Package ‘rutils’

September 15, 2016

Type Package

Title Utility Functions for Simplifying Financial Data Management and Modeling

Version 0.1

Date 2016-05-28

Author Jerzy Pawlowski (algoquant)

Maintainer Jerzy Pawlowski <jp3900@nyu.edu>

Description Functions for managing object names and attributes, applying
functions over lists, managing objects in environments.

License GPL (>= 2)

Depends xts,
quantmod,
RcppRoll,

Suggests knitr,
rmarkdown,
testthat

VignetteBuilder knitr

LazyData true

Repository GitHub

URL https://github.com/algoquant/rutils

RoxygenNote 5.0.1

R topics documented:

`adjust_ohlc` *Adjust the first four columns of* `OHLC` *data using the "adjusted" price column.*

Description

Adjust the first four columns of `OHLC` data using the "adjusted" price column.

Usage

```
adjust_ohlc(oh_lc)
```

Arguments

`oh_lc` an `OHLC` time series of prices in `xts` format.

Details

Adjusts the first four `OHLC` price columns by multiplying them by the ratio of the "adjusted" (sixth) price column, divided by the "close" (fourth) price column.

Value

An `OHLC` time series with the same dimensions as the input series.

Examples

```
# adjust VTI prices
VTI <- adjust_ohlc(env_etf$VTI)
```

`chart_xts` *Plot an* `xts` *time series with custom y-axis range and with vertical background shading.*

Description

A wrapper for function `chart_Series()` from package quantmod.

Usage

```
chart_xts(x_ts, ylim = NULL, in_dex = NULL, ...)
```

Arguments

x_ts	xts time series.
ylim	numeric vector with two elements containing the y-axis range.
in_dex	Boolean vector or xts time series for specifying the shading areas, with TRUE indicating "lightgreen" shading, and FALSE indicating "lightgrey" shading.
...	additional arguments to function chart_Series().

Details

Extracts the chart object and modifies its ylim parameter using accessor and setter functions. Also adds background shading using function add_TA(). The in_dex argument should have the same length as the x_ts time series. Finally the function chart_xts() plots the chart object and returns it invisibly.

Value

A chart object chob returned invisibly.

Examples

```
quantmod::chart_xts(env_etf$VTI["2015-11"],
  name="VTI in Nov 2015", ylim=c(102, 108),
  in_dex=zoo::index(env_etf$VTI["2015-11"]) > as.Date("2015-11-18"))
```

---

diff_it   *Calculate the row differences of a* numeric *vector or matrix.*

Description

Calculate the row differences of a numeric vector or matrix.

Usage

```
diff_it(in_put, lag = 1)
```

Arguments

in_put	a numeric vector or matrix.
lag	integer equal to the number of periods of lag.

Details

Calculates the row differences between rows that are lag rows apart. The leading or trailing stub periods are padded with zeros. Positive lag means that the difference is calculated as the current row minus the row that is lag rows above. (vice versa negative lag). This also applies to vectors, since they can be viewed as single-column matrices.

Value

A vector or matrix with the same dimensions as the input object.

Examples

```
# diff vector by 2 periods
diff_it(1:10, lag=2)
# diff matrix by negative 2 periods
diff_it(matrix(1:10, ncol=2), lag=-2)
```

---

diff_ohlc — *Calculate the reduced form of an OHLC time series, or calculate the standard form from the reduced form of an OHLC time series.*

Description

Calculate the reduced form of an OHLC time series, or calculate the standard form from the reduced form of an OHLC time series.

Usage

```
diff_ohlc(oh_lc, re_duce = TRUE, ...)
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
re_duce	Boolean should the reduced form be calculated or the standard form? (default is TRUE)
...	additional arguments to function xts::diff.xts().

Details

The reduced form of an OHLC time series is obtained by calculating the time differences of its Close prices, and by calculating the differences between its Open, High, and Low prices minus the Close prices. The standard form is the original OHLC time series, and can be calculated from its reduced form by reversing those operations.

Value

An OHLC time series with five columns for the Open, High, Low, Close prices, and the Volume, and with the same time index as the input series.

Examples

```
# calculate reduced form of an OHLC time series
diff_VTI <- rutils::diff_ohlc(env_etf$VTI)
# calculate standard form of an OHLC time series
VTI <- rutils::diff_ohlc(diff_VTI, re_duce=FALSE)
identical(VTI, env_etf$VTI[, 1:5])
```

`diff_xts`	*Calculate the time differences of an* `xts` *time series.*

Description

Calculate the time differences of an `xts` time series.

Usage

