Introduction to package regts

Rob van Harrevelt and Anita van der Roest

2017-11-27

Contents

1 Introduction

When you start looking in R for timeseries you easily find several classes and packages, class `ts` from the `stats` package and `zoo` being mentioned most.

The function `ts` creates a regular timeseries object with class `ts`. This can be a univariate or multivariate object. It has a start, an end, a frequency and you can print, plot, extend and update it. But class `ts` has no easy way to deal with fixed periods.

`Regts` is a package defined for regular timeseries objects built on class `ts`, but with features for easier selecting periods. The `regts` package defines a class `regts` that is an extension to the `ts` class. Because of this the `regts` class benefits of all the functionality of `ts`. The period handling is much easier than in the `ts` class, particularly for monthly, quarterly and annual timeseries.

Package `xts` also facilitates period selection, but concentrates more on daily (or weekly) based data.

Package `regts` provides some more extensions to package `ts`:

- functions for easily reading and writing timeseries to Excel and csv
- the use of labels, a description of the timeseries
- an easy way to convert a data frame to a timeseries object
- a function to calculate differences between multiple timeseries
- and furthermore a special set of aggregation functions for timeseries representing growth rates

To use this package `regts` load it with:

```
> library(regts)
```

2 Creation regts

2.1 Single timeseries

A single `regts` timeseries is created with the function `regts`. An example for a quarterly timeseries:

```
> tq <- regts(1:10, start = "2016Q3")
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016                1    2
2017    3    4    5    6
2018    7    8    9   10
```

The end period of the timeseries is based on the length of the input data. The period can also be a month, like `"2016M3"`, or a year:

```
> ty <- regts(c(.11, .29, .18, .24), start = "2017")
```

```
       +1   +2   +3
2017 0.11 0.29 0.18 0.24
```

Note that annual timeseries are printed differently than quarterly timeseries.

The `class` of `tq` and `ty` is `regts`, and because of the inheritance, also `ts`.

```
> class(tq)
```

```
[1] "regts" "ts"
```

If the input data is longer than you require you can make a selection with the `end` argument:

```
> ts_data <- regts(1:10, start = "2016Q1", end = "2016Q4")
```

If the input data is shorter than the period determined by the `start` and the `end` argument, the data is repeated:

```
> ts_data <- regts(1:2, start = "2016Q1", end = "2017Q1")
> ts_data
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    1    2    1    2
2017    1
```

As you can see the length of the period doesn't have to be a multiple of the length of the input data.

Class `regts` recognizes special formats for quarterly (`2016Q1`) and monthly (`2016M1`) timeseries. To create timeseries with a different frequency, specify the frequency argument:

```
> ts_data <- regts(1:10, start = "2016-1", end = "2017-1", frequency = 2)
```

Instead of arguments `start` and `end` you can also specify argument `period`:

```
> ts_data <- regts(1:10, period = "2016Q1/2016Q4")
```

Input for this period may be a character string as shown above, but also a variable of type `period_range`. This type will be explained in section 10.1.

2.2 Multivariate timeseries

Sometimes it is more convenient to combine several timeseries with the same period in a single object: a multivariate timeseries. A multivariate timeseries is just a matrix with two or more columns. multivariate `regts` is such a matrix with some extra attributes.

```
> rts <- regts(matrix(1:9, ncol = 3), start = "2016Q1", names = c("a", "b", "c"))
> rts
```

```
       a b c
2016Q1 1 4 7
2016Q2 2 5 8
2016Q3 3 6 9
```

Argument `names` can be used to set the column names. If omitted then the column names of the input matrix are used. In the next example the input matrix has no column names:

```
> rts1 <- regts(matrix(1:9, ncol = 3), start = "2016Q1")
> rts1
```

```
       [,1] [,2] [,3]
2016Q1    1    4    7
2016Q2    2    5    8
2016Q3    3    6    9
```

A multivariate timeseries inherits the classes `mts` (multivariate `ts`), `ts` and `matrix`:

```
> class(rts)
```

```
[1] "regts"  "mts"    "ts"     "matrix"
```

Timeseries names are preferably valid R names (only include a-z, A-Z, _, and 0-9 and start with a letter) and well chosen. In general they will be brief. If more documentation is needed for a timeseries than just this (short) name, optionally labels can be added (see also section labels).

2.2.1 Column selection and creation

Column selection for regts is the same as for matrices.

```
> a1_ts <- rts[ , "a"]
> a1_ts
```

```
     Qtr1 Qtr2 Qtr3
2016    1    2    3
```

```
> ac_ts <- rts[ , c("a", "c")]
> ac_ts
```

```
       a c
2016Q1 1 7
2016Q2 2 8
2016Q3 3 9
```

`a1_ts` is now a univariate timeseries, `ac_ts` a multivariate subset of `rts`.

For the `regts` class we can also create new columns:

```
> rts[, "x"] <- 2 * rts[, "a"]  # creating a new column "x"
> rts
```

```
       a b c x
2016Q1 1 4 7 2
2016Q2 2 5 8 4
2016Q3 3 6 9 6
```

This is not possible for classes `ts` and `matrix`.

2.2.2 Selecting columns with a regular expression

The `regts` package contains a function `select_columns` to select columns of an R object with column names. Besides `regts` this can be a `matrix` or a `data frame`. The columns with names matching a given regular expression are selected. A few examples:

```
> rts <- regts(matrix(1:8, ncol = 4), start = "2016Q1", names = c("a1", "b1", "a2", "b2"))
> select_columns(rts, regex = "a.*")  # all columns with names starting with "a"
```

```
       a1 a2
2016Q1  1  5
2016Q2  2  6
```

```
> select_columns(rts, regex = ".+1")  # all columns with names ending with 1
```

```
       a1 b1
2016Q1  1  3
2016Q2  2  4
```

The syntax of `regular expression` patterns is described in the R Documentation.

2.3 Matrices with one column

If a `regts` is created from a matrix with only one column, the timeseries is single (or univariate). So there are two types of single timeseries: one that is based on a vector and one that is based on a one-column matrix:

```
> ts      <- regts(1:8, start = "2016Q1")
> ts_1col <- regts(matrix(1:8, ncol = 1), start = "2016Q1")
```

They have identical output:

```
> ts
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    1    2    3    4
2017    5    6    7    8
```

```
> ts_1col
```

```
       [,1]
2016Q1    1
2016Q2    2
2016Q3    3
2016Q4    4
2017Q1    5
2017Q2    6
2017Q3    7
2017Q4    8
```

and identical classes:

```
> class(ts)
```

```
[1] "regts" "ts"
```

```
> class(ts_1col)
```

```
[1] "regts" "ts"
```

but different underlying data.

Another difference between the 'vector' version and the 'matrix-1-column' version is that the latter can have a column name:

```
> ts_1col <- regts(matrix(1:8, ncol = 1), start = "2016Q1", names = "a")
```

The column name is not printed with the output, but can be shown with the `View` or the `colnames` function:

```
> colnames(ts_1col)
```

```
[1] "a"
```

Single 'vector' timeseries never have a column name.

