>
<td>AUSYCalendar</td>
</tr>
<tr>
<td>EUR()</td>
<td>EUTACalendar</td>
</tr>
<tr>
<td>GBP()</td>
<td>GBLOCalendar</td>
</tr>
<tr>
<td>JPY()</td>
<td>JPTOCalendar</td>
</tr>
<tr>
<td>NZD()</td>
<td>NZAUCalendar, NZWECalendar</td>
</tr>
<tr>
<td>USD()</td>
<td>USNYCalendar</td>
</tr>
<tr>
<td>CHF()</td>
<td>CHZHCalendar</td>
</tr>
<tr>
<td>HKD()</td>
<td>HKHKCalendar</td>
</tr>
<tr>
<td>NOK()</td>
<td>NOOSCalendar</td>
</tr>
</tbody>
</table>

See Also

Other constructors: CurrencyPairConstructors, iborindices, oniaindices

Examples

AUD()
CurrencyPairConstructors

Description
Create an object of class CurrencyPair

Usage
CurrencyPair(base_ccy, quote_ccy, calendar = NULL)

Arguments
- base_ccy: a Currency object
- quote_ccy: a Currency object
- calendar: a JointCalendar object. Defaults to NULL which sets this to the joint calendar of the two currencies and removes any USNYCalendar object to allow currency pair methods to work correctly

Value
a CurrencyPair object

Examples
CurrencyPair(AUD(), USD())

Handy CurrencyPair constructors

Description
These handy CurrencyPair constructors use their single currency counterparts in the obvious fashion.

Usage
AUDUSD()
EURUSD()
NZDUSD()
GBPUSD()
USDJPY()
GBPJPY()
EURGBP()
A collection of methods related to currency pairs.

Usage

- `is_t1(x)`
- `to_spot(dates, x)`
- `to_spot_next(dates, x)`
- `to_forward(dates, tenor, x)`
- `to_today(dates, x)`
- `to_tomorrow(dates, x)`
- `to_fx_value(dates, tenor, x)`
- `invert(x)`
Arguments

- **x**: a CurrencyPair object
- **dates**: a vector of dates from which forward dates are calculated
- **tenor**: the tenor of the value date which can be one of the following: "spot", "spot_next", "today", "tomorrow" and the usual "forward" dates (e.g. `lubridate::months(3)`)

Details

The methods are summarised as follows:

- **is_t1**: Returns `TRUE` if the currency pair settles one good day after trade. This includes the following currencies crossed with the USD: CAD, TRY, PHP, RUB, KZT and PKR.

- **to_spot**: The spot dates are usually two non-NY good day after today. `is_t1()` identifies the pairs whose spot dates are conventionally one good non-NYC day after today. In both cases, if those dates are not a good NYC day, they are rolled to good NYC and non-NYC days using the following convention.

- **to_spot_next**: The spot next dates are one good NYC and non-NYC day after spot rolled using the following convention if necessary.

- **to_forward**: Forward dates are determined using the calendar's `shift()` method rolling bad NYC and non-NYC days using the following convention. The end-to-end convention applies.

- **to_today**: Today is simply dates which are good NYC and non-NYC days. Otherwise today is undefined and returns NA.

- **to_tomorrow**: Tomorrow is one good NYC and non-NYC day except where that is on or after spot. In that case, is is undefined and returns NA.

- **to_value**: Determine common value dates. The supported value date tenors are: "spot", "spot_next", "today", "tomorrow" and the usual "forward" dates (e.g. `lubridate::months(3)`).

- **invert**: Inverts the currency pair and returns new CurrencyPair object.

- **is.CurrencyPair**: Returns `TRUE` if `x` inherits from the CurrencyPair class; otherwise `FALSE`.

Examples

```r
library(lubridate)
is_t1(AUDUSD())
dts <- lubridate::ymd(20170101) + lubridate::days(0:30)
to_spot(dts, AUDUSD())
to_spot_next(dts, AUDUSD())
to_today(dts, AUDUSD())
to_tomorrow(dts, AUDUSD())
to_fx_value(dts, months(3), AUDUSD())
```
DiscountFactor class

Description

The DiscountFactor class is designed to represent discount factors. Checks whether: d1 is less than d2, elementwise, and that both are Date vectors; and value is greater than zero and is a numeric vector. An error is thrown if any of these are not true. The elements of each argument are recycled such that each resulting vectors have equivalent lengths.

