Package 'regts'

February 14, 2020

Type Package

Title Regular Timeseries

Version 1.1.0

Author Rob van Harrevelt [aut, cre],
Anita van der Roest [aut]

Maintainer Rob van Harrevelt <rvanharrevelt@gmail.com>

Description An extension of the ts class with enhanced possibilities for period selection. regts particularly aims at yearly, quarterly and monthly timeseries. It also supports labels that can be used to describe the timeseries in multivariate timeseries objects. The package includes functions for reading and writing timeseries from Excel and csv files, conversion of timeseries to growth series and the inverse transformation, special aggregation methods for growth series, and several other functions.

License GPL-3

LazyData TRUE

RoxygenNote 7.0.2

Roxygen list(markdown = TRUE, old_usage = TRUE)

Imports Rcpp,
data.table,
readxl (>= 1.2.0),
cellranger,
tibble,
openxlsx,
stringr,
testthat

Suggests knitr,
rmarkdown,
zoo,
xts,
timeSeries,
tseries,
tempdisagg,

Hmisc,
lubridate

LinkingTo Rcpp

VignetteBuilder knitr

R topics documented:

`aggregate_gr` *Convert timeseries with absolute or relative changes to a lower frequency*

Description

This function implements temporal aggregation for timeseries with absolute, relative or percentage changes. As shown in vignette *"Temporal Aggregation of (Growth) Timeseries"*, the standard function `aggregate` does not yield correct results for these type of timeseries.

Usage

```
aggregate_gr(x, method = c("dif1s", "dif1", "pct", "rel"), nfrequency = 1)
```

Arguments

`x`	a `ts` or `regts` object
`method`	Aggregation method: `"dif1s"`, `"dif1"`, `"pct"` or `"rel"`. See Details.
`nfrequency`	the frequency of the result. This should be higher than the frequency of timeseries `x`

Details

There are methods for different types of input timeseries. The `dif1s` and `dif1` methods assume that the input timeseries contain a first difference (for `dif1s` the input is also scaled). The result is a first difference in the output frequency. The `pct` and `rel` methods assume timeseries that contain percentage or relative changes of a timeseries with only positive values. They calculate the exact percentage or relative change for the output timeseries. More details for the various methods are provided in vignette *"Temporal Aggregation of (Growth) Timeseries"*.

As explained before, the `pct` and `rel` methods assume that `x` is is a percentage or relative change of a series with only positive values. This imposes a restriction on `x`: for the `pct` method, x >= -100%, and for `rel`, `x >= -1`. Function `aggregate_gr` gives an error if this condition is not satisfied.

Value

a `regts` with frequency `nfrequency`

Examples

```
ts_q <- regts(abs(rnorm(10)), start = "2016Q1")
aggregate_gr(ts_q, method = "dif1s")

ts_m <- regts(matrix(abs(rnorm(20)), ncol = 2), start = "2017M1", names = c("a", "b"))
aggregate_gr(ts_m, method = "rel", nfrequency = 4)
```

`as.data.frame` *Convert a* `regts` *to a* `data.frame`

Description

Convert a `regts` to a `data.frame`

Usage

```
## S3 method for class 'regts'
as.data.frame(x, ..., rowwise = FALSE, row_names = TRUE,
  period_as_date = FALSE)
```

Arguments

`x`	a `regts`
`...`	additional arguments to be passed to methods.
`rowwise`	a logical value: should the timeseries be stored rowwise or columnwise in the data frame? Defaults to `FALSE`
`row_names`	Whether to create row names. If `FALSE`, then an additional column with name `"period"` or `"name"` is created for columnwise or rowwise timeseries, respectively.
`period_as_date`	A logical (default `FALSE`). If `TRUE` the periods are stored as `Date` objects. Depending on arguments `rowwise` and `row_names` the periods may appear in the row or column names of the result data frame. In that case the dates are coerced to character vectors, using the standard date format `"%Y-%m-%d"` (see the documentation of function `strptime` for more information about date formats).

Details

If the `regts` has labels and argument `rowwise` is `FALSE`, then the labels are added to columns of the data frame These labels are visible in the data viewer.

Value

A `data.frame`

Examples

```
ts <- regts(matrix(1:4, ncol = 2) , start = "2015Q3", names = c("a", "b"),
           labels = c("Timeseries a", "Timeseries b"))
print(as.data.frame(ts))
```

`as.list` *Convert a* `regts` *to a list of univariate regts objects*

Description

This function converts a `regts` to a list of univariate `regts` objects.

Usage

```
## S3 method for class 'regts'
as.list(x, ...)
```

Arguments

`x`	a `regts` object
`...`	arguments passed to methods (not used in the default implementation)

Value

a list of univariate `regts` objects

See Also

`list2env` and `cbind`

Examples

```
regts1 <- regts(matrix(1:6, ncol = 2), start = "2015Q3", names = c("a", "b"))

# convert regts1 to a list
ts_list1 <- as.list(regts1)

# use the within function to modify timeseries and create new timeseries
ts_list2 <- within (ts_list1, {
   b["2015q2"] <- 2
   c <- a * b
   d <- lag(c)
 })

# use functions do.call and cbind to convert
# the list of timeseries objects to a multivariate regts
regts2 <- do.call(cbind, ts_list2)

# transfer all timeseries in the list to the global environment
list2env(ts_list2, .GlobalEnv)
```

as.regts *Coerce an object to a* regts *timeseries object*

Description

Coerce an object to a regts timeseries object

Usage

```
as.regts(x, ...)

## S3 method for class 'ts'
as.regts(x, ...)

## S3 method for class 'data.frame'
as.regts(x, time_column = 0, numeric = TRUE, fun = period, strict = TRUE, ...)

## S3 method for class 'matrix'
as.regts(x, numeric = TRUE, fun = period, strict = TRUE, ...)

## S3 method for class 'list'
as.regts(x, union = TRUE, ...)

## S3 method for class 'numeric'
as.regts(x, fun = period, strict = TRUE, ...)

## Default S3 method:
as.regts(x, ...)
```

Arguments

x	an arbitrary R object.
...	arguments passed to fun.
time_column	the column names or numbers of the data frame in which the time (periods) is stored. Specify 0 if the index is in the row names of the data frame. If time_column has length > 1, then argument fun should be a function which converts a data frame to period vector.
numeric	logical: should non numeric values be converted to numeric data. By default they are converted to numeric. This can be changed by setting numeric = FALSE.
fun	a function for converting values in the row names, time column(s), or names of a numeric vector to period objects. Normally this is a function which converts a vector to a period vector (for example function period). See argument time_column for exceptions.
strict	A logical. If TRUE (the default) all periods between the start and the end period must be present. Otherwise the timeseries are filled with NA for the missing periods.

union — A logical (default TRUE). Only used in as.regts.list. If the list contains multiple timeseries and union isTRUE, then the period range of the result is the union of the period ranges of the individual timeseries objecct in the list. Otherwise the period range is the intersection.

Value

a regts object

Methods (by class)

- ts: Coerce a ts to a regts
- data.frame: Convert a data.frame to a regts. The time should be stored in the row numbers of the matrix
- matrix: Convert a matrix to a regts
- list: Convert a list to a regts. At least one of the elements of the list should contain a timeseries.
- numeric: Convert a numeric vector to a regts
- default: Default method to convert an R object to a regts. This method first employs as.ts and then as.regts.ts

See Also

regts, is.regts, as.data.frame, as.list, start_period, end_period

Examples

```
# convert a ts to regts
x <- ts(1:3, start = c(2015,3), frequency = 4)
x <- as.regts(x)

# Now three examples for converting a data.frame

# create a data frame with timeseries and with the
# time index in the rownames, and convert to a regts
df <- data.frame(a = 1:3)
rownames(df) <- c("2015Q3", "2015Q4", "2016Q1")
ts <- as.regts(df)

# create a data frame with the time index in the first column and special
# time format "2015 3" instead of "2015Q3", and convert to regts
df <- data.frame(periods = c("2015 3", "2015 4", "2016 1"),  a = 1:3)
ts <- as.regts(df, time_column = 1, frequency = 4)

# create a data frame with non numeric data and convert to regts
# Strings containing non numeric values are converted to NA
# Logical values TRUE/FALSE are converted to 1/0
df <- data.frame(a = c("1", "2", "X"), b
= c(TRUE, FALSE, TRUE), stringsAsFactors = FALSE)
as.regts(df)
```

```
# data frame with the years in the first column and quarters in the
# second column
df <- data.frame(years = c(2018, 2018), quarters = c(1, 2), a = 1:2)
fun <- function(x) {period(paste(x[[1]], x[[2]]), frequency = 4)}
as.regts(df, time_column = c("years", "quarters"), fun = fun)
```

`as_matrix` *Convert a* `ts` *to a* `matrix`

Description

This function converts a `ts` object to a normal `matrix`, i.e. a matrix without timeseries class and attributes.The periods are stored in the row or column names of the returned matrix, depending on argument `rowwise`.

Usage

```
as_matrix(x, ...)

## S3 method for class 'ts'
as_matrix(x, rowwise = FALSE, ...)
```

Arguments

`x`	a `ts` or `regts`
`...`	additional arguments to be passed to methods.
`rowwise`	a logical value: should the timeseries be stored rowwise or columnwise in the matrix? Defaults to `FALSE`

Details

The function behaves differently than base method `as.matrix`. If the input timeseries is a matrix with timeseries attributes, then `as.matrix` just returns the input value. If the input timeseries is not a matrix (a univariate timeseries with vector data), then `as.matrix` returns a matrix without row and column names and without timeseries attributes. In contrast, `as_matrix` always returns a matrix without timeseries attributes.

