Package ‘collapse’

April 1, 2020

Title Advanced and Fast Data Transformation
Version 1.1.0
Date 2020-03-27

BugReports https://github.com/SebKrantz/collapse/issues

Description A C/C++ based package for advanced data transformation in R that is extremely fast, flexible and parsimonious to code with and programmer friendly. It is well integrated with 'dplyr', 'plm' and 'data.table'.

--- Key Features: ---

(1) Advanced data programming: A full set of fast statistical functions supporting grouped and/or weighted computations on vectors, matrices and data.frames. Fast (ordered) and reusable grouping, quick data conversions, and quick select, replace or add data.frame columns.

(2) Advanced aggregation: Fast and easy multi-data-type, multi-function, weighted, parallelized and fully customized data aggregation.

(3) Advanced transformations: Fast (grouped, weighted) replacing and sweeping out of statistics, scaling, centering, higher-dimensional centering, complex linear prediction and partialling-out.

(4) Advanced time-computations: Fast (sequences of) lags / leads, and (lagged / leaded, iterated) differences and growth rates on (unordered) time-series and panel data. Multivariate auto, partial and cross-correlation functions for panel data. Panel data to (ts-)array conversions.

(5) List Processing: (Recursive) list search / identification, extraction / subsetting, data-apply, and row-binding / unlisting in 2D.

(6) Advanced data exploration: Fast (grouped, weighted, panel-decomposed) summary statistics for complex multilevel / panel data.

License GPL (>= 2)
Encoding UTF-8
LazyData true
Depends R (>= 3.5.0)
Imports Rcpp (>= 1.0.1), lfe (>= 2.7)
LinkingTo Rcpp
Suggests dplyr, plm, data.table, ggplot2, scales, vars, knitr, rmarkdown, testthat, microbenchmark
**SystemRequirements**  C++11

**VignetteBuilder**  knitr

**NeedsCompilation**  yes

**Author**  Sebastian Krantz [aut, cre],
Matt Dowle [ctb],
Arun Srinivasan [ctb],
Simen Gaure [ctb],
Dirk Eddelbuettel [ctb],
R Core Team and contributors worldwide [ctb],
Martyn Plummer [cph],
1999-2016 The R Core Team [cph]

**Maintainer**  Sebastian Krantz <sebastian.krantz@graduateinstitute.ch>

**Repository**  CRAN

**Date/Publication**  2020-04-01 12:20:03 UTC

---

**R topics documented:**

- collapse-package .......................... 3
- A0-collapse-documentation .................. 8
- A1-fast-statistical-functions .............. 9
- A2-fast-grouping .......................... 11
- A3-select-replace-vars .................... 14
- A4-quick-conversion ....................... 17
- A5-advanced-aggregation ................... 18
- A6-data-transformations .................... 22
- A7-time-series-panel-series ............... 23
- A8-list-processing ........................ 24
- A9-summary-statistics ..................... 26
- AA1-recode-replace ......................... 26
- AA2-small-helpers .......................... 28
- BY ......................................... 30
- dapply ..................................... 33
- descr ..................................... 35
- extract-list ............................... 37
- fbetween-fwithin-B-W ....................... 39
- fdiff-fgrowth-D-G .......................... 43
- ffirst-flast ................................ 48
- fFtest ..................................... 50
- fHDbetween-fHDwithin-HDB-HDW ............... 52
- flag-L-F ................................... 56
- fmax-fmin .................................. 60
- fmean ..................................... 63
- fmedian ..................................... 65
- fmode ...................................... 67
- fNdistinct .................................. 69
- fNobs ...................................... 71
collapse-package

Advanced and Fast Data Transformation

Description
collapse is a C/C++ based package for data manipulation in R. It's aims are

- to facilitate complex data transformation and exploration tasks in R
- to help make R code fast, flexible, parsimonious and programmer friendly.

It is compatible with dplyr, data.table and the plm approach to panel-data.

Key Features:

1. Advanced data programming: A full set of fast statistical functions supporting grouped and/or weighted computations on vectors, matrices and data.frames. Fast (ordered) and reusable grouping, quick data conversions, and quick select, replace or add data.frame columns.