A single 'matrix-1-column' timeseries is also created when a single column is selected from a multivariate timeseries with the argument `drop = FALSE`. In this case the column name is preserved. If this argument is omitted (or `drop = TRUE` is used) a single 'vector' timeseries is created. A possible column name disapppears in this case. An example:

```
> rts <- regts(matrix(1:8, ncol = 2), start = "2016Q1", names = c("a", "b"))
>
> rts[ ,1, drop = FALSE] # result is a one-dimensional matrix with column name "a"
> rts[ ,2]               # result is a vector, column name "b" is lost
```

3 Period selection

The package `regts` is specially designed for the selection of periods. Where for a `ts` timeseries you have to use the `window` function:

```
> ts <- ts(c(1,2,3,4,5,6,7,8), start=c(2016,1), end=c(2017,4), freq=4)
> window(ts, start = c(2016,4), end = c(2017,2))
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016                   4
2017    5    6
```

with a `regts` you can select periods with the selection operator `[]`. Period selection is a form of subsetting.

```
> ts <- regts(1:8, start = "2016Q1", end = "2017Q4")
> ts["2016Q4/2017Q2"]
```

Besides a range you can also select one period:

```
> ts["2017Q1"]
```

or a range with an open beginning or end:

```
> ts["/2017Q2"]
> ts["2017Q1/"]
```

or a range with a lower frequency:

```
> ts["2017"]
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2017    5    6    7    8
```

If a selection is specified outside the defined range, the timeseries is filled with NA values for that range:

```
> ts["/2018Q2"]
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    1    2    3    4
2017    5    6    7    8
2018   NA   NA
```

```
> ts["2015Q3/"]
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2015             NA   NA
2016    1    2    3    4
2017    5    6    7    8
```

Period selection can also be applied on the left hand side of an assignment, in the target timeseries. The object must exist before applying selection.

```
> rts1 <- regts(1:8, start = "2010Q1", end = "2011Q4")
> rts1["2011Q2"] <- 2
> rts1["/2010Q3"] <- 99
> rts1
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2010   99   99   99    4
2011    5    2    7    8
```

The period of the timeseries can also be extended:

```
> rts1["2009Q1/2009Q2"] <- rts1["2010Q3/2010Q4"]
> rts1
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2009   99    4   NA   NA
2010   99   99   99    4
2011    5    2    7    8
```

The missing quarters (2009Q3 and 2009Q4) are filled in with NA values.

Period selection for multivariate regts

Selecting a period for a multivariate regts is quite similar. You can select one or more periods, or a combination of period(s) and columns:

```
> rts <- regts(matrix(1:12, ncol = 3), start = "2011Q1", names = c("a", "b", "c"))
> rts["2011Q2", ]
```

```
       a b  c
2011Q2 2 6 10
```

```
> rts["2011Q2"]                # the , is not necessary
```

```
       a b  c
2011Q2 2 6 10
```

```
> rts["2011Q1/2012Q2", "c"]      # extended selection
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2011    9   10   11   12
2012   NA   NA
```

All the extended elements become NA.

Select a combination of period and columns:

```
> rts["2011Q4/", c("a", "b")]
```

```
       a b
2011Q4 4 8
```

Selection can also take place in the target timeseries:

```
> rts[, "x"] <- NA              # define an extra column
> rts["2012Q1", ] <- 99         # extend period for all columns
> rts["2011", "x"] <- 1         # update all quarters in 2011 in column "x"
```

```
        a  b  c  x
2011Q1  1  5  9  1
2011Q2  2  6 10  1
2011Q3  3  7 11  1
2011Q4  4  8 12  1
2012Q1 99 99 99 99
```

4 Reading and writing timeseries

Some common used file types to store data in R are R data, csv or xls(x) files. Inside R projects, preferably use R data files. Writing and reading R objects to and from R data files is fast, easy and efficient. If data has to be exported, you can use for example csv or Excel files.

4.1 Rds and RData file

The main R data filetypes are `Rds` and `RData`. Both filetypes are R-readable binary files.

Rds stores a single R object. If you want to write more objects you can combine them in a list and write that list to the file. An R object can be stored to a `Rds` file by the `saveRDS` function in the `base` package:

```
> saveRDS(rts, file = "output.rds")
```

The standard extension is `.rds`.

An object saved in a `.rds` file can be reloaded by the `readRDS` function, and you can assign the contents of an `.rds` file, possibly under a different name.

```
> series <- readRDS("output.rds")
```

The result is an R object.

Another native R data type is `RData`. Functions `save` and `load` are available to save and restore single or multiple objects to and from a RData file. When loading, the objects are restored with the same name they had when saved (the `load` function has no `return` variable). This can be annoying. Also it is not clear what the result of the `load` function is.

4.2 External files

Package `regts` includes functions for easily reading and writing timeseries to and from csv or Excel files.

4.2.1 Reading and writing csv files

With the `read_ts_csv` function a `regts` variable can be read from a csv file. This function has several handy arguments such as column and decimal separators, skipping rows or columns and a name function. It employs function `fread` of the `data.table` package.

In most cases it succeeds in finding the correct frequency and determining how the timeseries are stored: rowwise or columnwise. If not, extra input arguments can be given. Consider the following csv file

```
      ,a__mn___,afk__c_wn,afk__f_r_wn,afk__l_wn,afk__r_wn
2010Q1,    6.83,    23.4,         19.3,     3.84,    19.38
2010Q2,    6.98,    23.0,         19.8,     3.45,    19.35
2010Q3,    6.65,    23.4,         19.5,     3.54,    19.83
2011Q1,    6.21,    23.2,         19.8,     3.85,    19.53
```

Notice that the periods are in the first column, the other columns contain the timeseries. The following command will do:

```
> read_ts_csv("test.csv")
```

```
       a__mn___ afk__c_wn afk__f_r_wn afk__l_wn afk__r_wn
2010Q1     6.83      23.4        19.3      3.84     19.38
2010Q2     6.98      23.0        19.8      3.45     19.35
```

```
2010Q3     6.65      23.4        19.5      3.54     19.83
2010Q4       NA        NA          NA        NA        NA
2011Q1     6.21      23.2        19.8      3.85     19.53
```

The function has an argument `rowwise`. We could have used `rowwise = FALSE` to indicate that the timeseries are stored columnwise. But in this case it is not needed, the period column is recognised by the function.

Sometimes the csv file is transposed, as in the following example:

```
names   ; 2017Q1; 2017Q2; 2017Q4
A__MN_BE;   6.83;    6.8;   6.71
AC_FWN  ;  23.34;  23.24;  23.20
BD_WN___;  19.35;  19.38;  19.58
BG_WN___;   3.84;   3.84;   3.54
C__WN___;  19.38;  19.58;  19.35
```

Now the periods are in the first row (column headers), the next rows contain the timeseries. Note that `;` is the field separator and that quarter `2017Q3` is missing in the file.

```
> read_ts_csv("transpose.csv")
```

```
       A__MN_BE AC_FWN BD_WN___ BG_WN___ C__WN___
2017Q1     6.83  23.34    19.35     3.84    19.38
2017Q2     6.80  23.24    19.38     3.84    19.58
2017Q3       NA     NA       NA       NA       NA
2017Q4     6.71  23.20    19.58     3.54    19.35
```

The `sep` argument is not needed, separator `;` (as well as `,` and `tab`) is recognised by the function. By default the separator is determined automatically.

Since `regts` (ànd `ts`) objects always have a continuous period[1], the missing quarters are filled with NA values.

The timeseries names in this file are in uppercase. You can convert them while reading:

```
> read_ts_csv("transpose.csv", name_fun = tolower)
```

```
       a__mn_be ac_fwn bd_wn___ bg_wn___ c__wn___
2017Q1     6.83  23.34    19.35     3.84    19.38
2017Q2     6.80  23.24    19.38     3.84    19.58
2017Q3       NA     NA       NA       NA       NA
2017Q4     6.71  23.20    19.58     3.54    19.35
```

Files may be more complicated then the examples shown above. Possibly there are comments, extra rows or columns, or redundant information in the file. This usually does not cause problems: `read_ts_csv` can skip these redundant rows. If necessary, use arguments `skiprow` and `skipcol`.