Usage

DiscountFactor(value, d1, d2)

Arguments

value a numeric vector containing discount factor values. Must be greater than zero
d1 a Date vector containing the as of date
d2 a Date vector containing the date to which the discount factor applies

Value

a (vectorised) DiscountFactor object

Examples

library("lubridate")
df <- DiscountFactor(c(0.95, 0.94, 0.93), ymd(20130101), ymd(20140101, 20150101))
as_InterestRate(df, 2, "act/365")

Description

A number of different operations can be performed on or with DiscountFactor objects. Methods have been defined for base package generic operations including arithmetic and comparison.
Details

The operations are:

• c: concatenates a vector of DiscountFactor objects
• [: extract parts of a DiscountFactor vector
• [<=: replace parts of a DiscountFactor vector
• rep: repeat a DiscountFactor object
• length: determines the length of a DiscountFactor vector
• *: multiplication of DiscountFactor objects. The end date of the first discount factor object must be equivalent to the start date of the second (or vice versa). Arguments are recycled as necessary.
• /: division of DiscountFactor objects. The start date date of both arguments must be the same. Arguments are recycled as necessary.
• <, >, <=, >=, ==, !=: these operate in the standard way on the discount_factor field.

Description

Implements basic financial market objects like currencies, currency pairs, interest rates and interest rate indices. You will be able to use Benchmark instances of these objects which have been defined using their most common conventions or those defined by International Swap Dealer Association legal documentation.

IborIndex

IborIndex class

Description

This can be used to represent IBOR like indices (e.g. LIBOR, BBSW, CDOR) and extends the Index class.

Usage

IborIndex(name, currency, tenor, spot_lag, calendar, day_basis, day_convention, is_eom)
**Arguments**

name  
the name of the index as a string

currency  
the currency associated with the index as a Currency object

tenor  
the term of the index as a period

spot_lag  
the period between the index’s fixing and the start of the index’s term

calendar  
the calendar used to determine whether the index fixes on a given date as a Calendar

day_basis  
the day basis associated with the index (e.g. "act/365")

day_convention  
the day convention associated with the index (e.g. "mf")

is_eom  
a flag indicating whether or not the maturity date of the index is subject to the end-to-end convention.

**Value**

an object of class IborIndex that inherits from Index

**Examples**

```r
library(lubridate)
library(fmdates)
# 3m AUD BBSW
IborIndex("BBSW", AUD(), months(3), days(0), c(AUSYCalendar()), 
"act/365", "ms", FALSE)
```

---

**ibtorindices**

**Standard IBOR**

**Description**

These functions create commonly used IBOR indices with standard market conventions.

**Usage**

AUDBBSW(tenor)

AUDBBSW1b(tenor)

EURIBOR(tenor)

GBP_LIBOR(tenor)

JPYLIBOR(tenor)

JPYTIBOR(tenor)
NZDBKBM(tenor)
USDLIBOR(tenor)
CHFLIBOR(tenor)
HKDHIBOR(tenor)
NOKNIBOR(tenor)

Arguments

`tenor` the tenor of the IBOR index (e.g. months(3))

Details

The key conventions are tabulated below.