3. Advanced transformations: Fast (grouped, weighted) replacing and sweeping out of statistics, scaling, centering, higher-dimensional centering, complex linear prediction and partialling-out.


5. List Processing: (Recursive) list search / identification, extraction / subsetting, data-apply, and row-binding / unlisting in 2D.

Getting Started

Please see Collapse Documentation & Overview, or the introductory vignette. A compact set of examples is also provided below.

Details

collapse provides an integrated set of functions organized into several topics (see Collapse Overview). Many are S3 generic with methods for vectors, matrices and data.frames. Inputs are quickly passed to compiled C/C++ code, enabling flexible and parsimonious coding at extreme speeds.

The package avoids non-standard evaluation and exports core methods for maximum programmability. Smart attribute handling and additional (not-exported) methods ensure compatibility and support for dplyr, data.table and the plm approach to panel-data. collapse comes with a built-in hierarchical documentation facilitating the use of the package.

collapse is mainly coded in C++ and built with Rcpp, but also uses C functions from data.table (grouping, row-subsetting, row-binding), lfe (centering on multiple factors) and stats (ACF and PACF).

Author(s)

Maintainer: Sebastian Krantz <sebastian.krantz@graduateinstitute.ch>

Other contributors from packages collapse utilizes:

- Matt Dowle, Arun Srinivasan and contributors worldwide (data.table)
- Simen Gaure (lfe)
- Dirk Eddelbuettel and contributors worldwide (Rcpp)
- R Core Team and contributors worldwide (stats)

I also thank Ralf Stubner, Joseph Wood and Dirk Eddelbuettel for helpful answers on Stackoverflow, and Joris Meys on R-Devel for encouraging me and helping to set up the github repository for collapse.

Developing / Feature Requests / Bug Reporting

- If you are interested in extending or optimizing this package, see the source code at https://github.com/SebKrantz/collapse/tree/master, fork and send pull-requests, or e-mail me.
- Please send feature requests via e-mail.
- Please report issues at https://github.com/SebKrantz/collapse/issues or e-mail me.

Examples

# World Bank World Development Data: 216 countries, 59 years, 4 series (columns 9-12)
head(wlddev)

# Describe data
descr(wlddev)

# Panel-summarize columns 9 though 12 of this data (within and between countries)
qsu(wlddev, pid = ~ country, cols = 9:12, vlabels = TRUE)

# Do all of that by region and also compute higher moments -> returns a 4D array
qsu(wlddev, ~ region, ~ country, cols = 9:12, higher = TRUE)

# Return as nested list of statistics-matrices instead
suml <- qsu(wlddev, ~ region, ~ country, cols = 9:12, higher = TRUE, array = FALSE)
str(suml)

# Create data.frame from this list with 3 identifier columns
unlist2d(suml, idcols = c("Variable","Trans"), row.names = "Region")

# Compute the means of all the regions and create a simpler data.frame instead
unlist2d(rapply2d(suml, fmean), idcols = c("Variable","Trans"))

# Select columns from wlddev: same as wlddev[9:12] but 2x faster and works with data.tables etc.
series <- get_vars(wlddev, 9:12)

# Replace columns, 4x faster than wlddev[9:12] <- series and also replaces names
get_vars(wlddev, 9:12) <- series

# Fast conversion to data.table with qDT, and subset rows..
library(data.table)
qDT(wlddev)[country == "Ireland"]

# Calculating fast column-wise statistics
fNobs(series) # Number of non-missing values
fmean(series) # means of series
fmedian(series) # medians of series
fmin(series) # mins of series

# Fast grouped statistics
fNobs(series, wlddev$region) # regional number of obs
fmean(series, wlddev$region) # regional means
fmedian(series, wlddev$region) # regional medians
fsd(series, wlddev$region) # regional standard-deviations

# Means by region and income
fmean(series, get_vars(wlddev, c("region","income")))

# Same using GRP objects:
g <- GRP(wlddev, ~ region + income)
print(g)
plot(g)

# GRP objects are extremely efficient inputs to fast functions
fmean(series, g)
fmedian(series, g)
fsd(series, g)

# Faster aggregations with dplyr:
library(dplyr) # This is a lot faster than summarize_all(mean)
wlddev %>% group_by(region, income) %>% select(PCGDP, LIFEEX) %>% fmean