Value

A `matrix`

Methods (by class)

- `ts`: Coerce a `ts` to a matrix without timeseries class and attributes

Examples

```
ts <- regts(matrix(1:4, ncol = 2) , start = 2015, names = c("a", "b"))
as_matrix(ts, rowwise = TRUE)
```

---

cbind *Bind two or more timeseries*

Description

Bind two or more timeseries objects with a common frequency. By default, the period range of the result is the union of the period ranges of the individual timeseries. The result is padded with NAs if necessary. If argument union is false, then the period range of the result is the intersection of the period ranges.

Usage

```
## S3 method for class 'regts'
cbind(..., union = TRUE, suffixes)
```

Arguments

...	two or more univariate or multivariate timeseries, or objects which can be coerced to timeseries
union	if TRUE, then the period range of the result is the union of the period ranges of the joined objects (the result is padded with NAs if necessary). If FALSE, then the period range of the result is the intersection of the period ranges of the joined objects.
suffixes	Suffixes appended to the column names for all overlapping columns. This argument is obligatory if the timeseries have overlapping column names. Length suffixes must be equal to the number of joined timeseries or objects.

See Also

as.list

Examples

```
a <- regts(1:5, start = "2011Q1")
b <- regts(matrix(11:15, nc = 1), start = "2011Q2")
cbind(a, b)
cbind(a, b, union = FALSE)
x1 <- regts(matrix(1:27, nc = 3), start = "2008Q4", names = c("a", "b", "c"))
x2 <- regts(matrix(1:27, nc = 3), start = "2008Q4", names = c("a", "c", "d"))
cbind(x1, x2, suffixes = c("_1","_2"))
```

change_frequency *Change the frequency of a* period *or* period_range *object.*

Description

A period can be converted to lower frequency. For example, a month "2017M4" can be converted to the quarter "2017Q2". The old frequency should be divisible by the new frequency.

A period_range can be converted to both lower and higher frequency. For example, the range "2017Q3/2018Q1" can be converted to the month range "2017M7/2018M3" or the year range "2017/2018". If the period_range is converted to lower frequency, then the old frequency should be divisible by the new frequency. If the range is converted to higher frequency, then the new frequency should be divisible by the old frequency.

Usage

```
change_frequency(x, new_frequency, ...)

## S3 method for class 'period'
change_frequency(x, new_frequency, ...)

## S3 method for class 'period_range'
change_frequency(x, new_frequency, ...)
```

Arguments

x	a period or period_range
new_frequency	the new_frequency
...	arguments passed to methods (not used in package regts)

Value

a period or period_range (depending on the type of argument x) with the new frequency

Methods (by class)

- period: Change the frequency of a period to lower frequency
- period_range: Change the frequency of a period_range to higher or lower frequency

Examples

```
p <- period("2017M4")
change_frequency(p, 4)

range <- period_range("2017Q3/2018Q1")
change_frequency(range, 12)
change_frequency(range, 1)
```

cvgdif *Calculate the 'convergence difference'*

Description

`cvgdif` calculates the difference between two numeric vectors `x1` and `x2` according to `|x1 -x2| / max(|x2|,1)`. This difference is equivalent to the convergence test employed in the package `isismdl`.

Usage

```
cvgdif(x1, x2)
```

Arguments

`x1`	first numeric vector
`x2`	second numeric vector

Value

the 'convergence difference' as described above

See Also

`tsdif`

Examples

```
# create two timeseries x1 and x2
x1 <- regts(matrix(data = rep(1:27), nc = 3), start = "2008Q4",
           names = c("a", "b", "c"))
x2 <- x1 + 0.001
colnames(x2) <- c("a", "b", "d")

# calculate the differences
cvgdif(x1, x2)
```

diff_ts *Lagged differences of a timeseries*

Description

Returns suitably lagged and iterated differences of a timeseries. This function works similarly as diff, except that the period range of the result is the same as that of the input timeseries. This behaviour can be changed by specifying argument keep_range.

Usage

```
diff_ts(x, lag = 1, differences = 1, keep_range = TRUE, ...)

## S3 method for class 'ts'
diff_ts(x, lag = 1, differences = 1, keep_range = TRUE, ...)

## Default S3 method:
diff_ts(x, lag = 1, differences = 1, keep_range = TRUE, ...)
```

Arguments

x	a univariate or multivariate timeseries. Can also be a vector, matrix or data frame (see details).
lag	an integer indicating which lag to use
differences	an integer indicating the order of the difference.
keep_range	if TRUE (the default), then the output timeseries has the same period range as the input timeseries. Then the result timeseries will have lag + differences -1 NA values at the beginning. If FALSE then the result timeseries starts lag + differences -1 periods later than the input timeseries.
...	further arguments to be passed to or from methods (currently not used in package regts)

Details

Vector, matrix and data frame arguments are first converted to a regts with function regts. This conversion results in a timeseries with frequency 1 and starting at year 1.

See Also

lag_ts and lead_ts

Examples

```
x <- regts(1:10, start = "2018q3")
diff_ts(x)
diff_ts(x, lag = 2, keep_range = FALSE)
```

disagg *Disaggregation of timeseries using cubic spline interpolation.*

Description

This function converts a timeseries to a timeseries with higher frequency, for example a yearly timeseries to a quarterly timeseries. Cubic spline interpolation is used to interpolate between the low frequency observations.

Usage

```
disagg(x, nfrequency, constraint = c("average", "sum", "first", "last"),
  conds = c("natural", "not-a-knot"))
```

Arguments

x	a regts or ts object
nfrequency	the frequency of the result. This should be higher than the frequency of time-series x.
constraint	Constraint on the high frequency result. Possible values are "average", "sum", "first" and "last". Either the average, the sum, the first or last value of the resulting high-frequency series should be equal to the corresponding low-frequency value.
conds	a character specifying the boundary conditions: "natural" or "not-a-knot". Default is "natural". See details.

Details

Argument conds can be used to select the boundary conditions for the cubic spline interpolation. Choose "natural" for a natural cubic spline (zero second derivatives at the end points). For "not-a-knot" the third derivative is continuous at the second and one but last point.

Leading and trailing NA values are removed before the interpolation.

See Also

Alternative spline methods are available in package tempdisagg

Examples

```
# construct quarterly series
q <- regts(matrix(c(1:3, NA, -3, -5),  ncol = 2),
           period = "2017Q2/2017Q4", names = c("a", "b"))

disagg(q, nfrequency = 12)

disagg(q, constraint = "last", conds = "not-a-knot", nfrequency = 12)
```

frequency *Return the frequency of a* period *or a* period_range

Description

This is an extension to S3 generic function frequency. Function now also returns the frequency of a period or a period_range

Usage

```
## S3 method for class 'period'
frequency(x, ...)

## S3 method for class 'period_range'
frequency(x, ...)
```

Arguments

x	a period or a period_range
...	additional arguments for future methods

Value

the frequency of the period or the period_range

Methods (by class)

- period: frequency of a period object
- period_range: frequency of a period_range object

Examples

```
p <- period("2016Q1")
freq <- frequency(p)

p <- period_range("2016Q1", "2018Q2")
freq <- frequency(p)#'
```

get_periods — *Return all periods in a* period_range *or timeseries.*

Description

The periods in a period_range or timeseries are returned as a period vector

Usage

```
get_periods(x, ...)
```

Arguments

x	a period_range or times series (ts or regts).
...	arguments passed to methods (currently not used).

See Also

get_period_range

Examples

```
# example for period range
range <- period_range("2018m1/2018m3")
get_periods(range)

# example for timeseries
x <- regts(1:3, start = "2010Q4")
get_periods(x)
```

get_period_range — *Return the period_range of a timeseries.*

Description

Return the period_range of a timeseries.

Usage

```
get_period_range(x)
```

Arguments

x	a timeseries (ts or regts)

Value

a period_range

See Also

get_periods

---

get_subperiod *Return the subperiod of a* period

Description

This function returns the subperiod within a year. For example, for period 2011Q3 the function returns 3.

Usage

```
get_subperiod(x)
```

Arguments

x	a period

Value

the subperiod of a period

See Also

get_year

Examples

```
get_subperiod(period("2010Q3"))
```

get_year *This function returns the year of a* period

Description

This function returns the year of a period

Usage

```
get_year(x)
```

Arguments

x	a period

Value

the year

See Also

get_subperiod

Examples

```
get_year(period("2010Q3"))
```

growth *Return the relative change of a timeseries*

Description

Function growth computes the relative change of a timeseries. The one period relative change of a timeseries is defined as: growth(x) = (x[t] -x[t-1]) / |x[t-1]|

The n period relative change of a timeseries is defined as: growth(x,n) = (x[t] -x[t-n]) / |x[t-n]|

The formula implies that when the timeseries decreases, the result will be negative regardless of the sign of x. The function also works for multivariate timeseries.

Usage

```
growth(x, n = 1, keep_range = TRUE)
```

Arguments

x	a ts or regts object
n	an integer indicating the period of relative change
keep_range	if TRUE (the default), then the output timeseries has the same period range as the input timeseries. Then the result timeseries will have n NA values at the start. If FALSE then the result timeseries is n periods shorter than the input timeseries.