<table>
<thead>
<tr>
<th>Creator</th>
<th>Spot lag (days)</th>
<th>Fixing calendars</th>
<th>Day basis</th>
<th>Day convention</th>
<th>EOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDBBSW()</td>
<td>0</td>
<td>AUSYCalendar</td>
<td>act/365</td>
<td>ms</td>
<td>FALSE</td>
</tr>
<tr>
<td>EURIBOR()</td>
<td>2</td>
<td>EUTACalendar</td>
<td>act/360</td>
<td>mf</td>
<td>TRUE</td>
</tr>
<tr>
<td>GBPLIBOR()</td>
<td>0</td>
<td>GBLOCalendar</td>
<td>act/365</td>
<td>mf</td>
<td>TRUE</td>
</tr>
<tr>
<td>JPYLIBOR()</td>
<td>2</td>
<td>GBLOCalendar</td>
<td>act/365</td>
<td>mf</td>
<td>TRUE</td>
</tr>
<tr>
<td>JPYTIBOR()</td>
<td>2</td>
<td>JPTOCalendar</td>
<td>act/365</td>
<td>mf</td>
<td>FALSE</td>
</tr>
<tr>
<td>NZDBKBM()</td>
<td>0</td>
<td>NZWECalendar, NZAUCalendar</td>
<td>act/365</td>
<td>mf</td>
<td>FALSE</td>
</tr>
<tr>
<td>USDLIBOR()</td>
<td>2</td>
<td>USNYCalendar, GBLOCalendar</td>
<td>act/360</td>
<td>mf</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHFLIBOR()</td>
<td>2</td>
<td>GBLOCalendar</td>
<td>act/360</td>
<td>mf</td>
<td>TRUE</td>
</tr>
<tr>
<td>HKDHIBOR()</td>
<td>0</td>
<td>HKHKCalendar</td>
<td>act/365</td>
<td>mf</td>
<td>FALSE</td>
</tr>
<tr>
<td>NOKNIBOR()</td>
<td>2</td>
<td>NOOSCalendar</td>
<td>act/360</td>
<td>mf</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

There are some nuances to this. Sub-1m LIBOR and TIBOR spot lags are zero days (excepting spot-next rates) and use the following day convention and the overnight USDLIBOR index uses both USNYCalendar and GBLOCalendar calendars.

References

BBSW EURIBOR ICE LIBOR BBA LIBOR TIBOR NZD BKBM OpenGamma Interest Rate Instruments and Market Conventions Guide HKD HIBOR

See Also

Other constructors: `CurrencyConstructors`, `CurrencyPairConstructors`, `oniaindices`
Description
Index class checkers

Usage
is.Nindex(x)

is.Nborindex(x)

is.Ncashindex(x)

Arguments
x an object

Value
TRUE if object inherits from tested class

Examples
is.Index(AONIA())

is.CashIndex(AONIA())

is.IborIndex(AONIA())

indexshifters  Index date shifters

Description
A collection of methods that shift dates according to index conventions.

Usage

to_reset(dates, index)

to_value(dates, index)

to_maturity(dates, index)

## Default S3 method:
to_reset(dates, index)

## Default S3 method:
to_value(dates, index)

## Default S3 method:
to_maturity(dates, index)
**InterestRate**

**Arguments**

- **dates**: a vector of dates to shift
- **index**: an instance of an object that inherits from the Index class.

**Details**

The following describes the default methods. `to_reset()` treats the input dates as value dates and shifts these to the corresponding reset or fixing dates using the index’s spot lag; `to_value()` treats the input dates as reset or fixing dates and shifts them to the corresponding value dates using the index’s spot lag; and `to_maturity()` treats the input dates as value dates and shifts these to the index’s corresponding maturity date using the index’s tenor.

**Value**

- a vector of shifted dates

**Examples**

```r
library(lubridate)

# to_reset
to_reset(ymd(20170101) + days(0:30), AUDBBSW(months(3)))

# to_value
to_value(ymd(20170101) + days(0:30), AUDBBSW(months(3)))

# to_maturity
to_maturity(ymd(20170101) + days(0:30), AUDBBSW(months(3)))
```

---

**Description**

The `InterestRate` class is designed to represent interest rates. Checks whether: the `day_basis` is valid; and the `compounding` is valid. An error is thrown if any of these are not true. The elements of each argument are recycled such that each resulting vectors have equivalent lengths.

**Usage**

```r
InterestRate(value, compounding, day_basis)
```

**Arguments**

- **value**: a numeric vector containing interest rate values (as decimals).
- **compounding**: a numeric vector representing the `compounding` frequency.
- **day_basis**: a character vector representing the day basis associated with the interest rate (see `fmdates::year_frac()`)

**Value**

- a vectorised `InterestRate` object
Examples

```r
library("lubridate")
interestrate<InterestRate(c(0.04, 0.05), c(2, 4), 'act/365')
rate <- InterestRate(0.04, 2, 'act/365')
as_DiscountFactor(rate, ymd(20140101), ymd(20150101))
as_InterestRate(rate, compounding = 4, day_basis = 'act/365')
```

Description

A number of different operations can be performed on or with `InterestRate` objects. Methods have been defined for base package generic operations including arithmetic and comparison.