# Data-Apply to columns
head(dapply(series, log))
dapply(series, quantile, na.rm = TRUE)

# Data-Apply to rows (for sum use rowSums(qM(series), na.rm = TRUE), same for rowMeans ...)
head(dapply(series, max, MARGIN = 1, na.rm = TRUE))
head(dapply(mtcars, quantile, MARGIN = 1))

# qM -> quickly convert data to matrix, qDF/qDT do the reverse
fmean(rowSums(qM(series), na.rm = TRUE))

# Split-apply combine computing on columns
BY(series, wlddev$region, sum, na.rm = TRUE) # Please use: fsum(series, wlddev$region) -> faster
BY(series, wlddev$region, quantile, na.rm = TRUE)
BY(series, wlddev$region, quantile, na.rm = TRUE, expand.wide = TRUE)

# Convert panel-data to array
psar <- psmat(wlddev, ~ country, ~ year, cols = 9:12)
str(psar)
psar[c("Ireland",], ] # Fast data access
psar[c("Ireland", "PCGDP"]
psar[, "2016"][, ]
qDF(psar[, "2016", ], row.names.col = "Country") # Convert to data.frame
plot(psar) # Visualize
plot(psar, colour = TRUE, labs = vlabels(wlddev)[9:12])
plot(psar[c("Brazil", "India", "South Africa", "Russian Federation", "China"), ,
        c("PCGDP", "LIFEEX", "ODA")], legend = TRUE, labs = vlabels(wlddev)[c(9:10, 12)])
plot(ts(psar[c("Brazil",), , ], 1960, 2018), main = "Brazil, 1960-2018")

# Aggregate this data by country and decade: Numeric columns with mean, categorical with mode
head(collap(wlddev, ~ country + decade, fmean, fmode))

# Multi-function aggregation of certain columns
head(collap(wlddev, ~ country + decade,
           list(fmean, fmedian, fsd),
           list(ffirst, flast), cols = c(3, 9:12)))

# Customized Aggregation: Assign columns to functions
head(collap(wlddev, ~ country + decade,
           custom = list(fmean = 9:10, fsd = 9:12, flast = 3, ffirst = 6:8)))

# Fast functions can also do grouped transformations:
head(fsd(series, g, TRA = "/")) # Scale series by region and income
head(fsum(series, g, TRA = "/")) # Percentages by region and income
head(fmean(series, g, TRA = "/")) # Demean / center by region and income
head(fmedian(series, g, TRA = "/")) # De-median by region and income
gmeds <- fmedian(series, g)
gmeds <- fmedian(series, g)
head(TRA(series, gmeds, "/", g))

# Faster transformations with dplyr:
Here are demeaning PCGDP and LIFEEX using weighted means, weighted by ODA:

```r
wlddev %>% group_by(region, income) %>% select(PCGDP, LIFEEX, ODA) %>% fmean(ODA, "-"
```

But there are also specialized transformation operators for common jobs:

- Centering (Within-transforming) the 4 series by country:
  ```r
  head(W(wlddev, ~ country, cols = 9:12))
  ```

- Same but adding overall mean back after subtracting out group means:
  ```r
  head(W(wlddev, ~ country, cols = 9:12, mean = "overall.mean")
  ```

- Partialing out country and year fixed effects from 2 series (qF = quick-factor):
  ```r
  head(HDW(wlddev, PCGDP + LIFEEX ~ qF(country) + qF(year)))
  ```

- Same, adding ODA as continuous regressor:
  ```r
  head(HDW(wlddev, PCGDP + LIFEEX ~ qF(country) + qF(year) + ODA))
  ```

- Standardizing (scaling and centering) by country:
  ```r
  head(STD(wlddev, ~ country, cols = 9:12))
  ```

- Computing 1 lead and 3 lags of the 4 series: Panel-computations efficient and exactly identified:
  ```r
  head(L(wlddev, -1:3, ~ country, ~year, cols = 9:12))
  ```

- Computing the 1- and 10-year first differences of the 4 series:
  ```r
  head(D(wlddev, c(1,10), 1, ~ country, ~year, cols = 9:12))
  ```