Value

a regts object with relative changes

See Also

rel2index

Examples

```
x <- regts(rnorm(10), start = "2018Q1")
growth(x, keep_range = FALSE)
growth(x, 4)
```

---

index_ts *Construct an index timeseries by scaling*

Description

This function scales a timeseries by dividing all observations by one selected observation or by the mean of a range of observations. The index series i is calculated from the input series x as

```
i[t] = scale * x[t] / mean(x[base]),
```

where scale is usually 100 and base the base period, which can be a single period or a period_range (by default the base period is the first period of x). If mean(x[base]) is negative then a warning is given and the (mean) value of the resulting index series at the base period will be -scale.

Usage

```
index_ts(x, base = NULL, scale = 100)
```

Arguments

x	a ts of regts object
base	a period or a period_range specifying the base period, or an object that can be coerced to a period or period_range. By default the base period is the first period of the input timeseries.
scale	the (average) value of the index series at the base period (by default 100). This should be a positive number.

See Also

rel2index and pct2index

Examples

```
ab <- regts(matrix(1:18 + rnorm(18), ncol = 2), start = "2016Q1",
            names = c("a", "b"))

index_ts(ab)

index_ts(ab, base = "2017", scale = 1)
```

---

is.period *Test if an object is a* period

Description

Test if an object is a period

Usage

```
is.period(x)
```

Arguments

x	any R object

Value

TRUE if the object is a period

Examples

```
p <- period("2016Q1")
is.period(p)
is.period("2016Q1")
```

is.period_range *Test if an object is a period_range*

Description

Test if an object is a period_range

Usage

```
is.period_range(x)
```

Arguments

x	any R object

Value

TRUE if the object is a period_range

Examples

```
range <- period_range("2016Q1/2017Q1")
is.period_range(range)
is.period_range("2016Q1/2017Q1")
```

is.regts *Test whether an object is a regts timeseries object*

Description

Test whether an object is a regts timeseries object

Usage

```
is.regts(x)
```

Arguments

x	any R object.

Value

TRUE if x is a regts

See Also

regts and as.regts

Examples

```
a <- regts(1:15, start = "2011Q2")
is.regts(a)
```

`join_ts` *Join timeseries object with different but overlapping period ranges*

Description

This function creates a new timeseries from two (partially) overlapping timeseries with the same frequency. All observations from the first timeseries are scaled in such a way that the overlapping observations from the two timeseries have the same value (on average). The second timeseries must contain the most recent data.

Usage

```
join_ts(old, new, method = c("mult", "add"))
```

Arguments

`old`	the first timeseries (a `regts` or `ts` object).
`new`	the second timeseries (a `regts` or `ts` object).
`method`	two different ways to join the timeseries: `mult` and `add`. By default the timeseries are joined multiplicatively.

Details

The period range of the result is the union of the period ranges of the first and second timeseries.

When the overlap period is determined, the trailing NA values of the old timeseries and the leading NA values of the new timeseries are ignored.

In case of multivariate regts only the common columns are joined. For each common timeseries a check is done whether an overlapping period exists (ignoring the NA values as described above). The non overlapping columns in both timeseries are added to the result. If both input timeseries are vectors (i.e. no column names), the result is also a vector.

Value

a `regts` object.

See Also

`regts` and `update_ts`

Examples

```
x1 <- regts((1:15)/10, start = "2016q1")
x2 <- regts(1:10, start = "2018q4")
res <- join_ts(x1, x2)

data <- (1:10)/10
x_old <- regts(cbind(data, 2 * data), period = "2001/2010",
            names = c("a", "b"))
x_new <- regts(cbind(10 * data, 20 * data), period = "2008/2017",
            names = c("a", "b"))
join_ts(x_old, x_new, method = "add")

# join timeseries with different column names
x_old <- regts(matrix(rep(10:15, 3), nc = 3), period = "2010/2015",
            names = c("a", "c", "d"))
x_new <- regts(matrix(rep(17:20, 3), nc = 3), period = "2014/2017",
            names = c("a", "b", "c"))
join_ts(x_old, x_new)
```

---

lag_ts *Lag a Timeseries*

Description

Compute the lag of a timeseries, shifting the observations forwards by a given number of periods.

Usage

```
lag_ts(x, n = 1, keep_range = TRUE, ...)

## S3 method for class 'ts'
lag_ts(x, n = 1, keep_range = TRUE, ...)
```

Arguments

x	a univariate or multivariate timeseries. Can also be a vector, matrix or data frame (see details).
n	the number of lags (in units of observations). Must be a positive number.
keep_range	if TRUE (the default), then the output timeseries has the same period range as the input timeseries. The result timeseries will have n NA values at the beginning. If FALSE the period range of the result timeseries is shifted by n periods. The result timeseries starts and ends n periods later.
...	further arguments to be passed to or from methods (currently not used in package regts)

Details

Vector, matrix and data frame arguments are first converted to a `regts` with function `regts`. This conversion results in a timeseries with frequency 1 and starting at year 1.

`lag_ts` differs from `lag` in the `stats` package in that the specified number of lags is positive, and that by default the resulting timeseries has the same period range as the input timeseries. In function `lag` the time base is always shifted. `lag_ts(x,1,keep_range = FALSE)` is the same as `lag(x,-1)`

See Also

`lead_ts`, `diff_ts` and `lag`

Examples

```
x <- regts(1:10, start = "2018q3")
lag_ts(x)
lag_ts(x, k = 2, keep_range = FALSE)
```

---

lead_ts *Lead a Timeseries*

Description

Compute the lead of a timeseries, shifting the observations backwards by a given number of periods.

Usage

```
lead_ts(x, n = 1, keep_range = TRUE, ...)

## S3 method for class 'ts'
lead_ts(x, n = 1, keep_range = TRUE, ...)

## Default S3 method:
lead_ts(x, n = 1, keep_range = TRUE, ...)
```

Arguments

`x`	a univariate or multivariate timeseries. Can also be a vector, matrix or data frame (see details).
`n`	the number of leads (in units of observations). Must be a positive number.
`keep_range`	if `TRUE` (the default), then the output timeseries has the same period range as the input timeseries. The result timeseries will have `n` `NA` values at the beginning end. If `FALSE` the period range of the result timeseries is shifted by `n` periods. The result timeseries starts and ends `n` periods earlier.
`...`	further arguments to be passed to or from methods (currently not used in package `regts`)

Details

Vector, matrix and data frame arguments are first converted to a `regts` with function `regts`. This conversion results in a timeseries with frequency 1 and starting at year 1.

Function `lead_ts` is an alternative for function `lag` in the `stats` package which computes both lags and leads. By default in `lead_ts` the resulting timeseries has the same period range as the input timeseries. In function `lag` the time base is always shifted. `lead_ts(x,1,keep_range = FALSE)` is the same as `lag(x,1)`

See Also

`lag_ts` and `diff_ts`

Examples

```
x <- regts(1:10, start = "2018q3")
lead_ts(x)
lead_ts(x, k = 2, keep_range = FALSE)
```

`movav` *Moving average of a timeseries*

Description

Function `movavb` computes the backward moving average and function `movavc` the centered moving average.

For example, the backward moving average of order 3 is defined as

`A[t] = (x[t -2] + x[t -1] + x[t]) / 3`,

while the centered moving average of order 3 is calculated as

`A[t] = (x[t -1] + x[t] + x[t + 1]) / 3`.

The calculation of the centered moving average for even orders is somewhat more complicated, see Details.

Usage

```
movavb(x, order, keep_range = TRUE)

movavc(x, order, keep_range = TRUE, method = c("centre", "left", "right"))
```

Arguments

`x`	a `ts` or `regts` object
`order`	the order of the moving average

keep_range — If TRUE (the default), then the output timeseries has the same period range as the input timeseries. Then the result timeseries will have order NA values. For movavb these NAs will appear on the left side and for movavc they will be distributed over both sides. If FALSE then the result timeseries is order periods shorter than the input timeseries.

method — method used to handle the centered moving average for even orders. Possible values are "centre" (the default), "left" and "right". See Details. This argument is ignored for odd orders.

Details

The centered moving average for even orders is usually computed by using one more observation than the order and to use weights 0.5 for the end points. For example, for order 4 we have

A[t] = (0.5 x[t -2] + x[t -1] + x[t] + x[t + 1] + 0.5 x[t + 2]) / 4.

In this way the observations are distributed evenly over the past and future. An alternative approach is to use the same number of observations as the order but use one more observation from the past than from the future, or the other way around. These methods can be used by specifying argument method. Possible methods are

centre Standard method e.g. (0.5 x[t - 2] + x[t - 1] + x[t] + x[t + 1] + 0.5 x[t + 2]) / 4

left Use one more observation from the past, e.g. (x[t - 2] + x[t - 1] + x[t] + x[t + 1]) / 4

right Use one more observation from the future, e.g. (x[t - 1] + x[t] + x[t + 1] + x[t + 2]) / 4

Value

a regts object with the moving average values

Functions

- movavb: Backward moving average
- movavc: Centered moving average

Examples

```
x <- regts(rnorm(10), start = "2018Q1")

movavb(x, order = 3)

movavc(x, order = 3, keep_range = FALSE)
```

na_trim *Function for removing leading and trailing NAs*

Description

This function removes leading or trailing NAs or both from a (multivariate) regts object. For multivariate regts a row will by default be regarded as NA if all elements in the row are NA. Use argument `is.na = "any"` to change this behaviour. The function returns `NULL` if all values are `NA`.

Usage

```
na_trim(x, method = c("both", "first", "last"), is_na = c("all", "any"))
```

Arguments

x	a regts object
method	character string with values `"both"`, `"first"` or `"last"` to remove NAs at both ends (by default), just at the start or just at the end.
is_na	character string with values `"all"` or `"any"`. If `"all"` (default) then a row will be regarded as `NA` only if all elements in the row are `NA`. If `"any"` then a row will be regarded as `NA` if it has any `NA`s. For one dimensional regts objects this argument has no effect.