Use function `write_ts_csv` to store a `regts` timeseries in a csv file. By default the timeseries are stored rowwise (i.e. the period is written in the first row).

```
> rts <- regts(matrix(1:8, ncol = 2), start = "2016Q1", names = c("a","b"))
> write_ts_csv(rts, "result.csv")
```

You may want to write the data in transposed form, with a different separator:

```
> write_ts_csv(rts, "result_transposed.csv", rowwise = FALSE, sep = "&")
```

[1] `regts` and `ts` objects are regular timeseries. Packages `xts` and `zoo` provide also irregularly spaced timeseries objects. Then periods can be missing.

4.2.2 Reading and writing Excel files

Special functions have been developed for `regts` objects

- `read_ts_xlsx` (employing the `read_excel` function in package `readxl`) for reading an Excel file, both for `xls` and `xlsx` format
- `write_ts_xlsx` for writing an xlsx file employing package `xlsx`

Like the corresponding csv function, `read_ts_xlsx` has arguments for skipping rows or columns, orientation and frequency. Besides that you can specify a sheet, a range and how to recognise na-strings. Consider the next xlsx file:

```
                   Tabel


names       code   2017Q1  2017Q2  2017Q3  2017Q4
A__MN_BE             6.83     6.8      NA    6.71
AC_FWN              23.34   23.24   23.20   23.68
BD_WN___            19.35   19.38   19.58   19.68
BG_WN___             3.84    3.84    3.54    3.74
C__WN___            19.38   19.58   19.35   19.69
```

When reading this file a `regts` object is returned:

```
> read_ts_xlsx("series.xlsx", skiprow = 2, name_fun = tolower, na_string = c("","NA"))
```

```
       a__mn_be ac_fwn bd_wn___ bg_wn___ c__wn___
2017Q1     6.83  23.34    19.35     3.84    19.38
2017Q2     6.80  23.24    19.38     3.84    19.58
2017Q3       NA  23.20    19.58     3.54    19.35
2017Q4     6.71  23.68    19.68     3.74    19.69
```

The function skips the first 2 lines, analyses where the period begins, converts the timeseries names to lowercase and recognises that empty cells and cells with text 'NA' are not available.

Function `write_ts_xlsx` is used for writing a `regts` object to an xlsx file. It creates or opens an Excel workbook depending on argument `append`. Appending to a non existent file means creating the file. You can specify the sheetname (by default Sheet1) and whether you want to write labels. An example:

```
> write_ts_xlsx(rts, "data.xlsx", sheet_name = "extra", append = TRUE)
```

In this case a sheet 'extra' is appended to the xlsx file.

4.2.3 Problems when reading

Sometimes data in a file needs to be processed before it can be converted to a timeseries. This happens when a file has a deviant format.

In that case try reading information as a `data frame` type. Most functions in R for reading a csv return a `data frame`. Examples: `read.csv`, `read.csv2` and `read.table` in package `utils`. Functions, `read_xls(x)` and `read_excel` in package `readxl` or `read.xlsx` (in package `xlsx`) for reading Excel files return some sort of `data frame`.

After some processing (e.g. selecting and/or deleting (parts of) rows and columns) the data frame can be converted to a `regts` as explained in section 11.5.

5 Labels

A label is a way to document an object or the elements of an object. Labels in base R exist but only for objects. In the `regts` package each timeseries in a `regts` object can be equipped with a label. Labels can be defined when the series are created and they are preserved in most timeseries functions.

```
> t1 <- regts(data1, start = "2017Q1", names = c("child", "adult", "senior"),
+        labels = c("Age 0-17", "Age 18-65", "Age 65+"))
> t1
```

```
       child adult senior
2017Q1     1     6      3
2017Q2     3     4      0
2017Q3     2     8      9
```

As you can see the labels are not shown automatically when the timeseries are printed. With the `View` function you can view timeseries and their labels in RStudio. The result looks like this:

```
          child       adult       senior
       Age 0-17   Age 18-65      Age 65+
2017Q1        1           6            3
2017Q2        3           4            0
2017Q3        2           8            9
```

The labels can also be retrieved with the `ts_labels` function:

```
> ts_labels(t1)
```

```
      child       adult      senior
 "Age 0-17" "Age 18-65"   "Age 65+"
```

In combination with the `<-` operator, the `ts_labels` command can be used to assign labels to, or remove labels from an existing `regts`:

```
> tx <- regts(data, start = "2017Q1")
> ts_labels(tx) <- c("Trans X", "Label X")
> ts_labels(tx)
```

```
[1] "Trans X" "Label X"
```

```
> ts_labels(tx) <- NULL       # remove all labels
> ts_labels(tx)
```

```
NULL
```

For updating one or more timeseries labels in a regts object, the function `update_ts_labels` is provided:

```
> t1_update <- update_ts_labels(t1, labels = c(adult = "Age 18-67", senior = "Age 67+"))
```

```
      child       adult      senior
 "Age 0-17" "Age 18-67"   "Age 67+"
```

Argument `labels` is a named character vector with the column names as names and the labels as values.

5.1 Labels in files

In this section we discuss how to read timeseries provided with labels from a csv or xls(x) file, or how to write timeseries including labels to a file. Also the situation is viewed where label information is available on a separate file.

5.1.1 Reading timeseries with labels from a file

It is easy to read timeseries with labels from a csv or xls(x) file, provided that the labels are placed before or after the timeseries names. Read the following Excel file including the labels:

```
name    label          2016Q1 2016Q2 2016Q3 2016Q4
bbpwn  "bbp"           6.83    6.98   6.65   6.21
bet_wn "taxes"         23.4    23.0   23.4   23.2
clt_wn "expenses"      19.3    19.8   19.5   19.8
hl__hr "productivity"  3.84    3.45   3.54   3.86
a__mn_ "labour"        19.38   19.35  19.83  19.54
```

```
> series <- read_ts_xlsx("tslabels.xlsx", labels = "after")
```

Now check the labels:

```
> ts_labels(series)
```

```
         bbpwn         bet_wn         clt_wn         hl__hr         a__mn_
         "bbp"        "taxes"     "expenses" "productivity"       "labour"
```

Labels and timeseries can also be stored columnwise:

```
       arbeidsvolume       bbp  belasting  coll.lasten
            a__m_mn_  bbp_m_wn    bet_c_wn     clt_c_wn
2010Q1          6.83      23.4        19.3         3.84
2010Q2          6.98      23.0        19.8         3.45
2010Q3          6.65      23.4        19.5         3.54
2010Q4          6.21      23.2        19.8         3.85
```

Read this file with result `tseries` and look at the labels:

```
> tseries <- read_ts_xlsx("testlabels.xlsx", labels = "after")
> ts_labels(tseries)
```

```
         bbpwn         bet_wn         clt_wn         hl__hr         a__mn_
         "bbp"        "taxes"     "expenses" "productivity"       "labour"
```

5.1.2 Reading labels from a separate file

Suppose we have a file with label information. This label file contains a list of timeseries names and the corresponding labels:

```
NAMES,LABELS
a,label_a
b,label_b
d,label_d
c,label_c
```

There is also a `regts` object which contains two series: `c` and `b`.

```
> ts <- regts(matrix(1:4, ncol=2), names = c("c","b"), start = "2016")
```

We want to add some labels from this file to the timeseries using the function `update_ts_labels`. Input for this function is a named character vector containing the labels.