- First and second differences:
  ```r
  head(D(wlddev, c(1,10), 1:2, ~ country, ~year, cols = 9:12))
  ```

- 1-year lagged and leaded FD:
  ```r
  head(D(wlddev, -1:1, 1, ~ country, ~year, cols = 9:12))
  ```

- Computing the 1- and 10-year growth rates of the 4 series (also keeping the level series):
  ```r
  head(G(wlddev, c(0,1,10), 1, ~ country, ~year, cols = 9:12))
  ```

- Adding exactly identified growth rates using data.table:
  ```r
  setDT(wlddev)[, paste0("G.", names(wlddev)[9:12]) := fgrowth(.SD,1,1,iso3c,year), .SDcols = 9:12]
  ```

- Computing the 1- and 10-year log-differences of GDP per capita and Life-Expectancy:
  ```r
  head(G(wlddev, c(0,1,10), 1, PCGDP + LIFEEX ~ country, ~year, logdiff = TRUE))
  ```

- Same transformations using plm package:

  ```r
  library(plm)
pwlddev <- pdata.frame(wlddev, index = c("country","year"))
  W(pwlddev$PCGDP) # Country-demeaning
  W(pwlddev, cols = 9:12)
  W(pwlddev$PCGDP, effect = 2) # Time-demeaning
  W(pwlddev, effect = 2, cols = 9:12)
  HDW(pwlddev$PCGDP) # Country- and time-demeaning
  HDW(pwlddev, cols = 9:12)
  STD(pwlddev$PCGDP) # Standardizing by country
  STD(pwlddev, cols = 9:12)
  L(pwlddev$PCGDP, -1:3) # Panel-lags
  L(pwlddev, -1:3, 9:12)
  G(pwlddev$PCGDP) # Panel-Growth rates
  G(pwlddev, 1, 1, 9:12)
  ```
### Description

The following table fully summarizes the contents of `collapse`. The documentation follows a hierarchical structure: This is the main overview page, linking to topical overview pages and associated function pages (unless functions are documented on the topic page). Calling `?FUN` brings up the documentation page for FUN, with links to associated topic pages and closely related functions.

### Topics and Functions

<table>
<thead>
<tr>
<th>Topic</th>
<th>Main Features / Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fast Statistical Functions</strong></td>
<td>Fast (grouped and weighted) statistical functions for vector, matrix, data.frame and grouped_df (dplyr compatible).</td>
</tr>
<tr>
<td><strong>Fast Grouping</strong></td>
<td>Fast (ordered or unordered) groupings from vectors, data.frames, lists. ’GRP’ objects are extremely efficient inputs for <code>collapse</code>.</td>
</tr>
<tr>
<td><strong>Quick Select and Replace Variables</strong></td>
<td>Quick and flexible select and replace (or add) variables from (to) data.frames / data.tables / tibbles etc... (speed about 2x ‘[’ for selecting and 4x ‘[&lt;‐’ for replacing).</td>
</tr>
<tr>
<td><strong>Quick Data Conversion</strong></td>
<td>Quick conversions: data.frame &lt;-&gt; data.table</td>
</tr>
<tr>
<td><strong>Advanced Data Aggregation</strong></td>
<td>Fast and easy (weighted and parallelized) aggregation of multi-type data, with (multiple) columns. Also supports fully customized aggregation tasks mapping functions directly to columns.</td>
</tr>
<tr>
<td><strong>Data Transformations</strong></td>
<td>Efficient row- and column-data-apply and Split-Apply-Combine computing. Fast (grouped and weighted) aggregations are particularly efficient.</td>
</tr>
<tr>
<td><strong>Time-Series and Panel-Series</strong></td>
<td>Fast (sequences of) lags / leads and (lagged / leaded and iterated) differences and growth conversions. Multivariate panel- auto-, partial- and cross-correlation functions. Additional methods for grouped_df (is.regular).</td>
</tr>
<tr>
<td><strong>List Processing</strong></td>
<td>Extremely fast (one-pass, grouped and weighted), summary statistics for cross-sectional data.</td>
</tr>
<tr>
<td><strong>Summary Statistics</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Recode and Replace Values</strong></td>
<td>Recode multiple values (exact or regex matching) and replace NaN/Inf/–Inf and others.</td>
</tr>
<tr>
<td><strong>Small (Helper) Functions</strong></td>
<td>Set and extract variable labels, extract variable classes and C storage types, display variable values...</td>
</tr>
<tr>
<td><strong>Data and Global Macros</strong></td>
<td>Groningen Growth and Development Centre 10-Sector Database, World Bank World Development indicators, etc.</td>
</tr>
</tbody>
</table>
Details