Value

A regts object in which leading and/or trailing NAs have been removed, or `NULL` if all values are `NA`.

See Also

zero_trim

Examples

```
# remove only leading NAs
ts1 <- regts(c(NA,1,3,NA,4,8,NA), start = "2000")
na_trim(ts1, method = "first")

# remove trailing NAs
data <- matrix(c(1,3,NA,2,5,NA,3,7,NA), ncol = 3)
rts <- regts(data, start = "2010Q2", names = c("a", "b", "c"))
na_trim(rts, method = "last")

data <- matrix(c(NA,3,NA,NA,5,6,NA,7,NA), ncol = 3)
rts <- regts(data, start = "2010Q1", names = c("a", "b", "c"))
# remove leading NAs if all elements in the row are NA
na_trim(rts, method = "first")
# or remove rows on both sides if any NA occurs in that row
na_trim(rts, is_na = "any")
```

nperiod *Return the number of periods in a period range*

Description

Return the number of periods in a period range

Usage

```
nperiod(x)
```

Arguments

x	a period_range or an object that can be coerced to a period_range

Value

The number of periods in the range, or Inf if the range is not bounded

Examples

```
range <- period_range("2010Q2", "2011Q3")
nperiod(range)  # the result will be 6
```

period *Create a* period *object*

Description

Function period creates a period object based on a character or numeric vector. Possible character string formats are for example "2017Q2", "2017m2", "2017", "2017-2", "aug 2017" or "august-2017". Possible numeric formats are for example 2017 or 2017.25 (the second quarter or the fourth month of 2017). The function also accepts a Date, POSIXct or POSIXlt argument. See Details.

Function as.period coerces an R object to a period object if possible.

Usage

```
period(x, frequency = NA)

as.period(x, ...)
```

Arguments

x	a character, numeric, Date, POSIXct or POSIXlt vector.
frequency	frequency of the period. Argument frequency is mandatory if the frequency cannot be inferred from x (for example "2017-2" could be a quarter, month, etc.)
...	additional arguments to be passed to or from methods (currently not used in package regts)

Details

The function period accepts a character or numeric vector as arguments. The specific format is described below.

string format

The format for yearly periods is for example "2017" or "2017Y" (the suffix "Y" is optional).

The standard format format for quarterly periods is for example "2017Q3". Alternative formats such as "2017 3Q" and "2017.3Q" are also recognized. The separator between the year and the quarter can be a blank or a dot, as in the previous examples, but also a forward slash ("/") and underscore ("_") are allowed.

The format for monthly periods is similar as that of quarterly periods, except that the "Q" is replaced by "M". Monthly periods may also be specified with a month name (possibly abbreviated) and year (e.g. "aug 2017", "2018-August"). The parser only understands English month names.

Periods with other frequencies than year, quarter and month can be specified as for example "2017-2". Alternative separators (blank, dot, etc.) are possible. In this case argument frequency should be specified.

The string format is case insensitive, and may be prefixed with "Y" or "T". Thus for example "t2017q3" is also an allowed period string.

numeric format

An integer number, such as 2017 specifies a year, or the first subperiod in a year if argument frequency has been specified.

If the numeric has a non-zero fractional part, then argument frequency is mandatory, For example, the numeric 2017.25 can specify the second quarter of 2017 or the fourth month of 2017.

Date, POSIXct and POSIXlt

The function also accepts a Date, POSIXct or POSIXlt argument. By default the function converts this object to a period with frequency month. It is possible to specify another output frequency, provided that this frequency is a divisor of 12.

Value

a period vector if all periods have the same frequency, otherwise a list of period objects.

See Also

period_range and seq

Examples

```
period("2010Q3")
period("2010-2", frequency = 3)
period(2015)
period(2010.25, frequency = 4)

# examples for as.period
as.period("2010q3")
p <- period("2010m11")
as.period(p)

# example with a Date object
d <- Sys.Date()
period(d)

# create a vector of period objects
period(c("2018q2", "2019q4"))
```

---

period_range *Create a* period_range *object.*

Description

A period_range object represents an interval of periods, for example a period from "2017Q2" to "2019Q3". Function period_range creates a period_range from a single character string (e.g. "2017Q2/2019Q3", see Details) or from two R objects that can be coerced to period objects.

Function as.period_range coerces an R object to a period_range if possible.

Usage

```
period_range(start = NULL, end = NULL, frequency = NA)

as.period_range(x, frequency = NA, ...)
```

Arguments

start	the first period (a period, an object that can be coerced to a period, or by default NULL). If start is NULL the lower bound of the period range is undetermined. start can also be a character string specifying a period range, for example "2010Q2/2011Q3").
end	the last period (a period, an object that can be coerced to a period, or by default NULL). If end is NULL, the upper bound of the period range is undetermined.
frequency	frequency of the period objects. This argument is mandatory if argument start or end is a character with general period format without frequency indicator (e.g. "2011-1")
x	an R object
...	additional arguments to be passed to or from methods (currently not used in package regts)

Details

It is possible to create a period_range from a single string specifying a period range, for example "2017Q2/2019Q3". For this format, the first and last period are separated by "/". The first and last period are specified according to the same format recognized by function period. The first or last period may be omitted (e.g. "2017Q3/")), in that case the period range has no lower or upper bound. The string format is case insensitive.

Value

a period_range object

See Also

period, nperiod, start_period, end_period, seq and get_periods.

Examples

```
# two methods to create a period_range from 2010Q2 to 2016Q3
period_range("2010Q2", "2016Q3")
period_range("2010Q2/2016Q3")

# create a period_range for the first 5 quarters after 2013Q2
start <- period("2013q3")
period_range(start, start + 5)

# create a period_range up to 2010Q2 with no lower bound
period_range(end = "2010q2")

# create a period_range for a timeseries with frequency 2 (half year)
period_range("2010-2", "2016-2", frequency = 2)

# convert a period object to a period_range with equal start and end period
p <- period("2010Q2")
as.period_range(p, p)

# create a month range starting at the month 1000 days before
# the current day and ending at the current month.
today <- Sys.Date()
period_range(today - 1000, today)
```

---

printobj *Print the name, class and value of an object*

Description

This function prints the name, class and value of its argument. The value is printed using the standard print function. It returns the value of the argument invisibly.

Usage

```
printobj(x, ...)
```

Arguments

x	an R object
...	further arguments passed to print

Examples

```
x <- regts(1:5, start = "2017Q2")
printobj(x)
printobj(2 * x)
```

---

range_intersect/range_union

Calculate the intersection or union of two period_range *objects.*

Description

These functions calculate the intersection or union of two period_range objects. The start and end periods of the period_range objects may not be NULL.

Usage

```
range_intersect(range1, range2)

range_union(range1, range2)
```

Arguments

range1	a period_range object or object that can be coerced to a period_range
range2	another such (period_range) object

Value

the intersection or union of range1 and range2. If there are no common periods range_intersection returns NULL

Examples

```
range1 <- period_range("2016Q1", "2017Q4")
range2 <- period_range("2017Q1", "2018Q2")
range_intersect <- range_intersect(range1, range2)
range_union <- range_union(range1, range2)
```

read_ts_csv *Read timeseries from a csv file*

Description

This function reads timeseries from a csv file, employing function fread of package data.table. The functions searches for period texts and automatically determines how the timeseries are stored (rowwise or columnwise) and which columns contain the numerical values of the timeseries. Period texts should have the format recognized by function period, for example "2010Q2", "2010.2Q", "2010m2", "2011" or "2011-1". Use argument period_fun if the period texts have a different format.

Usage

```
read_ts_csv(filename, skiprow = 0, skipcol = 0, rowwise, frequency = NA,
  labels = c("after", "before", "no"), sep = "auto", fill = FALSE,
  dec = if (sep != ".") "." else ",", na_string = "", name_fun, period_fun,
  strict = TRUE)
```

Arguments

filename	a string with the filename.
skiprow	the number of rows to skip. If 0 (default) and if argument fill is FALSE, then comment rows are automatically skipped. See Details.
skipcol	the number of columns to skip.
rowwise	a logical value: are the timeseries stored rowwise? If not specified, then read_ts_csv tries to figure out itself if the timeseries are stored rowwise or columnwise.
frequency	the frequency of the timeseries. This argument is mandatory if the file contains a period texts without frequency indicator (for example "2011-1").
labels	label option. See Details.
sep	the separator between columns. If not specified, then the separator is determined automatically by inspecting the first 30 lines of the csv file (see the details of function fread).
fill	logical (default is FALSE). If TRUE then in case the rows have unequal length, blank fields are implicitly filled with NA.
dec	the decimal separator as in base::read.csv. If not "." (default) then usually ",".
na_string	Character vector of strings to use for missing values. By default, read_ts_csv treats blank cells as missing data.
name_fun	function to apply to the names of the timeseries.
period_fun	function applied to period texts. This should be a function that converts a character vector to another character vector or a period vector with the same length. Use this argument if the period texts do not have a standard format (see Description).

`strict` A logical. If `TRUE` (the default) all periods between the start and the end period must be present. Otherwise the timeseries are filled with `NA` for the missing periods.