The label file contains the labels for the two timeseries. Variable `dflabel` is read from the file as a `data frame` using function `read.csv`:

```
> dflabel <- read.csv("label.csv")
> dflabel
```

```
  NAMES  LABELS
1     a label_a
2     b label_b
3     d label_d
4     c label_c
```

With the `dflabel` variable a named character vector `labels` is constructed. The names can be set with the `names` function:

```
> labels <- dflabel[[2]]
> names(labels) <- dflabel[[1]]
> labels
```

```
      a       b       d       c
label_a label_b label_d label_c
Levels: label_a label_b label_c label_d
```

The variable `labels` contains more labels than necessary. Also the ordering of the labels is not the same as in the column names of the regts object. This is not a problem, the function `update_ts_labels` takes care of all that:

```
> ts <- update_ts_labels(ts, labels)
```

This is the result (of `View(ts)`):

```
        c         b
        label_c   label_b
2016          1         3
2017          2         4
```

5.1.3 Writing timeseries with labels to file

Functions `write_ts_csv` and `write_ts_xlsx` write timeseries to file automatically including the labels. By default they are written after the names. An example:

```
> ts_data <- regts(matrix(1:8, ncol = 2), start = "2016Q1", end = "2016Q4",
+                  names = c("a", "b"), labels = c("Series_a", "Series_b"))
> write_ts_xlsx(ts_data, "result_label.xlsx")
```

Specify `labels = "before"` to write the labels before the names and specify `labels = "no"` to prevent the labels being written.

5.2 Labels in dataframes

How to handle labels when converting a `data frame` to a `regts` or vice versa and when transposing a `data frame` is described in section 11.

6 Operators and functions for `regts`

Since the `regts` class is an extension of the `ts` class all functionality from this class is available. All arithmetic (`+`, `-`, `*`, `/`, `^`, `%%`, `%/%`), logical (`&&`, `||`, `!`), relational (`==`, `!=`, `<`, `<=`, `>`, `>=`) and subsetting (`[]`) operators applicable for a `ts` variable, can be used for a `regts` timeseries. Some examples:

```
> tx <- regts(1:10, start = "2017Q1")
>
> # operators
> ty    <- 2*tx + 1
> tpyth <- tx^2 + ty^2
> modulo <- tx %% 2
> tlog  <- tx < ty && ty > 1
> number <- tx["2017Q2"]
```

When a binary arithmetic operator is used on two timeseries objects, the intersection of the periods is taken:

```
> tx <- regts(1:6, start = "2016Q1")
> ty <- regts(1:6, start = "2017Q1")
> tx + ty
```

```
     Qtr1 Qtr2
2017    6    8
```

You can use all functions applicable to a `ts` class timeseries. Some examples:

```
> tsin <- sin(tx)
> tlog <- log(tx)
> diff(tx)
> plot(tx)
```

Combining timeseries

As we have seen all functions that are available for `ts` are also available for `regts`. The `cbind` function (an S3 generic function) is one of them: it binds two or more objects with a common frequency. With the `cbind` function it is easy to combine several univariate timeseries to a multivariate timeseries, or a combination of univariate and multivariate timeseries to a new multivariate timeseries.

Yet the extension for `regts` has special features: arguments `union` and `suffixes`.

The argument `union` effects the period range. With `union = TRUE`, the default value, the union of the period ranges of the joined objects is taken, otherwise it is the `intersection`. An example:

```
> rt1 <- regts(1:4, start = "2016Q1")
> rt2 <- regts(1:4, start = "2016Q2")
```

```
> cbind(rt1, rt2, union = TRUE)
```

```
       rt1 rt2
2016Q1   1  NA
2016Q2   2   1
2016Q3   3   2
2016Q4   4   3
2017Q1  NA   4
```

```
> cbind(rt1, rt2, union = FALSE)
```

```
       rt1 rt2
2016Q2   2   1
2016Q3   3   2
2016Q4   4   3
```

For `union = TRUE` the result is padded with NA's if needed.

The `suffixes` argument is obligatory if the joined objects have overlapping column names:

```
> rt1 <- regts(data1, start = "2016Q1", names = c("b", "a", "c"))
> rt2 <- regts(data, start = "2016Q1", names = c("d", "b"))
>
> cbind(rt1, rt2, suffixes = c("_1","_2"))
```

```
       b_1  a  c d b_2
2016Q1   1  6  3 1   5
2016Q2   3  4  0 2   6
2016Q3   2  8  9 3   7
2016Q4  NA NA NA 4   8
```

The implementation of `cbind.regts` is also different from function `cbind.ts` (which is an alias for `ts.union`) because of the fact that available column names are always preserved.

7 Multiple or chain calculations

Sometimes you have a 'chain' of calculations, where the left hand side of an equation is used in the right hand side of a following equation. For example:

```
> a <- regts(1:4, start = "2010Q2")
> b <- a + 1
> c <- b / a
> d <- lag(a, -1)
> cbind(a, b, c, d)
```

```
        a  b        c  d
2010Q2  1  2 2.000000 NA
2010Q3  2  3 1.500000  1
2010Q4  3  4 1.333333  2
2011Q1  4  5 1.250000  3
2011Q2 NA NA       NA  4
```

The result is a multivariate `regts`.

The code is less transparent when the chain is evaluated using a multivariate timeseries:

```
> xts <- regts(data, start = "2010Q2", names = c("a", "x"))
> xts[, "b"] <- xts[, "a"] + 1
> xts[, "c"] <- xts[, "b"] / xts[, "a"]
> xts["2010Q3/2011Q2", "d"] <- lag( xts[, "a"], -1)
> xts
```

```
        a  x  b        c  d
2010Q2  1  5  2 2.000000 NA
2010Q3  2  6  3 1.500000  1
2010Q4  3  7  4 1.333333  2
2011Q1  4  8  5 1.250000  3
2011Q2 NA NA NA       NA  4
```

Note that when timeseries `d` is created you also have to select the lagged period. Otherwise the `xts` period is used. The reason is that in an assignment the period is *not* aligned.

In this case another method can be preferable. First convert the multivariate regts to a list of univariate timeseries with the `as.list` function. Then apply the `within` function.

```
> xts <- regts(data, start = "2010Q2", names = c("a", "x"))
> list <- as.list(xts)
> list2 <- within(list, {
+     b <- a + 1
+     c <- b / a
+     d <- lag(a, -1)
+ })
```

The result is a list. The timeseries in this list can be combined with a combination of functions `cbind` and `do.call`. Note the reverse ordering of the columns.

```
> do.call(cbind, list2)
```

```
        a  x  d        c  b
2010Q2  1  5 NA 2.000000  2
2010Q3  2  6  1 1.500000  3
2010Q4  3  7  2 1.333333  4
2011Q1  4  8  3 1.250000  5
2011Q2 NA NA  4       NA NA
```

The elements in the result list all have type `regts`, so the result is a `multivariate regts`.

If another numeric or logical type is added to this list, it is also converted to a regts by `cbind`:

```
> list3 <- within(list2, {
+     e <- 0
+     f <- TRUE
+ })
> do.call(cbind, list3)
```

```
        a  x  d        c  b f e
2010Q2  1  5 NA 2.000000  2 1 0
2010Q3  2  6  1 1.500000  3 1 0
2010Q4  3  7  2 1.333333  4 1 0
2011Q1  4  8  3 1.250000  5 1 0
2011Q2 NA NA  4       NA NA 1 0
```

The elements `e` and `f` are repeated for the whole period and the value `TRUE` is coerced to `numeric`. [2]

8 Update function

Sometimes you want to use a multivariate timeseries to update another multivariate timeseries. Function `update_ts` updates the columns of a timeseries with the values in the columns of another timeseries with the same name.

Of course the two timeseries must have the same frequency but their period ranges may be different (non overlapping periods will be set to NA). For the result timeseries the union of the ranges is taken. There are three different methods to deal with NA values. Possible methods for this function `update_ts` are:

[2] You could also add a character variable, i.e. g <- "a". Now the result will remain a `regts`, but all elements of the timeseries will be coerced to character data, which is probably not what you want.