The added top-level documentation infrastructure in collapse allows you to effectively navigate the package (as in other commercial software documentations like Mathematica). Calling `?FUN` brings up the documentation page documenting the function as in other R packages. You can also call topical documentation pages directly from the console. The links to these pages are contained in `.COLLAPSE_TOPICS` (i.e. calling `help(.COLLAPSE_TOPICS[1])` brings up this page).

Author(s)

Maintainer: Sebastian Krantz <sebastian.krantz@graduateinstitute.ch>

See Also

collapse-package

A1-fast-statistical-functions

Fast (Grouped, Weighted) Statistical Functions for Matrix-Like Objects

Description

With `fsum`, `fprod`, `fmean`, `fmedian`, `fvar`, `fsd`, `fmin`, `fmax`, `ffirst`, `flast`, `fNobs` and `fNdistinct`, collapse presents a coherent set of extremely fast and flexible statistical functions (S3 generics) to perform column-wise, grouped and weighted computations on atomic vectors, matrices and data.frames, with special support for `dplyr` grouped tibbles and data.table's.

(Note: The vector-valued functions and operators `fscale/STD`, `fbetween/B`, `fHDbetween/HDB`, `fwithin/W`, `fHDbetween/HDW`, `flag/L/F`, `fdiff/D` and `fgrowth/G` are documented under Data Transformations and Time-Series and Panel-Series. These functions also support plm::pseries and plm::pdata.frame's.)

Usage

```r
## All functions (FUN) follow a common syntax in 4 methods:
FUN(x, ...)

## Default S3 method:
FUN(x, g = NULL, [w = NULL,] TRA = NULL, [na.rm = TRUE,]
   use.g.names = TRUE, ...)

## S3 method for class 'matrix'
FUN(x, g = NULL, [w = NULL,] TRA = NULL, [na.rm = TRUE,]
   use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
FUN(x, g = NULL, [w = NULL,] TRA = NULL, [na.rm = TRUE,]
   use.g.names = TRUE, drop = TRUE, ...)
```
use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'grouped_df'
FUN(x, [w = NULL,] TRA = NULL, [na.rm = TRUE,]
    use.g.names = FALSE, keep.group_vars = TRUE, [keep.w = TRUE,] ...)

Arguments

x   a vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).

g   a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.

w   a numeric vector of (non-negative) weights, may contain missing values. Supported by fsum, fprod, fmean, fvar, fsd and fmode.

TRA  an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "-" | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "*" | 9 - "%%" | 10 - "-%%". See TRA.

na.rm  logical. Skip missing values in x. Defaults to TRUE in all functions and implemented at very little computational cost. Not available for fNobs.

use.g.names  make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.

drop  matrix and data.frame methods: Drop dimensions and return an atomic vector if g = NULL and TRA = NULL.

keep.group_vars  grouped_df method: Logical. FALSE removes grouping variables after computation.

keep.w  grouped_df method: Logical. TRUE also aggregates weights and saves them in a column, FALSE removes weighting variable after computation (if contained in grouped_df).

...  arguments to be passed to or from other methods, and extra arguments to some functions, i.e. the algorithm used to compute variances etc.

Details

Please see the documentation of individual functions.