Details

The operations are:

- `c`: concatenates a vector of `InterestRate` objects
- `[`: extract parts of a `InterestRate` vector
- `<-`: replace parts of a `InterestRate` vector
- `rep`: repeat a `InterestRate` object
- `length`: determines the length of a `InterestRate` vector
- `+`, `-`: addition/subtraction of `InterestRate` objects. Where two `InterestRate` objects are added/subtracted, the second is first converted to have the same compounding and day basis frequency as the first. Numeric values can be added/subtracted to/from an `InterestRate` object by performing the operation directly on the `rate` field. Arguments are recycled as necessary.
- `*`: multiplication of `InterestRate` objects. Where two `InterestRate` objects are multiplied, the second is first converted to have the same compounding and day basis frequency as the first. Numeric values can be multiplied to an `InterestRate` object by performing the operation directly on the `rate` field. Arguments are recycled as necessary.
- `/`: division of `InterestRate` objects. Where two `InterestRate` objects are divided, the second is first converted to have the same compounding and day basis frequency as the first. Numeric values can divide an `InterestRate` object by performing the operation directly on the `rate` field. Arguments are recycled as necessary.
- `<`, `>`, `<=`, `>=`, `==`, `!=`: these operate in the standard way on the `rate` field, and if necessary, the second `InterestRate` object is converted to have the same compounding and day basis frequency as the first.
interpolate  

Interpolate values from an object

Description
Interpolate values from an object

Usage
interpolate(x, ...)

Arguments
x  the object to interpolate.
...  other parameters that defines how to interpolate the object

Value
an interpolated value or set of values

See Also
Other interpolate functions: \texttt{interpolate.ZeroCurve}, \texttt{interpolate_dfs}, \texttt{interpolate_zeros}

interpolate.ZeroCurve  

Interpolate a ZeroCurve

Description
There are two key interpolation schemes available in the stats package: constant and linear interpolation via \texttt{stats::approxfun()} and spline interpolation via \texttt{stats::splinefun()}. The \texttt{interpolate()} method is a simple wrapper around these methods that are useful for the purposes of interpolation financial market objects like zero coupon interest rate curves.

Usage
\#
\#
\#
## S3 method for class 'ZeroCurve'
interpolate(x, at, ...)

Arguments
x  a ZeroCurve object
at  a non-negative numeric vector representing the years at which to interpolate the zero curve
...  unused in this method
**interpolate_dfs**

**Value**

a numeric vector of zero rates (continuously compounded, act/365)

**See Also**

Other interpolate functions: **interpolate_dfs, interpolate_zeros, interpolate**

**Examples**

```r
zc <- build_zero_curve(LogDFInterpolation())
interpolate(zc, c(1.5, 3))
```

---

**interpolate_dfs**  
**Interpolate forward rates and discount factors**

**Description**

This interpolates forward rates and forward discount factors from either a ZeroCurve or some other object that contains such an object.

**Usage**

```r
interpolate_dfs(x, from, to, ...)
interpolate_fwds(x, from, to, ...)
```

```r
## S3 method for class 'ZeroCurve'
interpolate_fwds(x, from, to, ...)
## S3 method for class 'ZeroCurve'
interpolate_dfs(x, from, to, ...)
```

**Arguments**

- `x` the object to interpolate
- `from` a Date vector representing the start of the forward period
- `to` a Date vector representing the end of the forward period
- `...` further arguments passed to specific methods

**Value**

`interpolate_dfs` returns a DiscountFactor object of forward discount factors while `interpolate_fwds` returns an InterestRate object of interpolated simply compounded forward rates.

**See Also**

Other interpolate functions: **interpolate.ZeroCurve, interpolate_zeros, interpolate**
**interpolate_zeros**

**Interpolate zeros**

**Description**

This interpolates zero rates from either a ZeroCurve or some other object that contains such an object.

**Usage**

```r
interpolate_zeros(x, at, compounding = NULL, day_basis = NULL, ...)
```

```r
## S3 method for class 'ZeroCurve'
interpolate_zeros(x, at, compounding = NULL,
                  day_basis = NULL, ...)
```

**Arguments**

- `x` the object to interpolate
- `at` a Date vector representing the date at which to interpolate a value
- `compounding` a valid `compounding` string. Defaults to `NULL` which uses the curve’s native compounding basis
- `day_basis` a valid `day basis` string. Defaults to `NULL` which uses the curve’s native day basis.
- `...` further arguments passed to specific methods

**Value**

an `InterestRate` object of interpolated zero rates with the compounding and day basis requested.