```
diff_xts(x_ts, lag = 1, ...)
```

Arguments

`x_ts`	an `xts` time series.
`lag`	integer equal to the number of time periods of lag.
`...`	additional arguments to function `xts::diff.xts()`.

Details

Calculates the time differences of an `xts` time series and pads with `zeros` instead of `NA`s. Positive `lag` means differences are calculated with values from `lag` periods in the past (vice versa negative `lag`). The function `diff()` is just a wrapper for `diff.xts()` from package xts, but it pads with `zeros` instead of `NA`s.

Value

An `xts` time series with the same dimensions and the same time index as the input series.

Examples

```
# calculate time differences over lag by 10 periods
rutils::diff_xts(env_etf$VTI, lag=10)
```

`do_call`	*Recursively apply a function to a list of objects, such as* `xts` *time series.*

Description

Performs a similar operation as `do.call()`, but using recursion, which is much faster and uses less memory. The function `do_call()` is a generalization of function `do_call_rbind()`.

Usage

```
do_call(func_tion, li_st, ...)
```

Arguments

`func_tion`	name of function that returns a single object from a list of objects.
`li_st`	list of objects, such as `vectors`, `matrices`, `data frames`, or `time series`.
`...`	additional arguments to function `func_tion()`.

Details

Performs lapply loop, each time binding neighboring elements and dividing the length of li_st by half. The result of performing do_call(rbind, list_xts) on a list of xts time series is identical to performing do.call(rbind, list_xts). But do.call(rbind, list_xts) is very slow, and often causes an 'out of memory' error.

Value

A single vector, matrix, data frame, or time series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
# split time series into daily list
list_xts <- split(x_ts, "days")
# rbind the list back into a time series and compare with the original
identical(x_ts, do_call(rbind, list_xts))
```

---

do_call_assign — *Apply a function to a list of objects, merge the outputs into a single object, and assign the object to the output environment.*

Description

Apply a function to a list of objects, merge the outputs into a single object, and assign the object to the output environment.

Usage

```
do_call_assign(func_tion, sym_bols = NULL, out_put, env_in = .GlobalEnv,
  env_out = .GlobalEnv, ...)
```

Arguments

func_tion	name of function that returns a single object (vector, xts time series, etc.)
sym_bols	vector of strings with names of input objects.
out_put	string with name of output object.
env_in	environment containing the input sym_bols.
env_out	environment for creating the out_put.
...	additional arguments to function func_tion().

Details

Performs an lapply loop over sym_bols, applies the function func_tion(), merges the outputs into a single object, and creates the object in the environment env_out. The output object is created as a side effect, while its name is returned invisibly.

Value

A single object (matrix, xts time series, etc.)

Examples

```
new_env <- new.env()
do_call_assign(
   func_tion=ex_tract,
   sym_bols=env_etf$sym_bols,
   out_put="price_s",
   env_in=env_etf, env_out=new_env)
```

`do_call_rbind` *Recursively* `'rbind'` *a list of objects, such as* `xts` *time series.*

Description

Performs the same operation as `do.call(rbind, li_st)`, but using recursion, which is much faster and uses less memory. This is the same function as '`do.call.rbind`' from package 'qmao'.

Usage

```
do_call_rbind(li_st)
```

Arguments

`li_st`	list of objects, such as `vectors`, `matrices`, `data frames`, or `time series`.

Details

Performs lapply loop, each time binding neighboring elements and dividing the length of `li_st` by half. The result of performing `do_call_rbind(list_xts)` on a list of `xts` time series is identical to performing `do.call(rbind, list_xts)`. But `do.call(rbind, list_xts)` is very slow, and often causes an 'out of memory' error.

Value

A single `vector`, `matrix`, `data frame`, or `time series`.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
# split time series into daily list
list_xts <- split(x_ts, "days")
# rbind the list back into a time series and compare with the original
identical(x_ts, do_call_rbind(list_xts))
```

`end_points` *Calculate an index (integer vector) of equally spaced end points for a time series.*

Description

Calculate an index (integer vector) of equally spaced end points for a time series.

Usage

```
end_points(x_ts, inter_val = 10, off_set = 0)
```

Arguments

`x_ts`	vector or time series.
`inter_val`	the number of data points per interval.
`off_set`	the number of data points in the first interval (stub interval).

Details

The end points divide the time series into equally spaced intervals. The off_set argument shifts the end points forward and creates an initial stub interval.

Value

An `integer` vector of equally spaced end points.

Examples