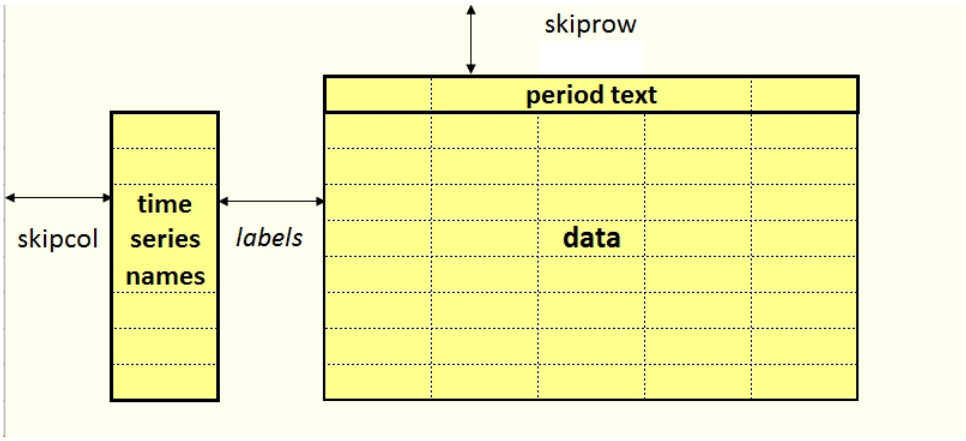
Details

In many cases, this function will read timeseries correctly. If the function fails or if the result is not what you want, it might help to specify arguments `rowwise`, `frequency`, `period_fun`, `skipcol` or `skiprow`. Specify option `rowwise` if you know that the timeseries are stored rowwise or columnwise. Specify argument `frequency` if you already know the frequency of the timeseries. Arguments `skipcol` and `skiprow` can be used to read only a part of the file. If that does not help, then you can read the data into a data frame (for example by using function `read.csv` or function `fread` of package `data.table`), then convert the data frame to a standard columnwise data frame and finally convert it to a `regts` by using function `as.regts`.

If argument `rowwise` has not been specified, then function `read_ts_csv` tries to guess if the timeseries are stored rowwise or columnwise based on the positions of the fields with period texts.

rowwise timeseries

For rowwise timeseries, the function searches for the first row with periods. All rows before the period row are ignored. Columns without a valid period in the period row are also ignored. The first non-empty column should contain the timeseries names (or labels if argument `labels = "before"`, see the discussion below). Otherwise use argument `skipcol` to specify the number of columns to skip.



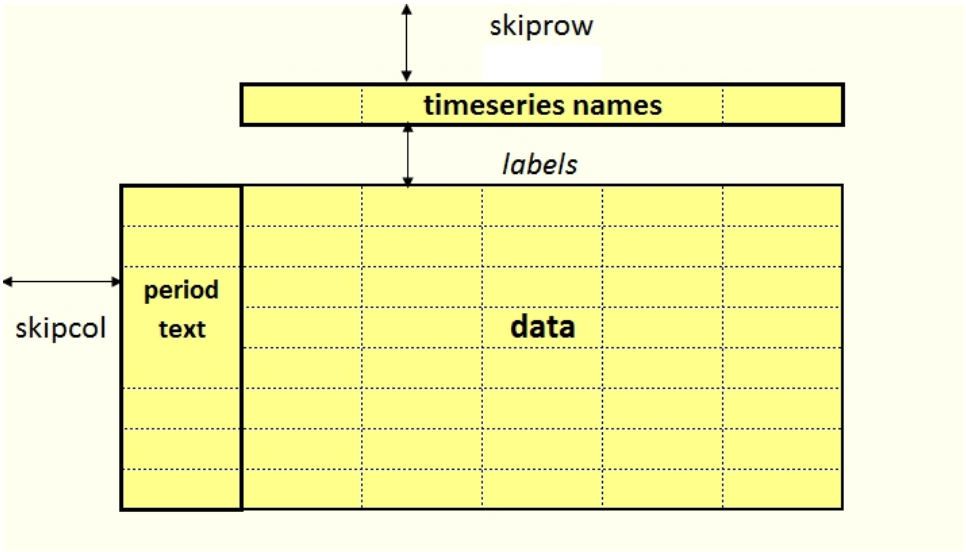
There may be more than one column before the columns with timeseries values (data columns). In that case one column should contain the variable names. The other columns before the first data column are used to create timeseries labels (see `ts_labels`). If argument `labels = "after"` (default), then the first column contains the variable names. If `labels = "no"` the first column also contains variable names but the other columns before the first data column are ignored. If argument `labels = "before"`, then the variable names should be in the last column before the first data column.

With argument `name_fun` a function can be applied to names of the timeseries, e.g. `tolower`.

columnwise timeseries

For columnwise timeseries, the first non-empty row that is not skipped (see argument `skiprow`) should contain the variable names (or labels if argument `labels = "before"`, see the discussion

below). The periods can be in any column. Rows without a valid period in the period column are ignored. All columns to the left of the period column are also ignored.



There may be more than one row before the rows with timeseries values (data rows). In that case one row should contain the variable names. The other rows before the first data row are used to create timeseries labels (see `ts_labels`). If argument `labels = "after"` (default), then the first row contains the variable names. If `labels = "no"` the first row also contains variable names but the other rows before the first data row are ignored. If argument `labels = "before"`, then the variable names should be in the last row before the first data row.

automatic row skip

If `skiprow = 0`, then the first rows with less columns than the rest of the file are automatically skipped. These rows are assumed to be comment rows. This procedure is described in detail in the documentation of function `fread` of the `data.table` package. Briefly, `fread` first determines the number of columns and then searches for the first data row based on this number of columns. All rows before this data row are skipped.

If argument `fill` is `TRUE`, then all rows have the same number of columns, and automatic row skipping is therefore disabled.

Value

a `regts` object

See Also

`write_ts_csv` and `read_ts_xlsx`

Examples

```
## Not run:
read_ts_csv("series.csv", sep = ";", dec = ",")
read_ts_csv("data.csv", labels = "after", name_fun = tolower)
```

```
## End(Not run)
```

`read_ts_xlsx` *Read timeseries from a sheet of an xls(x) file*

Description

This function reads timeseries from a sheet of an xls(x) file, employing function `read_excel` of package `readxl`. `read_ts_xlsx` searches for period cells and automatically determines how the timeseries are stored (rowwise or columnwise) and which columns contain the numerical values of the timeseries. Period cells are cells containing

- a text with the format recognized by function `period`, for example `"2010Q2"`, `"2010.2Q"`, `"2010m2"`, `"2011"` or `"2011-1"`,
- an integer value (e.g. `2018`), which is considered as a year,
- a date, which is assumed to specify a month unless argument `frequency` has been specified.

Use argument `period_fun` if the period cells contain a text with a format not recognized by function `period`.

Usage

```
read_ts_xlsx(filename, sheet = NULL, range = NULL, skiprow = 0,
  skipcol = 0, rowwise, frequency = NA, labels = c("after", "before",
  "no"), na_string = "", name_fun, period_fun, strict = TRUE,
  warn_num_text = TRUE)
```

Arguments

`filename`	a string with the filename.
`sheet`	Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet). Ignored if the sheet is specified via range. If neither argument specifies the sheet, defaults to the first sheet.
`range`	A cell range to read from, as described in `cell-specification`. Includes typical Excel ranges like `"B3:D87"`, possibly including the sheet name like `"Budget!B2:G14"`, and more. Takes precedence over `skiprow`, `skipcol` and `sheet`.
`skiprow`	the number of rows to skip, including leading empty rows. Ignored if `range` is given. By default, all leading empty rows are skipped.
`skipcol`	the number of columns to skip, including empty columns. Ignored if `range` is given. By default, all leading empty columns are skipped.
`rowwise`	a logical value: are the timeseries stored rowwise? If not specified, then `read_ts_xlsx` tries to figure out itself if the timeseries are stored rowwise or columnwise.

frequency	the frequency of the timeseries. This argument is mandatory if the file contains period texts without frequency indicator (for example "2011-1").
labels	label option. See Details.
na_string	Character vector of strings to use for missing values. By default, read_ts_xlsx treats blank cells as missing data.
name_fun	function to apply to the names of the timeseries.
period_fun	function applied to period texts. This should be a function that converts a character vector to another character vector or a period vector with the same length. Use this argument if the period texts do not have a standard format (see Description).
strict	A logical. If TRUE (the default) all periods between the start and the end period must be present. Otherwise the timeseries are filled with NA for the missing periods.
warn_num_text	A logical. If TRUE (the default) a warning is issued when a cell contains a number as text (e.g. "2012.2") when a numeric value is expected. The text is always converted to a numeric value assuming the decimal separator ".".

Details

read_ts_xlsx reads the timeseries data in two steps. In the first step, the first 25 rows are read to inspect the structure of the data on the sheet: are the timeseries stored rowwise or columnwise, which row or column contains the period cells and which columns contain the numerical data of the timeseries. Using this information, the complete sheet is read and the timeseries are constructed.

In many cases, this function will read timeseries correctly. If the function fails or if the result is not what you want, it might help to specify arguments rowwise, frequency, period_fun, range, skipcol or skiprow. Specify option rowwise if you know that the timeseries are stored rowwise or columnwise. Specify argument frequency if you already know the frequency of the timeseries. Arguments range, skipcol and skiprow can be used to read only a part of the file.