**See Also**

Other interpolate functions: `interpolate.ZeroCurve`, `interpolate_dfs`, `interpolate`

---

**Interpolation**

**Description**

These are lightweight interpolation classes that are used to specify typical financial market interpolation schemes. Their behaviour is dictated by the object in which they defined.
Usage

ConstantInterpolation()
LogDFInterpolation()
LinearInterpolation()
CubicInterpolation()

Value

an object that inherits from the Interpolation class.

Examples

ConstantInterpolation()

---

is.CashFlow

Inherits from CashFlow

Description

Checks whether object inherits from CashFlow class

Usage

is.CashFlow(x)

Arguments

x an R object

Value

TRUE if x inherits from the CashFlow class; otherwise FALSE

See Also


Examples

is.CashFlow(CashFlow(as.Date("2017-11-15"),
MultiCurrencyMoney(list(SingleCurrencyMoney(1, AUD())))))
is.Currency

### Inherit from Currency

**Description**
Checks whether object inherits from Currency class

**Usage**

```
is.Currency(x)
```

**Arguments**

- `x` an R object

**Value**

TRUE if `x` inherits from the Currency class; otherwise FALSE

**Examples**

```
is.Currency(AUD())
```

is.CurrencyPair

### Inherit from CurrencyPair class

**Description**
Inherits from CurrencyPair class

**Usage**

```
is.CurrencyPair(x)
```

**Arguments**

- `x` an R object

**Value**

TRUE if `x` inherits from the CurrencyPair class; otherwise FALSE

**Examples**

```
is.CurrencyPair(AUDUSD())
```
is.\texttt{DiscountFactor} \hspace{1cm} \textit{Inherits from DiscountFactor}\hfill

\textbf{Description} \\
Checks whether object inherits from \texttt{DiscountFactor} class

\textbf{Usage} \\
is.\texttt{DiscountFactor}(x)

\textbf{Arguments} \\
\begin{itemize}
  \item \texttt{x} an \texttt{R} object
\end{itemize}

\textbf{Value} \\
\texttt{TRUE} if \texttt{x} inherits from the \texttt{DiscountFactor} class; otherwise \texttt{FALSE}

\textbf{Examples} \\
is.\texttt{DiscountFactor}\left(\texttt{DiscountFactor}(0.97, \texttt{Sys.Date()}, \texttt{Sys.Date()} + 30)\right)

is.\texttt{InterestRate} \hspace{1cm} \textit{Inherits from InterestRate}\hfill

\textbf{Description} \\
Checks whether object inherits from \texttt{InterestRate} class

\textbf{Usage} \\
is.\texttt{InterestRate}(x)

\textbf{Arguments} \\
\begin{itemize}
  \item \texttt{x} an \texttt{R} object
\end{itemize}

\textbf{Value} \\
\texttt{TRUE} if \texttt{x} inherits from the \texttt{InterestRate} class; otherwise \texttt{FALSE}

\textbf{Examples} \\
is.\texttt{InterestRate}\left(\texttt{InterestRate}(0.04, 2, "act/365")\right)
**is.Interpolation**  
*Check Interpolation class*

**Description**

These methods check whether an interpolation is of a particular scheme.

**Usage**

```r
is.Interpolation(x)
is.ConstantInterpolation(x)
is.LogDFInterpolation(x)
is.LinearInterpolation(x)
is.CubicInterpolation(x)
```

**Arguments**

- `x`  
  an object

**Value**

a logical flag

**Examples**

```r
is.Interpolation(CubicInterpolation())
is.CubicInterpolation(CubicInterpolation())
```

**is.MultiCurrencyMoney**  
*Inherits from MultiCurrencyMoney*

**Description**

Checks whether object inherits from MultiCurrencyMoney class

**Usage**

```r
is.MultiCurrencyMoney(x)
```

**Arguments**

- `x`  
  an R object
**Value**

TRUE if x inherits from the MultiCurrencyMoney class; otherwise FALSE

**See Also**


**Examples**

```r
is.MultiCurrencyMoney(MultiCurrencyMoney(list(SingleCurrencyMoney(1, AUD()))))
```

```r
is.SingleCurrencyMoney(SingleCurrencyMoney(1:5, AUD()))
```
is.ZeroCurve