```
# calculate end points with initial stub interval
end_points(env_etf$VTI, inter_val=7, off_set=4)
```

`etf_data` *The etf_data dataset contains a single environment called env_etf, which includes daily* `OHLC` *time series data for a portfolio of symbols.*

Description

The env_etf environment includes daily `OHLC` time series data for a portfolio of symbols, and reference data:

sym_bols a `vector` of `strings` with the portfolio symbols.

price_s a single `xts` time series containing daily closing prices for all the `sym_bols`.

re_turns a single `xts` time series containing daily returns for all the `sym_bols`.

Individual time series "VTI", "VEU", etc., containing daily `OHLC` prices for the `sym_bols`.

Usage

```
data(etf_data)  # not required - data is lazy load
```

Format

Each xts time series contains the columns:

Open Open prices

High High prices

Low Low prices

Close Close prices

Volume daily trading volume

Adjusted Adjusted closing prices

Examples

```
# data(etf_data)  # not needed - data is lazy load
# get first six rows of OHLC prices
head(env_etf$VTI)
chart_Series(x=env_etf$VTI["2009-11"])
```

---

ex_tract *Extract columns of prices from an* OHLC *time series.*

Description

Extract columns of prices from an OHLC time series.

Usage

```
ex_tract(oh_lc, col_name = "Close")
```

Arguments

oh_lc	an OHLC time series.
col_name	string with the field name of the column to be be extracted. (default is "Close")

Details

Extracts columns of prices from an OHLC time series by grepping column names for the col_name string. The OHLC column names are assumed to be in the format "symbol.field_name", for example "VTI.Close". Performs a similar operation to the extractor functions Op(), Hi(), Lo(), Cl(), and Vo(), from package quantmod. But ex_tract() is able to handle symbols like "LOW", which the function Lo() can't handle. The col_name argument is partially matched, for example "vol" is matched to "Volume".

Value

A single column OHLC time series in xts format.

Examples

```
# get close prices for VTI
ex_tract(env_etf$VTI)
# get volumes for VTI
ex_tract(env_etf$VTI, col_name="vol")
```

get_symbols *Download time series data from an external source (by default* OHLC *prices from* YAHOO*), and save it into an environment.*

Description

Download time series data from an external source (by default OHLC prices from YAHOO), and save it into an environment.

Usage

```
get_symbols(sym_bols, env_out, start_date = "2007-01-01",
  end_date = Sys.Date())
```

Arguments

sym_bols	vector of strings representing instrument symbols (tickers).
env_out	environment for saving the data.
start_date	start date of time series data. (default is "2007-01-01")
end_date	end date of time series data. (default is Sys.Date())

Details

The function get_symbols downloads OHLC prices from YAHOO into an environment, adjusts the prices, and saves them back to that environment. The function get_symbols() calls the function getSymbols.yahoo() to download the OHLC prices, and performs a similar operation to the function getSymbols() from package quantmod. But get_symbols() is faster (because it's more specialized), and is able to handle symbols like "LOW", which getSymbols() can't handle because the function Lo() can't handle them. The start_date and end_date must be either of class Date, or a string in the format "YYYY-mm-dd". get_symbols() returns invisibly the vector of sym_bols.

Value

A vector of sym_bols returned invisibly.

Examples

```
## Not run:
new_env <- new.env()
get_symbols(sym_bols=c("MSFT", "XOM"),
            env_out=new_env,
            start_date="2012-12-01",
            end_date="2015-12-01")

## End(Not run)
```

lag_it — *Apply a lag to a* `numeric` *vector or matrix.*

Description

Apply a lag to a `numeric` vector or matrix.

Usage

```
lag_it(in_put, lag = 1)
```

Arguments

`in_put`	a `numeric` vector or matrix.
`lag`	integer equal to the number of periods of lag.

Details

Applies a lag to a vector or matrix, by shifting its values by a certain number of rows, equal to the integer `lag`, and pads the leading or trailing stub periods with `zeros`. Positive `lag` means that values in the current row are replaced with values from the row that are `lag` rows above. (vice versa negative `lag`). This also applies to vectors, since they can be viewed as single-column matrices.

Value

A vector or matrix with the same dimensions as the input object.

Examples

```
# lag vector by 2 periods
lag_it(1:10, lag=2)
# lag matrix by negative 2 periods
lag_it(matrix(1:10, ncol=2), lag=-2)
```

lag_xts — *Apply a time lag to an* `xts` *time series.*

Description

Apply a time lag to an `xts` time series.

Usage