- `upd` use all of the second timeseries to update the first timeseries
- `updna` only replace NA values in the first timeseries with values from the second timeseries
- `updval` only use valid values in the second timeseries to update the first timeseries

Of course only the common columns in both timeseries are updated. Extra columns in the second timeseries aer added to the result, depending on the method and the content of that columns.

Some examples:

```
> series1 <- regts(matrix(rep(1,6), ncol = 3), start = "2017Q1", names = c("a","b","c"))
> series2 <- regts(matrix(rep(2,4), ncol = 2), start = "2017Q1", names = c("a","d"))
> series1["2017q2", "b"] <- NA
> series2[, "c"] <- NA
> update_ts(series1, series2, method = "upd")
```

```
       a  b  c d
2017Q1 2  1 NA 2
2017Q2 2 NA NA 2
```

```
> update_ts(series1, series2, method = "updna")
```

```
       a  b c d
2017Q1 1  1 1 2
2017Q2 1 NA 1 2
```

```
> update_ts(series1, series2, method = "updval")
```

```
       a  b c d
2017Q1 2  1 1 2
2017Q2 2 NA 1 2
```

In the first result as much values as possible are updated. A difference between the second and the last result: variable “a” is left unchanged, because it is not “NA”, or updated, because of valid values in the second series.

9 Differences between multivariate timeseries

Comparing two multivariate timeseries with a large number of columns can be quiet tedious, therefore function `tsdif` is created. It compares two multivariate timeseries objects which must have the same frequency. The function calculates differences between columns with the same name, they do not have to be in the same order. The actual function used to compute the differences can be specified, by default normal differences are computed. Also argument `tol` can be specified: all differences smaller than `tol` will be ignored. By default `tol` is 0.

```
> rt1 <- regts(matrix(1:12, ncol = 3), start = "2016Q1", names = c("a", "b", "c"))
> # create timeseries with slight differences, larger and smaller than the tolerance factor
> rt2 <- rt1
> rt2[ ,"a"] <- rt1[ ,"a"] + 0.01
> rt2[ ,"b"] <- rt1[ ,"b"] + 1e-6
> rt2
```

```
          a        b  c
2016Q1 1.01 5.000001  9
2016Q2 2.01 6.000001 10
2016Q3 3.01 7.000001 11
2016Q4 4.01 8.000001 12
```

```
> tsdif(rt1, rt2, tol = 1e-3)
```

```
     tsdif timeseries comparison result

Compared timeseries: rt1 and rt2
Difference tolerance: 0.001

Differences

           a
2016Q1 -0.01
2016Q2 -0.01
2016Q3 -0.01
2016Q4 -0.01
Names of timeseries with differences:
[1] "a"
```

The function shows a limited report of the result, but it's result is a list with several components. Some of them are shown in the next example.

```
> difvar <- tsdif(rt1, rt2, tol = 1e-3)
> difvar$equal
```

```
[1] FALSE
```

```
> difvar$difnames
```

```
[1] "a"
```

```
> difvar$tol
```

```
[1] 0.001
```

In this exampl `equal` is `FALSE`, because there are differences larger than the tolerance `tol`. The names of the common columns with differences larger than the tolerance appear in `difnames`. Component `dif` is a `regts` with the computed differences of these columns. If there are no differences `dif` becomes `NULL`. In this case the difference between columns `a` is equal for all quarters: 0.01. The tolerance argument `tol` is actually an input argument but added to the list of components for the sake of completeness.

In the previous example the two objects had the same column names. Now we look at an example where the column names are different. We create a new timeseries `rt3`, an extension of timeseries `rt1` with an extra column:

```
> rt3 <- rt1
> rt3[, "d"] <- regts(9:12, start = "2016Q1")
> tsdif(rt1, rt3)
```

```
     tsdif timeseries comparison result

Compared timeseries: rt1 and rt3

No differences found

Missing timeseries in  rt1 :
[1] "d"
```

```
> tsdif(rt1, rt3)$missing_names1
```

```
[1] "d"
```

The `missing_names1` component contains the column names in the second timeseries that are missing in the first timeseries, in this case variable `d`. There are no `missing_names2`.

```
> tsdif(rt1, rt3)$equal
```

```
[1] FALSE
```

```
> tsdif(rt1, rt3)$dif
```

```
NULL
```

All computed differences are smaller than or equal to the tolerance factor, so component `dif` is `NULL`. Component `equal` is `FALSE`; it is only `TRUE` if both component `dif` is `NULL` and there are no missing names.

Let's compare two columns of regts objects `rt1` and `rt2`: columns `b` differ less than the tolerance factor and columns `c` are equal. Now the result is equal:

```
> dif_bc <- tsdif(rt1[, c("b","c")], rt2[, c("c","b")], tol = 1e-5)
> dif_bc$equal
```

```
[1] TRUE
```

As you can see the ordering of the columns makes no difference.

10 period and period_range

When the same period is used several times, it is preferable to define a variable containing this period. This could be a text string containing a period but package `regts` also defines special classes representing periods: the `period` class for a single period and the `period_range` class for a range of periods. With these classes is it possible to manipulate the period or the period_range, as shown further on in this section.

10.1 period

A `period` object can be created with the `period` function.

```
> prd1 <- period("2016Q4")
> prd1
```

```
[1] "2016Q4"
```

```
> prd2 <- period("2015-2", frequency = 4)
> prd2
```

```
[1] "2015Q2"
```

In the last example the `frequency` has to be specified. `"2015-2"` is ambiguous. Here "2" denotes the second quarter but it could also be for instance a month indicator or a half year. The `frequency` argument is required if you want to define a period with a frequency other than 1, 4 or 12, since there is no special format for these frequencies.

Special functions have been designed for the `period` class e.g. to extract a subperiod or the year:

```
> get_subperiod(prd1)
```

```
[1] 4
```

```
> get_year(prd2)
```

```
[1] 2015
```

`period` objects can be used as input for the `start` and `end` arguments of the `regts` function.

```
> ts    <- regts(1:10, start = prd1)
```

When printed, a `period` variable seems to be nothing more than a character string. With the `str` function we can show the internal representation of a `period` object:

```
> str(period("2016Q4"))
```

```
Class 'period'  atomic [1:1] 8067
  ..- attr(*, "frequency")= num 4
```

The internal representation of period `2016Q4` is the number of quarters since the beginning of the Christian era, starting with 0 (`8067 = 2016 * 4 + 3`).

10.2 period_range

There is also a `period_range`, an object that represents an interval of periods. Such a range can be created in the following ways:

```
> range <- period_range("2014Q4", "2017Q4")
> range
```

```
[1] "2014Q4/2017Q4"
```

```
> period_range("2014Q4/2017Q4")
```

```
[1] "2014Q4/2017Q4"
```

The lower or upper bound of the range may be undetermined:

```
> period_range("2017Q1", NULL)      # No upper bound
```

```
[1] "2017Q1/"
```

A `period_range` variable can be used in the `regts` function, specifying the `period` argument:

```
> range1 <- period_range("2016Q1", "2016Q3")
> rts <- regts(matrix(1:18, ncol = 3), period = range1, names = c("a", "b", "c"))
> rts
```

```
       a b  c
2016Q1 1 7 13
2016Q2 2 8 14
2016Q3 3 9 15
```

A `period_range` variable can also be obtained from a timeseries:

```
> get_period_range(rts)
```

```
[1] "2016Q1/2016Q3"
```

The length of a `period_range` can be determined by the `nperiod` function:

```
> nperiod(range1)
```

```
[1] 3
```

Functions `start_period` and `end_period` are used to retrieve the first and last period in a `period_range`:

```
> start_period(range1)
```

```
[1] "2016Q1"
```

```
> end_period(range1)
```

```
[1] "2016Q3"
```

```
> range2 <- period_range(start_period(range1)+1, end_period(range1)-1)
> ts <- regts(data, period = range2)
```

10.3 Shifting period(s)

With operators `+` and `-` the period can be shifted. Some examples for `period`:

```
> period("2016Q4") + 1
```

```
[1] "2017Q1"
```

```
> p1 <- period("2015Q1")
> p1-5
```

```
[1] "2013Q4"
```

And for a `period_range`:

```
> period_range("2016Q4", "2017Q4") + 1
```

```
[1] "2017Q1/2018Q1"
```

```
> period_range(p1, p1+4)             # a `period` as input
```

```
[1] "2015Q1/2016Q1"
```

Note that one of the operands must be numeric[3].