Inherits from ZeroCurve

Description
Checks whether object inherits from ZeroCurve class

Usage
is.ZeroCurve(x)

Arguments
x an R object

Value
TRUE if x inherits from the ZeroCurve class; otherwise FALSE

Examples
is.ZeroCurve(build_zero_curve())

iso.CurrencyPair

Get ISO

Description
The default method assumes the ISO can be accessed as if it were an attribute with name iso (e.g. x$iso). The method for CurrencyPair concatenates the ISOs of the constituent currencies (e.g. iso(AUDUSD()) returns "AUDUSD") while the methods for CashIndex and IborIndex return the ISO of the index's currency.

Usage
## S3 method for class 'CurrencyPair'
is(x)

iso(x)

## Default S3 method:
is(x)

## S3 method for class 'IborIndex'
is(x)

## S3 method for class 'CashIndex'
is(x)
Arguments

x object from which to extract an ISO

Value

a string of the ISO

Examples

library("lubridate")
iso(AUD())
iso(AUDUSD())
iso(AUDBBSW(months(3)))
iso(AONIA())

is_valid_compounding  Compounding frequencies

Description

A non-exported function that checks whether compounding values frequencies are supported.

Usage

is_valid_compounding(compounding)

Arguments

compounding a numeric vector representing the compounding frequency

Details

Valid compounding values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Simply, T-bill discounting</td>
</tr>
<tr>
<td>0</td>
<td>Simply</td>
</tr>
<tr>
<td>1</td>
<td>Annually</td>
</tr>
<tr>
<td>2</td>
<td>Semi-annually</td>
</tr>
<tr>
<td>3</td>
<td>Tri-annually</td>
</tr>
<tr>
<td>4</td>
<td>Quarterly</td>
</tr>
<tr>
<td>6</td>
<td>Bi-monthly</td>
</tr>
<tr>
<td>12</td>
<td>Monthly</td>
</tr>
<tr>
<td>24</td>
<td>Fortnightly</td>
</tr>
<tr>
<td>52</td>
<td>Weekly</td>
</tr>
<tr>
<td>365</td>
<td>Daily</td>
</tr>
<tr>
<td>Inf</td>
<td>Continuously</td>
</tr>
</tbody>
</table>
**Value**

a flag (TRUE or FALSE) if all the supplied compounding frequencies are supported.

---

**Description**

This class associates a vector of numeric values with a list of currencies. This can be useful for example to store value of cash flows. Internally it represents this information as an extension to a tibble. You are able to bind MultiCurrencyMoney objects by using `rbind()` (see example below).

**Usage**

```r
MultiCurrencyMoney(monies)
```

**Arguments**

- `monies` a list of `SingleCurrencyMoney`

**Value**

a MultiCurrencyMoney object that extends `tibble::tibble()`

**See Also**


**Examples**

```r
mcm <- MultiCurrencyMoney(list(
  SingleCurrencyMoney(1, AUD()),
  SingleCurrencyMoney(2, USD())
))
rbind(mcm, mcm)
```
## Description

These functions create commonly used ONIA indices with standard market conventions.

## Usage

- AONIA()
- EONIA()
- SONIA()
- TONAR()
- NZIONA()
- FedFunds()
- CHFTOIS()
- HONIX()

## Details

The key conventions are tabulated below. All have a zero day spot lag excepting CHFTOIS which has a one day lag (it is a tom-next rate, per 2006 ISDA definitions).