```
lag_xts(x_ts, k = 1, ...)
```

Arguments

`x_ts`	an `xts` time series.
`k`	integer equal to the number of time periods of lag. (default is 1)
`...`	additional arguments to function `xts::lag_xts()`.

Details

Applies a time lag to an `xts` time series and pads with the first and last values instead of `NA`s. Positive lag `k` means values from `k` periods in the past are moved to the present. Negative lag `k` moves values from the future to the present. The function `lag()` is just a wrapper for function `lag_xts()` from package xts, but it pads with the first and last values instead of `NA`s.

Value

An `xts` time series with the same dimensions and the same time index as the input `x_ts` time series.

Examples

```
# lag by 10 periods
rutils::lag_xts(env_etf$VTI, k=10)
```

`na_me` *Extract the name of an* `OHLC` *time series from its first column name.*

Description

Extract the name of an `OHLC` time series from its first column name.

Usage

```
na_me(x_ts, field = 1)
```

Arguments

`x_ts`	`OHLC` time series.
`field`	the integer index of the field to be extracted.

Details

Extracts the symbol name (ticker) from the name of the first column of an `OHLC` time series. The column name is assumed to be in the format "symbol.Open". It can also extract the field name after the "." separator, for example "Open" from "SPY.Open".

Value

A `string` with the name of time series.

Examples

```
# get name for VTI
na_me(env_etf$VTI)
```

roll_max — *Calculate the rolling maximum of an* xts *time series over a sliding window (lookback period).*

Description

Performs the same operation as function `runMax()` from package TTR, but using vectorized functions, so it's a little faster.

Usage

```
roll_max(x_ts, win_dow)
```

Arguments

`x_ts`	an `xts` time series containing one or more columns of data.
`win_dow`	the size of the lookback window, equal to the number of data points for calculating the rolling sum.

Details

For example, if win_dow=3, then the rolling sum at any point is equal to the sum of `x_ts` values for that point plus two preceding points. The initial values of roll_max() are equal to cumsum() values, so that roll_max() doesn't return any NA values.

Value

An `xts` time series with the same dimensions as the input series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
roll_max(x_ts, win_dow=3)
```

roll_sum — *Calculate the rolling sum of an* xts *time series over a sliding window (lookback period).*

Description

Performs the same operation as function `runSum()` from package TTR, but using vectorized functions, so it's a little faster.

Usage

```
roll_sum(x_ts, win_dow)
```

Arguments

x_ts	an xts time series containing one or more columns of data.
win_dow	the size of the lookback window, equal to the number of data points for calculating the rolling sum.

Details

For example, if win_dow=3, then the rolling sum at any point is equal to the sum of x_ts values for that point plus two preceding points. The initial values of roll_sum() are equal to cumsum() values, so that roll_sum() doesn't return any NA values.

Value

An xts time series with the same dimensions as the input series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
roll_sum(x_ts, win_dow=3)
```

---

to_period — *Aggregate an OHLC time series to a lower periodicity.*

Description

Given an OHLC time series at high periodicity (say seconds), calculates the OHLC prices at lower periodicity (say minutes).

Usage

```
to_period(oh_lc, period = "minutes", k = 1,
  end_points = xts::endpoints(oh_lc, period, k))
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
period	aggregation interval ("seconds", "minutes", "hours", "days", "weeks", "months", "quarters", and "years").
k	number of periods to aggregate over (for example if period="minutes" and k=2, then aggregate over two minute intervals.)
end_points	an integer vector of end points.

Details

The function to_period() performs a similar aggregation as function to.period() from package xts, but has the flexibility to aggregate to a user-specified vector of end points. The function to_period() simply calls the compiled function toPeriod() (from package xts), to perform the actual aggregations. If end_points are passed in explicitly, then the period argument is ignored.

Value

A OHLC time series of prices in xts format, with a lower periodicity defined by the end_points.

Examples

```
# define end points at 10-minute intervals (SPY is minutely bars)
end_points <- rutils::end_points(SPY["2009"], inter_val=10)
# aggregate over 10-minute end_points:
to_period(x_ts=SPY["2009"], end_points=end_points)
# aggregate over days:
to_period(oh_lc=SPY["2009"], period="days")
# equivalent to:
to.period(x=SPY["2009"], period="days", name=rutils::na_me(SPY))
```

Index