10.4 Selection with period(_range) objects

In section 3 we have shown how periods can be selected with a string. A `period` can also be used to select or modify a period of a timeseries object. Consider the following (previously defined) timeseries:

```
> rts
```

```
       a b  c
2016Q1 1 7 13
2016Q2 2 8 14
2016Q3 3 9 15
```

Use a `period` to select and modify the timeseries.

```
> prd1 <- period("2016Q2")
> rts[prd1]
```

```
       a b  c
2016Q2 2 8 14
```

```
> rts[prd1] <- 3
> rts
```

[3] Command `period("2015Q1") + period("2015Q2")` results in an error message: 'Arithmetic operation on two periods is not allowed.'

```
       a b  c
2016Q1 1 7 13
2016Q2 3 3  3
2016Q3 3 9 15
```

A `period_range` can be used to select and modify multiple periods:

```
> # define a period range and make selections
> range <- period_range("2016Q2", "2016Q3")
> var <- rts[range]
> rts[range, c("a", "b")]
```

```
       a b
2016Q2 3 3
2016Q3 3 9
```

```
> # define another period range and create a new timeseries column
> smpl <- period_range("2016Q1", "2017Q1")
> rts[smpl, "xx"] <- 2
> rts
```

```
        a  b  c xx
2016Q1  1  7 13  2
2016Q2  3  3  3  2
2016Q3  3  9 15  2
2016Q4 NA NA NA  2
2017Q1 NA NA NA  2
```

Note that because timeseries `xx` is created with an extended period, all timeseries in (multivariate regts) object `rts` are extended.

11 Conversion between regts and data frame

In R a data frame is a much-used type for storing data tables or vectors. It is therefore useful to convert a `data frame` object to a `regts` and vice versa. In this section we explain how this can be done. We also discuss how to handle labels. And we give an example how to read information from a deviant file.

11.1 data frame to regts

Suppose we have a data frame with (numerical) data. It is possible to use such a data frame as input data for the `regts` function:

```
> df  <- data.frame(a = 1:3, b = 4:6, c = 7:9)
> ts <- regts(df, start = "2017Q1")             # create a multivariate regts
> ts
```

```
       a b c
2017Q1 1 4 7
2017Q2 2 5 8
2017Q3 3 6 9
```

The function `regts` employs the `data.matrix` function in the `base` package to convert a data frame to a numeric matrix[4]. By default the column names of the data frame are used for the timeseries names.

[4]Columns with text strings are converted to NA

As with matrix input, it is also possible to specify names or add labels. Specifying names is already described in section 2.2. Labeling of `data frames` will be discussed in section 11.3.

In the previous example we had to specify the period of the result timeseries. Sometimes the period is already contained in the data frame, e.g. in the row names. Consider the following data frame:

```
> df <- data.frame(a = 1:3, b = 4:6)   # data frame with period in row names
> rownames(df) <- c("2015Q3", "2015Q4", "2016Q1")
> df
```

```
       a b
2015Q3 1 4
2015Q4 2 5
2016Q1 3 6
```

Now the function `as.regts` can be used to convert a `data.frame` to a `regts`:

```
> ts <- as.regts(df)
> ts
```

```
       a b
2015Q3 1 4
2015Q4 2 5
2016Q1 3 6
```

Function `as.regts` assumes that the period is stored in the row names. If the periods are in a column of the data frame, then the `time_column` argument can be used. You can specify the column name or number of the data frame in which the period is stored. Specifying 0 means the period is contained in the row names of the data frame.

```
> # data frame with period in column 'periods'
> df2 <- data.frame(periods = c("2015Q3", "2015Q4", "2016Q1"), a = 1:3, b = 4:6)
> ts2 <- as.regts(df2, time_column = "periods")
> ts2
```

```
       a b
2015Q3 1 4
2015Q4 2 5
2016Q1 3 6
```

Function `as.regts` uses the function `period` to convert the 'period text' (like `2015Q3`) to a `period`. This function can handle several period formats: `2016`, `2016Q4`, `2016.4q`, `2016/4Q`, `2016-4q`, `2016_q4`, ... where no distinction is made between 'q' and 'Q'.

If the period is ambiguous, use the frequency argument:

```
> # data frame with indicisive period in row names
> df <- data.frame(a = 1:3, b = 4:6)
> rownames(df) <- c("2015-3", "2015-4", "2016-1")
> ts <- as.regts(df, frequency = 4)
```

If the data frame contains periods in another, non standard, format, it is possible to specify your own function for the conversion of a text to a `period`. Use the `fun` argument, if necessary with extra arguments.

11.2 `regts` to `data frame`

The reverse conversion from a `regts` to a `data frame` can be achieved with function `as.data.frame`. The `as.data.frame` function has been extended for `regts` types. Variable `ts` has been previously defined:

```
> as.data.frame(ts)
```

```
       a b
2015Q3 1 4
2015Q4 2 5
2016Q1 3 6
```

The results look similar to `ts`. The timeseries names have become the data frame column names and the periods show up in the row names. Of course the `classes` are different.

11.3 Labels when converting `data frame` to `regts` and vice versa

Base R does not have facilities for adding labels to a data frame. However the `Hmisc` package has introduced a function `label` for adding label attributes to data frames columns and to retrieve them. Package `regts` uses this function to preserve the labels, when converting a `regts` object with labels to a `data frame` or vice versa.

Consider the following data frame:

```
> df <- data.frame(a = 1:3, b = 6:8, c = 10:12)
> rownames(df) <- c("2017", "2018", "2019")
```

Now add labels to the columns of the data frame. First load the library:

```
> library(Hmisc)
> label(df) <- list(a = "aap", b = "noot", c = "mies")
```

If you want to see the labels in the result, retrieve them with the `label` function[5].

```
> label(df)
```

```
     a      b      c
 "aap" "noot" "mies"
```

When this data frame with labels is converted to a multivariate regts you can display the labels as usual employing the `ts_labels` function:

```
> tt <- as.regts(df)
> ts_labels(tt)
```

```
     a      b      c
 "aap" "noot" "mies"
```

Similarly, if a `regts` object with labels is converted to a data frame, then the timeseries labels are used to create the label attributes of the data frame.

```
> ts_data <- regts(matrix(1:8, ncol = 2), start = "2016Q1", end = "2016Q4",
+                  names = c("a", "b"), labels = c("Series_a", "Series_b"))
> df1 <- as.data.frame(ts_data)
> label(df1)
```

```
         a          b
"Series_a" "Series_b"
```

11.4 Transposing a `data frame` (including labels)

Sometimes it can be necessary to transpose a `data frame`. Package `regts` includes a transpose function: `transpose_df`. Variable `df` in the next example is defined in the previous section.

[5] You can also view the whole data frame including labels with the `View` function in Rstudio.

```
> df
```

```
     a b  c
2017 1 6 10
2018 2 7 11
2019 3 8 12
```

```
> dft <- transpose_df(df)
> dft
```

```
  labels 2017 2018 2019
a    aap    1    2    3
b   noot    6    7    8
c   mies   10   11   12
```

As you can see, if a data frame contains labels, the `transpose_df` function puts the labels in the first column of the result data frame. The labels have now become data and the resulting data frame contains no labels.