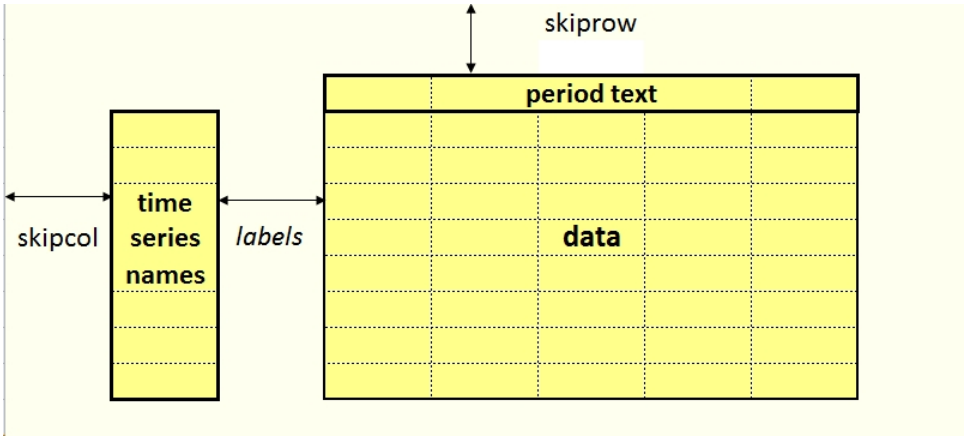
If that does not help, then you can read the data into a data frame (for example by using function read_excel of package readxl), then convert the data frame to a standard columnwise data frame and finally convert it to a regts by using function as.regts.

read_ts_xlsx skips all empty rows and columns. Use arguments skipcol and skiprow to skip additional leading rows and columns. Argument range can be used to read only a part of the sheet.

If argument rowwise has not been specified, then function read_ts_xlsx tries to guess if the timeseries are stored rowwise or columnwise based on the positions of the period cells.

rowwise timeseries

For rowwise timeseries, the function searches for the first row with periods. All rows before the period row are ignored. Columns without a valid period in the period row are also ignored. The first non-empty column in the sheet should contain the timeseries names (or labels if argument labels = "before", see the discussion below). Otherwise, use argument skipcol to specify the number of columns to skip.

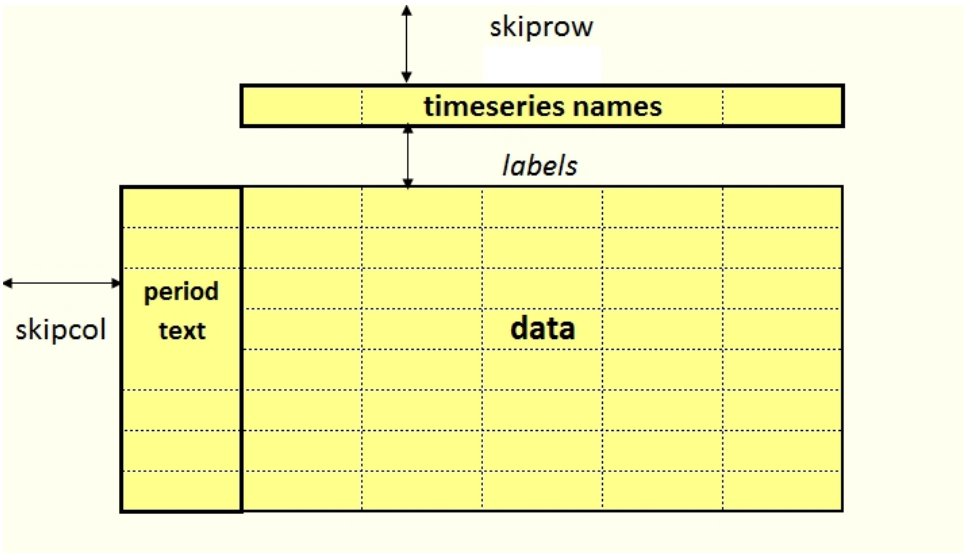


There may be more than one column before the columns with timeseries values (data columns). In that case one column should contain the variable names. The other columns before the first data column are used to create timeseries labels (see `ts_labels`). If argument `labels = "after"` (default), then the first column contains the variable names. If `labels = "no"` the first column also contains variable names but the other columns before the first data column are ignored. If argument `labels = "before"`, then the variable names should be in the last column before the first data column.

With argument `name_fun` a function can be applied to names of the timeseries, e.g. `tolower`.

columnwise timeseries

For columnwise timeseries, the first non-empty row that has been read (see argument `range` or `skiprow`) should contain the variable names (or labels if argument `labels = "before"`, see the discussion below). The periods can be in any column on the sheet. Rows without a valid period in the period column are ignored. All columns to the left of the period column are also ignored.



There may be more than one row before the rows with timeseries values (data rows). In that case one row should contain the variable names. The other rows before the first data row are used to create timeseries labels (see `ts_labels`). If argument `labels = "after"` (default), then the first

row contains the variable names. If `labels = "no"` the first row also contains variable names but the other rows before the first data row are ignored. If argument `labels = "before"`, then the variable names should be in the last row before the first data row.

Value

a `regts` object

See Also

`write_ts_xlsx` and `read_ts_csv`

Examples

```
## Not run:
read_ts_xlsx("series.xlsx", skipcol = 2, na_string = c("", "NA"))
read_ts_xlsx("data.xlsx", sheet = "Budget", labels = "after",
             name_fun = tolower)

## End(Not run)
```

---

regts *Create a* `regts` *timeseries object*

Description

The `regts` class is an extension of the `ts` class of the stats package. Working with `regts` makes it easier to select periods.

Usage

```
regts(data, start, end, period, frequency = NA, names = colnames(data),
  labels = NULL)
```

Arguments

`data`	a vector or matrix of the observed timeseries values. A `data.frame` will be coerced to a numeric matrix via `data.matrix`. (See also the description of the function `ts` of the `stats` package).
`start`	the starting period as a `period` object or a character string that can be converted to a `period` object. If not specified, then the start period is calculated from argument `end` and the dimension of `data`.
`end`	the end period as a `period` object or a character string that can be converted to a `period` object. If not specified, then the end period is calculated from argument `start` and the dimension of `data`.

period	the period range as a period_range object or a character string that can be converted to a period_range object. This argument replaces arguments start and end.
frequency	the frequency of the timeseries. This argument should only be specified if the start, end or period argument is specified with a general period format without period indicator, e.g. "2011-3".
names	a character vector with the column names for the series if data is a matrix or data frame. Defaults to the column names of data.
labels	a character vector of labels (descriptions of the timeseries)

Value

a regts object

See Also

The function is.regts can be used to test if an object is a regts.

The S3 generic as.regts can be used to coerce an R object to a regts. There are currently methods for ts and data.frame.

as.data.frame and as.list can be used to convert regts to a data.frame or a list.

Function cbind can be used to bind two or more timeseries objects and create a multivariate regts.

Information about the time period of the timeseries can be obtained with the functions get_period_range, start_period and end_period.

See also the description of the functions for handling labels (ts_labels and update_ts_labels).

Examples

```
# univariate timeseries
ts1 <- regts(1:10, start = "2010Q4")

# period selection
print(ts1["2011Q2/2011Q3"])

# multivariate timeseries
ts2 <- regts(matrix(1:9, ncol = 3), start = "2010Q4", names = c("a", "b", "c"))

# two equivalent ways to select a column in a multivariate ts
print(ts2$a)
print(ts2[, "a"])

# period selection in multivariate regts
print(ts2["2011Q2/2011Q3"])

# period and column selection in multivariate regts
print(ts2["2011Q2/2011Q3", "a"])

# two equivalent ways to add a column
ts2$d <- 2
```

```
ts2[ , "e"] <- 2

# multivariate timeseries with labels
ts3 <- regts(matrix(1:9, ncol = 3), start = "2010Q4", names = c("a", "b", "c"),
             labels = paste("Timeseries", c("a", "b", "c")))

# multivariate timeseries created with a period_range object
range <- period_range("2016Q1", "2017Q4")
ts4 <- regts(matrix(1:16, ncol = 2), period = range, names = c("a", "b"))

# create a half-yearly timeseries; because argument end is specified the
# length of the timeseries is smaller than the length of data (10).
ts5 <- regts(1:10, start = "2010-1", end = '2011-2', frequency = 2)
```

`rel2index/pct2index` *Calculate an index timeseries from a timeseries with relative or percentage changes.*

Description

Function `rel2index` is the inverse of function `growth`. The growth `x[t]` (also called the relative change) of a timeseries `z[t]` is defined as

`x[t] = (z[t] - z[t - 1]) / |z[t - 1]|.`

The function constructs an index series for `z[t]` given the values of `x[t]`, assuming that the value of timeseries `z` at the period before the start period of timeseries `x` is positive. See Details.

Function `pct2index` computes the index series from a timeseries of percentage changes, defined as `100 * g[t]`. Thus expression `pct2index(x)` gives the same result as `rel2index(x / 100)`.

Usage

```
rel2index(x, base = NULL, scale = 100, keep_range = TRUE)

pct2index(x, base = NULL, scale = 100, keep_range = TRUE)
```

Arguments

`x`	a `ts` or `regts` (can also be a multivariate timeseries) with the relative of percentage changes.
`base`	a `period` or a `period_range` specifying the base period, or an object that can be coerced to a `period` or `period_range`. By default the base period is the period before the first period of the input timeseries `x`. For example, if `x` starts at 2018q1, then the default base period is 2017q4. If the base period is a `period_range`, then the average value of the index series will be equal to `scale`.
`scale`	the (average) value of the index series at the base period (by default 100)
`keep_range`	if `TRUE` (the default), then the output timeseries has the same period range as the input timeseries. If `FALSE` then the result timeseries starts 1 period earlier.

Details

If x[t] is given but z[t] is unknown, we can compute z[t] as

z[t] = z[t - 1] * (1 + sign(z[t - 1]) x[t]).