<table>
<thead>
<tr>
<th>Creator</th>
<th>Fixing calendars</th>
<th>Day basis</th>
<th>Day convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>AONIA()</td>
<td>AUSYCalendar</td>
<td>act/365</td>
<td>f</td>
</tr>
<tr>
<td>EONIA()</td>
<td>EUTACalendar</td>
<td>act/360</td>
<td>f</td>
</tr>
<tr>
<td>SONIA()</td>
<td>GBLOCalendar</td>
<td>act/365</td>
<td>f</td>
</tr>
<tr>
<td>TONAR()</td>
<td>JPTOCalendar</td>
<td>act/365</td>
<td>f</td>
</tr>
<tr>
<td>NZIONA()</td>
<td>NZWECalendar, NZAUCalendar</td>
<td>act/365</td>
<td>f</td>
</tr>
<tr>
<td>FedFunds()</td>
<td>USNYCalendar</td>
<td>act/360</td>
<td>f</td>
</tr>
<tr>
<td>CHFTOIS()</td>
<td>CHZHCalendar</td>
<td>act/360</td>
<td>f</td>
</tr>
<tr>
<td>HONIX()</td>
<td>HKHKCalendar</td>
<td>act/365</td>
<td>f</td>
</tr>
</tbody>
</table>

Note that for some ONIA indices, the overnight rate is not published until the following date (i.e. it has publication lag of one day).

## References

AONIA EONIA SONIA TONAR NZIONA FedFunds OpenGamma Interest Rate Instruments and
SingleCurrencyMoney

Market Conventions Guide

See Also

Other constructors: \texttt{CurrencyConstructors, CurrencyPairConstructors, iborindices}

---

**Description**

This class associates a numeric vector with a currency. This is useful for example in representing the value of a derivative. You can concatenate a set \texttt{SingleCurrencyMoney} objects and return a \texttt{MultiCurrencyMoney} object (see example below)

**Usage**

\texttt{SingleCurrencyMoney(value, currency)}

**Arguments**

- \texttt{value}\quad a numeric vector of values
- \texttt{currency}\quad a single \texttt{Currency} object

**Value**

- a \texttt{SingleCurrencyMoney} object

**See Also**

Other money functions: \texttt{CashFlow, MultiCurrencyMoney, is.CashFlow, is.MultiCurrencyMoney, is.SingleCurrencyMoney}

**Examples**

\texttt{SingleCurrencyMoney(1:5, AUD())}
\texttt{c(SingleCurrencyMoney(1, AUD()), SingleCurrencyMoney(100, USD()))}
ZeroCurve

ZeroCurve class

Description
A class that defines the bare bones of a zero-coupon yield curve pricing structure.

Usage
ZeroCurve(discount_factors, reference_date, interpolation)

Arguments
- discount_factors: a DiscountFactor object. These are converted to continuously compounded zero coupon interest rates with an act/365 day basis for internal storage purposes
- reference_date: a Date object
- interpolation: an Interpolation object

Details
A term structure of interest rates (or yield curve) is a curve showing several yields or interest rates across different contract lengths (2 month, 2 year, 20 year, etc...) for a similar debt contract. The curve shows the relation between the (level of) interest rate (or cost of borrowing) and the time to maturity, known as the "term", of the debt for a given borrower in a given currency. For example, the U.S. dollar interest rates paid on U.S. Treasury securities for various maturities are closely watched by many traders, and are commonly plotted on a graph. More formal mathematical descriptions of this relation are often called the term structure of interest rates. When the effect of coupons on yields are stripped away, one has a zero-coupon yield curve.

The following interpolation schemes are supported by ZeroCurve: ConstantInterpolation, LinearInterpolation, LogDFInterpolation and CubicInterpolation. Points outside the calibration region use constant extrapolation on the zero rate.

Value
a ZeroCurve object

See Also
Interpolation

Examples
build_zero_curve()
Index

AONIA (oniaindices), 30
as_DiscountFactor, 3
as_InterestRate, 3
as_tibble.ZeroCurve, 4
AUD (CurrencyConstructors), 7
AUDBBSW (iborindices), 14
AUDBBSW1b (iborindices), 14
AUDNZD (CurrencyPairConstructors), 9
AUDUSD (CurrencyPairConstructors), 9
build_zero_curve, 5

Calendar, 6, 14
Cashflow, 5, 22, 26, 29, 31
CashIndex, 6
CHF (CurrencyConstructors), 7
CHFLIBOR (iborindices), 14
CHFTOIS (oniaindices), 30
componding, 4, 17, 21
componding (is_valid_compounding), 28
ConstantInterpolation (Interpolation), 21