However, function `transpose_df` has an argument `label_column`. If you specify this argument, then the values in the selected column are used to create labels:

```
> df2 <- transpose_df(dft, label_column = "labels")
> df2
     a b  c
2017 1 6 10
2018 2 7 11
2019 3 8 12
> label(df2)
     a      b      c
 "aap" "noot" "mies"
```

11.5 Example: reading timeseries from a deviant file

As discussed in section 4.2 package `regts` provides functions for directly reading timeseries from csv or Excel files (`read_ts_csv` and `read_ts_xlsx`). However, sometimes these functions cannot be used, because the file has a deviant layout.In that case try reading the information as a form of data frame and convert this data frame to a `regts` timeseries. An example file:

```
                                    kt      soort    type     2018    2019 2020 2021
AOW heffing door beperkt indexeren
                                    lasten  Lh       Fiscaal  3       3    3    3
                                    trans   Lh       Fiscaal  1       1    1    1
                                    kas     Lh       Fiscaal  1       1    1    1
Willekeurige afschrijving investeringen
                                    lasten  Inkh     Fiscaal
                                    trans   Inkh     Fiscaal  0       0    0    0
                                    kas     Inkh     Fiscaal -2.325
Vervallen vrijstelling MRB auto's > 25jr
                                    lasten  Vb       Fiscaal
                                    trans   Vb       Fiscaal  0       0    0    0
                                    kas     Vb       Fiscaal -7
```

The timeseries are stored rowwise, but there are extra comment lines with a description of the three variables below. We want to use these descriptions to create labels for the timeseries. If we didn't need them, there would be no problem, comment rows are normaly skipped when using functions `read_ts_csv` and read_ts_xlsx. A more serious problem is that there are no unique variable names. Therefore we want to combine columns 'soort' and 'kt' to create these names.

Read this file with the read_excel function. The result of this function is a tibble, a special sort of data frame.

```
> df <- read_excel("test.xlsx")
> df
```

```
# A tibble: 12 x 8
                                         X__1     kt soort    type `2018` `2019` `2020` `2021`
                                        <chr>  <chr> <chr>   <chr>  <dbl>  <dbl>  <dbl>  <dbl>
 1      AOW heffing door beperkt indexeren   <NA>  <NA>    <NA>     NA     NA     NA     NA
 2                                       <NA> lasten    Lh Fiscaal  3.000      3      3    3.0
 3                                       <NA>  trans    Lh Fiscaal  1.000      1      1    1.0
 4                                       <NA>    kas    Lh Fiscaal  1.000      1      1    1.0
 5 Willekeurige afschrijving investeringen   <NA>  <NA>    <NA>     NA     NA     NA     NA
 6                                       <NA> lasten  Inkh Fiscaal     NA     NA     NA     NA
 7                                       <NA>  trans  Inkh Fiscaal  0.000      0      0    0.0
 8                                       <NA>    kas  Inkh Fiscaal -2.325     NA     NA     NA
 9  Vervallen vrijstelling MRB auto's>25jr   <NA>  <NA>    <NA>     NA     NA     NA     NA
10                                       <NA> lasten    Mb Fiscaal  6.000      6      6    0.0
11                                       <NA>  trans    Mb Fiscaal  6.000      6      6     NA
12                                       <NA>    kas    Mb Fiscaal  6.000      6      6    0.6
```

For collecting the labels, select the rows[6] with label information (not NA).

```
> is_label_row <- !is.na(df[[1]])
> labels <- df[[1]][is_label_row]
> labels
```

```
[1] "AOW heffing door beperkt indexeren"      "Willekeurige afschrijving investeringen"
[3] "Vervallen vrijstelling MRB auto's>25jr"
```

Each label is used three times:

```
> labels <- rep(labels,each = 3)
```

Remove comment rows by selecting rows with NA in first column. This is the reverse of the selection we applied before.

```
> df <- df[!is_label_row, ]
```

Next construct lowercase row names from columns soort and kt

```
> df$names <- tolower(paste(df$soort, df$kt, sep = "_"))
> df$names
```

```
[1] "lh_lasten"   "lh_trans"    "lh_kas"      "inkh_lasten" "inkh_trans"  "inkh_kas"
[7] "mb_lasten"   "mb_trans"    "mb_kas"
```

Now remove redundant columns 1 to 4 ('X___1', 'kt', 'soort' and 'type') and transpose the data frame with the colname_column argument to set the column names:

```
> df[1:4] <- NULL
> dft <- transpose_df(df, colname_column = "names")
```

Finally, convert to regts and add the labels.

[6] For selecting a single column by position from a tibble [[is used.

```
> rts <- as.regts(dft)
> ts_labels(rts) <- labels
> rts
```

```
     lh_lasten lh_trans lh_kas inkh_lasten inkh_trans inkh_kas mb_lasten mb_trans mb_kas
2018         3        1      1          NA          0   -2.325         6        6    6.0
2019         3        1      1          NA          0       NA         6        6    6.0
2020         3        1      1          NA          0       NA         6        6    6.0
2021         3        1      1          NA          0       NA         0       NA    0.6
```

12 Aggregate functions

12.1 Function aggregate

The function `aggregate` can be used to convert a timeseries to a lower frequency. For example to convert a monthly timeseries to a quarterly timeseries, or a quarterly timeseries to a yearly timeseries.

An example with a monthly timeseries:

```
> regtm1 <- regts(1:24, start = "2016M1")
> regtm1
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2016   1   2   3   4   5   6   7   8   9  10  11  12
2017  13  14  15  16  17  18  19  20  21  22  23  24
```

```
> aggregate(regtm1, FUN = mean, nfrequency = 4)
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    2    5    8   11
2017   14   17   20   23
```

And another with a quarterly timeseries:

```
> regt1 <- regts(1:10, start = "2016Q1")
> regt1
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    1    2    3    4
2017    5    6    7    8
2018    9   10
```

```
> aggregate(regt1, FUN = sum)
```

When the newlines are removed the result looks like this:

```
regt1  Time Series: Start = 2016  End = 2017  Frequency = 1  [1] 10 26
```

The quarters in 2016 add up to 10, the quarters in 2017 to 26. Note that the 2 quarters in 2018 are ignored. The `aggregate` function skips the incomplete years at the end.

A special situation occurs when for instance a quarterly timeseries doesn't start in the first quarter:

```
> regt2 <- regts(2:10, start = "2016Q2")
> regt2
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016         2    3    4
```

```
2017     5     6     7     8
2018     9    10
```

```
> aggregate(regt2, FUN = sum)
```

Now the result is:

```
regt2  Time Series: Start = 2017     End = 2017     Frequency = 1  [1] 26
```

The quarters in 2017 add up to 26 as before. But now the information for 2016 is also not complete, and therefore ignored. This is what you would expect: for `regts` objects the `aggregate` function skips all years for which not all quarters are present.

Usually the `aggregate` function works the same for `regts` and `ts`, but in this case the result is very different. The same example with a `ts` timeseries:

```
> t2    <- ts(2:10, start = c(2016,2), frequency = 4)
> aggregate(t2, FUN = sum)
```

The result is:

```
t2     Time Series: Start = 2016.25  End = 2017.25  Frequency = 1  [1] 14 30
```

Now the first year (2016) is not ignored. This leads to a shifted result period (the start is `2016.25`) and shifted input ($14 = 2 + 3 + 4 + 5$), which makes the results peculiar.[7]

So the function `aggregate` operates differently for a `regts` or a `ts` object, if the first period does not start at a subperiod for the new frequency.