Given an initial value for z at some period (say z[0]), the equation above can be used repeatedly to calculate the values of z[t] for t > 0. In function rel2index z[0] is not known, but if we assume that it is positive, then this value is not needed if we calculate the index series defined as

i[t] = scale * z[t] / mean(z[base]),

where base is the base period. The index series i is independent of the absolute value of z[0], but does depend on the sign of z[0]. If z[0] is actually negative then the results of rel2index and pct2index are not correct.

If mean(x[base]) is negative then a warning is given and the (mean) value of the resulting index series in the base period will be -scale.

Functions

- rel2index: Calculates an index timeseries from a timeseries with relative changes
- pct2index: Calculates an index timeseries from a timeseries with percentage changes

See Also

index_ts and growth

Examples

```
ts1 <- regts(abs(rnorm(10)), start = "2010Q2")
print(rel2index(ts1))
print(rel2index(ts1, base = "2010Q3", scale = 1, keep_range = TRUE))
```

---

remove_na_columns *Function for removing columns with NAs in a (multivariate) regts*

Description

This function removes columns with NA values from a (multivariate) regts. A column will by default be regarded as NA if all elements in the column are NA. Specify argument is.na = "any" to change this behaviour. If all columns are removed the function returns NULL.

Usage

```
remove_na_columns(x, is_na = c("all", "any"))
```

Arguments

x	a regts object
is_na	character with values "all" or "any". If "all" (default) then a column will be regarded as NA only if all elements in the column are NA. If "any" then a column will be regarded as NA if it has any NAs.

Value

A regts object in which NA columns have been removed.

Examples

```
# remove columns with all NAs
data <- matrix(c(1,3, 5, NA, NA, NA,3,7,9), ncol = 3)
rts <- regts(data, start = "2010Q2", names = c("a", "b", "c"))
remove_na_columns(rts)

data <- matrix(c(NA,3,NA,NA,5,6,NA,7,9), ncol = 3)
rts <- regts(data, start = "2010Q1", names = c("a", "b", "c"))
remove_na_columns(rts, is_na = "any")
```

---

select_columns *Select columns using a regular expression*

Description

This function selects columns of an R object with column names (for example a data.frame, matrix, ts or regts). The columns with names matching a given regular expression are selected. This function employs base R function grep

Usage

```
select_columns(x, regex, drop = TRUE, ...)
```

Arguments

x	an R object with column names (e.g. a data.frame, matrix, ts or regts)
regex	a regular expression used to select a column
drop	if TRUE, the result is coerced to a vector if possible
...	arguments passed to function grep

Value

the column selection of object x

Examples

```
data <- regts(matrix(1:20, ncol = 4), start = "2010Q2",
              names = c("nlc", "ukc", "nly", "uky"))

# select all columns with names starting with nl
nl_data <- select_columns(data, "^nl")

# select all columns except column "nlc"
no_nlc <- select_columns(data, "^nlc$", invert = TRUE)
```

---

seq *Generates a sequence of periods*

Description

Generates a regular sequence of `period` objects.

Usage

```
## S3 method for class 'period'
seq(from, to, by, length.out, ...)
```

Arguments

`from`	a `period` object specifying the first period of the sequence.
`to`	a `period` object (or an object that can be coerced to a `period` object) specifying the last period of the sequence
`by`	an integer number, the increment of the sequence (the number of periods between each period in the sequence).
`length.out`	the desired length of the sequence. A non-negative number. If both `from` and `to` have been specified, and if `length.out` > 1, then the number of periods between `from` and `to` should be divisible by `length.out` -1.
`...`	arguments passed to or from methods (not used)

Value

a `period` vector

See Also

`period`, `period_range` and `get_periods`

Examples

```
p1 <- period("2018q2")
seq(p1, length.out = 4)
seq(p1, "2019q4")
seq(p1, "2019q4", by = 2)

# use seq in a for loop
ts <- regts(1:10, start = "2018q1")
seqp <- seq(period("2018q1"), "2020q1", by = 4)

# print first quarters ts
for (prd in as.list(seqp))
  {print(ts[prd])
}

# Note that we do not loop directly over the period vector, but first convert
# the vector to a list. Otherwise the \code{period} class is lost.
```

`start_period/end_period` *Return the start or end period of a timeseries object or a* `period_range`

Description

This function returns the start or end period of a timeseries object (a `regts` or `ts`) or a `period_range`.

Usage

```
start_period(x)

end_period(x)

## S3 method for class 'period_range'
start_period(x)

## S3 method for class 'ts'
start_period(x)

## S3 method for class 'period_range'
end_period(x)

## S3 method for class 'ts'
end_period(x)
```

Arguments

`x`	a `regts`, `ts` or `period_range` object.

Value

A `period` object representing the first or last period of the range. The return value can be `NULL` if argument `x` is a `period_range` with no lower or upper boundary.

Examples

```
# start and end period of a range
range <- period_range("2010Q4", "2011Q3")
start_period(range)
end_period(range)

# start and end period of a regts
data <- regts(matrix(1:20, ncol = 2), start = "2010Q2", names = c("nl", "uk"))
start_period(data)
end_period(data)
```

`topleft` *Return the topleft part of a* `regts`.

Description

This function returns the topleft part of a `regts`. By default the first 10 columns for the first 6 periods.

Usage

```
topleft(x, n = 6L, ncol = 10L)
```

Arguments

`x`	a multivariate `regts` or `ts`
`n`	a single integer. Length period for the resulting object.
`ncol`	a single integer. Number of columns in `regts`. By default only the first 10 columns are printed.

See Also

`head`, `tail`

Examples

```
data <- regts(matrix(rnorm(200), ncol = 20), start = "2010Q2",
              names = paste0("abc", 1:20))
topleft(data)
```

transpose_df *Transpose a* data.frame

Description

The function transposes a data.frame. If the input data frame has column labels (i. e., column vectors with an attribute "label"), then the first column of the returned data frame will contain the labels. With argument label_column you can specify a column that will be used to create column labels for the output data frame. These labels are visible in de data viewer.

Usage

```
transpose_df(x, colname_column, label_column)
```

Arguments

x	a data frame
colname_column	the name or the index of the column that contains the column names of the transposed data frame. By default the row names of the original data frame are used as column names of the new data frame.
label_column	a numeric or character vector with the indices or the names of the columns that contains the row labels. If this is a vector of length larger than 1, then the texts in the columns are combined to create single labels for the columns of the transposed data frame. By default labels are ignored.

Value

the transposed data frame

Examples

```
df <- data.frame(variables = c("a", "b"),
                 labels = c("Variabele a", "Variable b"),
                 x = 1:2, y = 10:11)
df_t <- transpose_df(df, colname_column = 1, label_column = 2)
print(df_t)
print(transpose_df(df_t))
```

tsdif *Calculate the differences between two multivariate timeseries objects*

Description

This function can be used to compare two multivariate timeseries objects. The result is a list with a regts component with the computed differences or NULL if there are no differences. The function returns a list with the differences, the names of columns with differences larger than a specified tolerance, and the names of the columns present in one object but missing in the other object. The return value also includes differences in the period ranges.

Usage

```
tsdif(x1, x2, tol = 0, fun = function(x1, x2) (x1 - x2))
```

Arguments

x1	the first timeseries (a multivariate regts or ts object).
x2	the second timeseries (a multivariate regts or ts object).
tol	difference tolerance (by default zero). Differences with absolute values smaller than or equal to tol are ignored.
fun	function to compute differences. This function should accept two arguments (two numbers) for which the difference is computed. By default the normal difference ($x_1 - x_2$) is computed. A useful function for computing differences is cvgdif, which computes relative differences if the absolute value of x2 is larger than 1.

Details

This function calculates the difference between common columns of two multivariate timeseries objects x1 and x2. The two timeseries must have the same frequency, but may have a different period range. The difference is computed for the intersection of the two period ranges. Two NA or two NaN values are considered to be equal. A NA value is not equal to a NaN value. The function also returns missing column names in one of the two objects,

Value

a list with class "tsdif", with the following components

equal	TRUE if x1 and x2 have the same column names and period ranges, and if all differences are smaller than or equal to tol
difnames	The names of the timeseries with differences larger than tol
dif	A regts with the computed differences, or NULL if there are no differences larger than tol. Only timeseries with differences larger than tol are included. Leading and trailing rows with differences less than tol have also been removed.
common_names	the names of the common columns

missing_names1	The names of columns present in x2 but missing in x1
missing_names2	The names of columns present in x1 but missing in x2
period_range1	The period ranges of x1 as a period_range object
period_range2	The period ranges of x2 as a period_range object
common_range	The intersection of the period ranges
ranges_equal	A logical indicating whether the period ranges of x1 and x2 differ
ts_names	a character string giving the names of the two input timeseries
tol	The tolerance parameter
fun	a character string specifying the supplied function fun, or NULL if fun has not been specified

See Also

regts

Examples

```
# create two timeseries x1 and x2
x1 <- regts(matrix(data = rep(1:27), nc = 3), start = "2008Q4",
           names = c("a", "b", "c"))
x2 <- x1 + 0.001
colnames(x2) <- c("a", "b", "d")

# calculate and print the differences
dif1 <- tsdif(x1, x2)
print(dif1)

# use the function cvgdif (convergence difference)
dif2 <- tsdif(x1, x2, fun = cvgdif)

# calculate differences with tol = 1e-4, and print the
# names of the timeseries with differences larger than tol
dif3 <- tsdif(x1, x2, tol = 1e-4, fun = cvgdif)
print(dif3$difnames)
```

---

ts_labels *Timeseries labels*

Description

Retrieve or set labels for the timeseries. Timeseries labels can be used to give a description of the contents of the timeseries.