CubicInterpolation (Interpolation), 21
Currency, 6, 7, 9, 14, 31
CurrencyConstructors, 7, 7, 10, 15, 31
CurrencyPair, 8
CurrencyPairConstructors, 8, 9, 15, 31
CurrencyPairMethods, 10

Date, 5, 20, 21
day basis, 21
DiscountFactor, 12, 12, 20, 32
DiscountFactor-operators, 12

EONIA (oniaindices), 30
EUR (CurrencyConstructors), 7
EURCHF (CurrencyPairConstructors), 9
EURGBP (CurrencyPairConstructors), 9
EURIBOR (iborindices), 14
EURNOK (CurrencyPairConstructors), 9

EURUSD (CurrencyPairConstructors), 9
FedFunds (oniaindices), 30
fmbasics, 13
fmbasics-package (fmbasics), 13
fmdates::year_frac(), 4, 17

GBP (CurrencyConstructors), 7
GBPJPY (CurrencyPairConstructors), 9
GBPPLIBOR (iborindices), 14
GBPUSD (CurrencyPairConstructors), 9

HKD (CurrencyConstructors), 7
HKHIBOR (iborindices), 14
HONIX (oniaindices), 30
IborIndex, 13
iborindices, 8, 10, 14, 31
indexcheckers, 15
indexshifters, 16
InterestRate, 17, 18, 20, 21
InterestRate-operators, 18
interpolate, 19, 20, 21
interpolate.ZeroCurve, 19, 19, 20, 21
interpolate_dfs, 19, 20, 20, 21
interpolate_fwds (interpolate_dfs), 20
interpolate_zeros, 19, 20, 21
Interpolation, 21, 32
invert (CurrencyPairMethods), 10
is.CashFlow, 6, 22, 26, 29, 31
is.CashIndex (indexcheckers), 15
is.ConstantInterpolation (is.Interpolation), 25
is.CubicInterpolation (is.Interpolation), 25
is.Currency, 23
is.CurrencyPair, 23
is.DiscountFactor, 24
is.IborIndex (indexcheckers), 15
is.Index (indexcheckers), 15
is.InterestRate, 24
is.Interpolation, 25
is.LinearInterpolation (is.Interpolation), 25
is.LogDFInterpolation (is.Interpolation), 25
is.MultiCurrencyMoney, 6, 22, 26, 29, 31
is.SingleCurrencyMoney, 6, 22, 26, 29, 31
is.ZeroCurve, 27
is_t1 (CurrencyPairMethods), 10
is_valid_compounding, 28
iso (iso.CurrencyPair), 27
iso.CurrencyPair, 27
JointCalendar, 7, 9
JPY (CurrencyConstructors), 7
JPYLIBOR (iborindices), 14
JPTIBOR (iborindices), 14
LinearInterpolation (Interpolation), 21
LogDFInterpolation (Interpolation), 21
MultiCurrencyMoney, 5, 6, 22, 26, 29, 31
NOK (CurrencyConstructors), 7
NOKNIBOR (iborindices), 14
NZD (CurrencyConstructors), 7
NZDBKBM (iborindices), 14
NZDUSD (CurrencyPairConstructors), 9
NZONA (oniaindices), 30
oniaindices, 8, 10, 15, 30
period, 14
rbind(), 29
single currency counterparts, 9
SingleCurrencyMoney, 6, 22, 26, 29, 31
SONIA (oniaindices), 30
stats::approxfun(), 19
stats::splinefun(), 19
tibble, 29
tibble::tibble(), 4, 6, 29
to_forward (CurrencyPairMethods), 10
to_fx_value (CurrencyPairMethods), 10
to_maturity (indexshifters), 16
to_reset (indexshifters), 16
to_spot (CurrencyPairMethods), 10
to_spot_next (CurrencyPairMethods), 10
to_today (CurrencyPairMethods), 10
to_tomorrow (CurrencyPairMethods), 10
to_value (indexshifters), 16
TONAR (oniaindices), 30
USD (CurrencyConstructors), 7
USDCIF (CurrencyPairConstructors), 9
USDHKO (CurrencyPairConstructors), 9
USDJPY (CurrencyPairConstructors), 9
USDLIBOR (iborindices), 14
USDNOK (CurrencyPairConstructors), 9
USNYCalendar, 9
ZeroCurve, 5, 20, 21, 32