12.2 Aggregation for growth timeseries

Another case is the treatment of timeseries that contain absolute, relative or percentage changes. For this case the function `aggregate_gr` is developed with special aggregation algorithms, the so called ‘cumulative growth’-methods. A more detailed description of these methods can be found in the next subsection. This function can be employed for both `regts` and `ts` timeseries.

All methods convert input timeseries with high frequency to low frequency outputseries. There are four different type of methods for different types of input timeseries:

The `dif1s` and `dif1` methods assume that the input timeseries is a first difference of length 1 in the input frequency (for `dif1s` the input is also scaled) and calculate a first difference of length 1 in the output frequency.

The `pct` and `rel` methods assume that the input timeseries is a one-period relative or percentage change and calculate the exact relative or percentage change for the output timeseries.

First a timeseries is created containing differences:

```
> xdif <- diff(regts(c(1,3,4,3,6,2,4,1,3,2), start = "2015Q4"))
> xdif
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2016    2    1   -1    3
2017   -4    2   -3    2
2018   -1
```

Two examples of aggregation methods are shown:

```
> aggregate_gr(xdif, method = "dif1s", nfrequency = 1)  # result is a yearly timeseries
```

[7] This problem does not occur when a quarterly timeseries ends before the last quarter, or before the end of the quarter in case of a monthly timeseries. Then the last year or quarter is ignored.

```
2017 -1.5
```

```r
> aggregate_gr(xdif, method = "dif1")
```

```
2017 -6
```

12.3 The cumulative growth methods

The so called growth timeseries require special methods, a simple averaging of the subperiods in the high frequency observations will not return the correct answer. In this section a more detailed description of the four previously defined frequency conversion methods is given.

In advance some definitions are needed:

We are temporally aggregating a timeseries x with n subperiods to a timeseries X with a lower frequency. In other words we want to convert a timeseries x with a high frequency (monthly, quarterly) to a new timeseries with a lower frequency (quarterly, annual).

Let $x_{t,i}$ stand for the value of x in subperiod i of main period t. The index i takes on values in the range $1..n$. We always interpret $x_{t,0}$ to mean $x_{t-1,n}$. The time index t refers to the periods in the time domain of X. Many conversion methods only need the observations subperiod i of main period t. For example the `mean` method calculates X_t as follows

$$X_t = \sum_{i=1}^{i=n} x_{t,i}/n$$

`dif1s` method

Define x to be the first difference of z scaled to the output frequency. Thus

$$x_{t,i} = n(z_{t,i} - z_{t,i-1})$$

We can also define $z_{t,i}$ in terms of x as follows

$$\begin{aligned} z_{t,i} &= z_{t,i-1} + x_{t,i}/n \\ &= z_{t-1,n} + \sum_{j=1}^{i} x_{t,j}/n \end{aligned}$$

Now define Z_t as the level timeseries in time domain t corresponding to z

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} z_{t,i} \\ &= \sum_{i=1}^{n} z_{t-1,n} + \sum_{i=1}^{n}\sum_{j=1}^{i} x_{t,j}/n \\ &= \sum_{i=1}^{n} z_{t-2,n} + \sum_{i=1}^{n}\sum_{j=1}^{n} x_{t-1,j}/n + \sum_{i=1}^{n}\sum_{j=1}^{i} x_{t,j}/n \end{aligned}$$

Then X in any period t can be calculated from

$$X_t = Z_t - Z_{t-1}$$

Since Z_{t-1} can be written as

$$Z_{t-1} = \sum_{i=1}^{n} z_{t-2,n} + \sum_{i=1}^{n}\sum_{j=1}^{i} x_{t-1,j}/n$$

it is easy to show that X_t does not depend on $z_{t-2,n}$. In any practical algorithm $z_{t-2,n}$ can be set to 0.

A compact equation for X_t can be derived by substituting the equations for Z_t and Z_{t-1} into the definition of X_t

$$X_t = \sum_{i=1}^{n} \left(\sum_{j=i+1}^{n} x_{t-1,j} + \sum_{j=1}^{i} x_{t,j} \right) /n$$

This expression can be futher simplified by changing the order of summation. For the first summation, we can write

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} x_{t-1,j} = \sum_{j=2}^{n}\sum_{i=1}^{j-1} x_{t-1,j} = \sum_{j=2}^{n}(j-1)x_{t-1,j}$$

Similarly,

$$\sum_{i=1}^{n}\sum_{j=1}^{i} x_{t,j} = \sum_{j=1}^{n}\sum_{i=j}^{n} x_{t,j} = \sum_{j=1}^{n}(n-j+1)x_{t,j}$$

The final equation is given by

$$X_t = \left(\sum_{j=2}^{n}(j-1)x_{t-1,j} + \sum_{j=1}^{n}(n-j+1)x_{t,j} \right) /n$$

`dif1` method

Define x to be the first difference of z. Thus

$$x_{t,i} = z_{t,i} - z_{t,i-1}$$

We can also define $z_{t,i}$ in terms of x as follows

$$\begin{aligned} z_{t,i} &= z_{t,i-1} + x_{t,i} \\ &= z_{t-1,n} + \sum_{j=1}^{i} x_{t,j} \end{aligned}$$

Further derivations are analogous to the case for the `dif1s` method described in the previous section.

`rel` method

Define x to be the relative change in z. Thus

$$x_{t,i} = (z_{t,i} - z_{t,i-1})/z_{t,i-1}$$

We can also define $z_{t,i}$ in terms of x as follows

$$\begin{aligned} z_{t,i} &= z_{t,i-1}(1 + x_{t,i}) \\ &= z_{t-1,n} \prod_{j=1}^{i} (1 + x_{t,j}) \\ &= z_{t-2,n} \prod_{j=1}^{n} (1 + x_{t-1,j}) \prod_{j=1}^{i} (1 + x_{t,j}) \end{aligned}$$

Now define Z_t as the level timeseries in time domain t corresponding to z

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} z_{t,i} \\ &= z_{t-1,n} \sum_{i=1}^{n} \prod_{j=1}^{i} (1 + x_{t,j}) \\ &= z_{t-2,n} \prod_{j=1}^{n} (1 + x_{t-1,j}) \sum_{i=1}^{n} \prod_{j=1}^{i} (1 + x_{t,j}) \end{aligned}$$

Then X in any period t can be calculated from

$$X_t = Z_t/Z_{t-1} - 1$$

Using the expression for Z_{t-1}

$$Z_{t-1} = z_{t-2,n} \sum_{i=1}^{n} \prod_{j=1}^{i} (1 + x_{t-1,j})$$

we obtain

$$X_t = \frac{\prod_{j=2}^{n}(1 + x_{t-1,j}) \sum_{i=1}^{n} \prod_{j=1}^{i}(1 + x_{t,j})}{1 + \sum_{i=2}^{n} \prod_{j=2}^{i}(1 + x_{t-1,j})} - 1$$

`pct` method

Define x to be the percentage change in z. Thus

$$x_{t,i} = 100(z_{t,i} - z_{t,i-1})/z_{t,i-1}$$

We can also define $z_{t,i}$ in terms of x as follows

$$
\begin{aligned}
z_{t,i} &= z_{t,i-1}(1 + 0.01x_{t,i}) \\
&= z_{t-1,n} \prod_{j=1}^{i} (1 + 0.01x_{t,j})
\end{aligned}
$$

Now define Z_t as the level timeseries in time domain t corresponding to z

$$Z_t = \sum_{i=1}^{n} z_{t,i}$$

Then X in any period t can be calculated from

$$X_t = 100(Z_t/Z_{t-1} - 1)$$

Further derivations are analogous to the case for the `dif1s` method described in the previous section.