Usage

```
ts_labels(x)

ts_labels(x) <- value
```

Arguments

x	a regts
value	a character vector with the labels or NULL. The length should be equal to the number of columns. Specify NULL to remove all labels.

Value

The retrieval function ts_labels() returns a named character vector: the names are the timeseries names (the column names) and the values the corresponding labels. The replacement method returns a regts object with labels.

Functions

- ts_labels: Retrieve timeseries labels
- ts_labels<-: Sets the timeseries labels

See Also

regts, update_ts_labels

Examples

```
ts <- regts(matrix(1:6, ncol = 2), start = "2016Q2", names = c("a", "b"))
ts_labels(ts) <- c("Timeseries a", "Timeseries b")
print(ts_labels(ts))

# print the column names and labels as a nice data.frame
print(as.data.frame(ts_labels(ts)))
```

---

update_ts *Update a timeseries with another timeseries object*

Description

This function can be used to update, replace or extend a (reg)ts object with another (reg)ts object. The result is an updated regts object.

Usage

```
update_ts(x1, x2, method = c("upd", "updna", "updval", "replace"))
```

Arguments

x1	the first timeseries (a regts or ts object).
x2	the second timeseries (a regts or ts object).
method	four different ways to update the timeseries. By default the timeseries are updated. This behaviour can be changed by using one of the other methods. See details.

Details

The two timeseries must have the same frequency, but may have a different period range. The common columns in the timeseries can be updated in four different ways:

upd the first timeseries are updated with the second timeseries for the total period range of the second timeseries. Outside this period the values in the first timeseries do not change.

updna if method updna is selected instead of upd, only NA values in the first timeseries will be updated

updval if method updval is selected instead of upd, the values in the first timeseries are only replaced with valid (i.e. non-NA) values from the second timeseries.

replace like method upd, the values in the first timeseries are replaced by the values in the second timeseries for the total period range of these second timeseries. Outside this period the values in the first timeseries will become NA.

The non overlapping columns in both timeseries are added to the result.

The period range of the result is the union of the period ranges of the first and second timeseries, except for the updval method. For this method the result period range is the union of the period ranges of the first timeseries and the timeseries obtained by applying function na_trim to the second timeseries.

Value

an updated regts object.

See Also

regts

Examples

```
x1 <- regts(matrix(data = rep(1:9), nc = 3), period = "2000/2002",
            names = c("a", "b", "c"))
x2 <- regts(matrix(data = rep(10:15), nc = 3), period = "2000/2001",
            names = c("a", "c", "d"))
update_ts(x1, x2, method = "upd")
```

update_ts_labels *Update one or more timeseries labels in a multivariate* regts *object*

Description

Update one or more timeseries labels in a multivariate regts object

Usage

```
update_ts_labels(x, labels)
```

Arguments

x	a multivariate regts object
labels	a named character vector. The names are the names of the timeseries (columns) whose label will be updated. Specify NULL to remove all labels.

See Also

ts_labels

Examples

```
ts <- regts(matrix(1:6, ncol = 2), start = "2016Q2", names = c("a", "b"),
            labels = c("Timeseries a", "???"))
ts <- update_ts_labels(ts, c(b = "Timeseries b"))
print(ts_labels(ts))
```

write_ts_csv *Write timeseries to a csv file*

Description

This function writes timeseries to a csv file. The csv file is actually written by function fwrite of package data.table.

Usage

```
write_ts_csv(x, file, rowwise = TRUE, sep = ",", dec = ".",
  labels = c("after", "before", "no"), period_format = "regts")
```

Arguments

x	a ts or regts object
file	a regts object
rowwise	a logical value: should the timeseries be written rowwise?
sep	The separator between columns. Default is ",".
dec	The decimal separator, by default ".". Cannot be the same as sep.
labels	should labels we written, and if so before the names or after the names? By default, labels are written after the names if present.
period_format	The period format. By default the regts format (e.g. "2010Q2", see period) is used. Alternatively, it is possible to specify a format employed by base R function strptime, e.g. "%Y-%m-%d".

See Also

read_ts_csv and write_ts_xlsx

Examples

```
# create a timeseries object
ts1 <- regts(matrix(rnorm(50), ncol =  2), names = c("a", "b"),
             labels = c("Timeseries a", "Timeseries b"), start = "2017Q2")

# write timeseries to csv
write_ts_csv(ts1, file = "ts1.csv", labels = "after")

# write timeseries columnwise to csv, using a specified period_format
write_ts_csv(ts1, file = "ts1_2.csv", rowwise = FALSE, period_format = "%Y-%m-%d")
```

---

write_ts_xlsx/write_ts_sheet

Functions for writing timeseries to an xlsx file

Description

These functions can be used to write timeseries to a sheet of an xlsx file. write_ts_xlsx creates or opens an Excel workbook (depending on argument append) and writes the timeseries to a sheet with a specified name. write_ts_sheet writes timeseries to a sheet of a Workbook object created with function createWorkbook or loadWorkbook of package openxlsx.

Usage

```
write_ts_xlsx(x, file, sheet_name = "Sheet1", rowwise = TRUE,
  append = FALSE, labels = c("after", "before", "no"), comments,
  number_format, period_as_date = FALSE)

write_ts_sheet(x, wb, sheet_name = "Sheet1", rowwise = TRUE,
  labels = c("after", "before", "no"), comments, number_format,
  period_as_date = FALSE)
```

Arguments

x	a ts or regts object
file	the filename of the output file
sheet_name	the sheet name
rowwise	a logical value: should the timeseries be written rowwise?
append	If FALSE (the default), then the original file, if it exists, is replaced with the new file. All original data is lost. If TRUE, then only data on the sheet with the specified sheet name is erased and replaced with new data. If the sheet does not yet exist, then a new sheet is created and appended to the original file.
labels	should labels be written, and if so before or after the names? By default, labels are written after the names if present
comments	a character vector or data frame. The comments are written to the beginning of the sheet, before the timeseries data is written.
number_format	a character value specifying the number format. For example, "#.00" corresponds to two decimal spaces. For details see the description of the function createStyle in the openxlsx package.
period_as_date	A logical (default FALSE). If TRUE the periods are written as date values to the Excel file. By default the periods are written as characters using the standard regts format (e.g. "2010Q2", see period).
wb	a Workbook object created with function createWorkbook or loadWorkbook of package openxlsx

Details

The functions employ package openxlsx package for writing the Excel file.

If you want to write multiple timeseries objects to different sheets, you can use write_ts_xlsx with argument append = TRUE. Alternatively, you can create a Workbook object with function createWorkbook of package openxlsx and then add a sheet with write_ts_sheet. The latter approach is more efficient. When the workbook is written to a file with function saveWorkbook, it is often useful to set the minimum column width option for package openxlsx, as shown in the example below.

Functions

- write_ts_xlsx: writes timeseries to an Excel workbook
- write_ts_sheet: writes a timeseries to a Workbook object

See Also

read_ts_xlsx and write_ts_csv

Examples

```
# create a timeseries object
ts1 <- regts(matrix(rnorm(50), ncol =  2), names = c("a", "b"),
             labels = c("Timeseries a", "Timeseries b"), start = "2017Q2")

# write timeseries ts1 to an Excel file
write_ts_xlsx(ts1, file = "ts1.xlsx", sheet_name = "ts1", labels = "after")

# write two sheets using write_ts_sheet
library(openxlsx)
wb <- createWorkbook()
write_ts_sheet(ts1, wb, "ts1", labels = "after")
write_ts_sheet(ts1 * 100, wb, "ts1_times_100", labels = "after")

# Set the minimum column width. saveWorkbook will adjust
# the column widths for the sheets written by write_ts_xlsx,
# Setting a minimum column width prevents that some columns are very
# narrow.
options("openxlsx.minWidth" = 8.43)

saveWorkbook(wb, "timeseries.xlsx", overwrite = TRUE)

# write a timeseries with comments
comments <- c("Timeseries ts1 is created on the Central Bureau of Policy Analysis",
              "using a random number generator")
write_ts_xlsx(ts1, file = "ts_comments.xlsx", sheet_name = "ts1",
              comments = comments)
```

---

zero_trim *Function for removing leading and trailing zeros*

Description

This function removes leading or trailing zeros or both from a (multivariate) regts object. For multivariate regts a row will by default be regarded as 0 if all elements in the row are 0. The function returns NULL if all values are zero.

Usage

```
zero_trim(x, method = c("both", "first", "last"))
```

Arguments

x	a regts object
method	character string with values "both", "first" or "last" to remove zeros at both ends (by default), just at the start or just at the end.

Value

A regts object in which leading and/or trailing zeros have been removed, or NULL if all values in the timeseries are zero.

Examples

```
# remove only leading zeros
ts1 <- regts(c(0, 1, 3, 0, 4, 8, 0), start = "2000")
zero_trim(ts1, method = "first")

# remove trailing zeros
data <- matrix(c(1, 3, 0,2, 5, 0, 3, 7, 0), ncol = 3)
rts <- regts(data, start = "2010Q2", names = c("a", "b", "c"))
zero_trim(rts, method = "last")

# removing zeros in a multivariate regts
data <- matrix(c(0, 3, 0, 0, 5, 6, 0, 7, 0), ncol = 3)
rts <- regts(data, start = "2010Q1", names = c("a", "b", "c"))
# remove leading zeros if all elements in the row are zero
zero_trim(rts, method = "first")
```

Index