Value

x aggregated. data.frame column-attributes and overall attributes are preserved.
See Also

Collapse Overview, Data Transformations, Time-Series and Panel-Series

Examples

```r
## default vector method
mpg <- mtcars$mpg
fsum(mpg) # Simple sum
fsum(mpg, TRA = "%") # Simple transformation: obtain percentages of mpg
fsum(mpg, mtcars$cyl) # Grouped sum
fmean(mpg, mtcars$cyl) # Grouped mean
fmean(mpg, w = mtcars$hp) # Weighted mean, weighted by hp
fmean(mpg, mtcars$cyl, mtcars$hp) # Grouped mean, weighted by hp
fmean(mpg, mtcars$cyl, TRA = "%") # Percentages by group
fmean(mpg, mtcars$cyl, mtcars$hp, # Replace vector elements with their weighted group-mean
     TRA = "replace")

## data.frame method
fsum(mtcars)
fsum(mtcars, TRA = "%")
fsum(mtcars, mtcars[,c(2,8:9)]) # Grouped column sum
g <- GRP(mtcars, ~ cyl + vs + am) # Here precomputing the groups!
fsum(mtcars, g) # Faster !!
fmean(mtcars, g, mtcars$hp)
fmean(mtcars, g, mtcars$hp, "-")) # demeaning by weighted group means... see also ?W

fmode(wlddev, drop = FALSE) # Compute statistical modes of variables in this data
fmode(wlddev, wlddev$income) # grouped statistical modes ..

## matrix method
m <- qM(mtcars)
fsum(m)
fmean(m, g) # ...

## method for grouped tibbles - for use with dplyr
library(dplyr)
mtcars %>% group_by(cyl, vs, am) %>% select(mpg, carb) %>% fsum
mtcars %>% group_by(cyl, vs, am) %>% fsum(TRA = "%")
mtcars %>% group_by(cyl, vs, am) %>% fmean(hp) # weighted grouped mean, save sum of weights
mtcars %>% group_by(cyl, vs, am) %>% fmean(hp, keep.group_vars = FALSE)
```

Description

GRP performs fast, ordered and unordered, groupings of vectors and data.frames (or lists of vectors) using data.table’s fast grouping and ordering C routine (forder). The output is a list-like object of class 'GRP' which can be printed, plotted and used as an efficient input to all of collapse’s fast functions, operators, as well as collaps, BY and TRA.
Usage

```r
GRP(X, ...)
```

## Default S3 method:
```r
GRP(X, by = NULL, sort = TRUE, order = 1L, na.last = TRUE, 
    return.groups = TRUE, return.order = FALSE, ...)
```

## S3 method for class 'factor'
```r
GRP(X, ...)
```

## S3 method for class 'qG'
```r
GRP(X, ...)
```

## S3 method for class 'pseries'
```r
GRP(X, effect = 1L, ...)
```

## S3 method for class 'pdata.frame'
```r
GRP(X, effect = 1L, ...)
```

## S3 method for class 'grouped_df'
```r
GRP(X, ...)
```

is.GRP(x)
```r
group_names.GRP(x, force.char = TRUE)
as.factor.GRP(x)
```

## S3 method for class 'GRP'
```r
print(x, n = 6, ...)
```

## S3 method for class 'GRP'
```r
plot(x, breaks = "auto", type = "s", horizontal = FALSE, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>a vector, list of columns or data.frame (default method), or a classed object (conversion/extractor methods).</td>
</tr>
<tr>
<td>x</td>
<td>a GRP object.</td>
</tr>
<tr>
<td>by</td>
<td>if X is a data.frame or list, by can indicate columns to use for the grouping (by default all columns are used). Columns must be passed using a vector of column names, indices, or using a one-sided formula i.e. ~ col1 + col2.</td>
</tr>
<tr>
<td>sort</td>
<td>logical. sort the groups (argument passed to data.table:::forderv, TRUE is like using keyby with data.table, vs. by).</td>
</tr>
<tr>
<td>order</td>
<td>integer. sort the groups in ascending (1L, default) or descending (-1L) order (argument passed to data.table:::forderv).</td>
</tr>
<tr>
<td>na.last</td>
<td>logical. if missing values are encountered in grouping vector/columns, assign them to the last group (argument passed to data.table:::forderv).</td>
</tr>
<tr>
<td>return.groups</td>
<td>logical. include the unique groups in the created 'GRP' object.</td>
</tr>
<tr>
<td>return.order</td>
<td>logical. include the output from data.table:::forderv in the created 'GRP' object.</td>
</tr>
</tbody>
</table>
The sorting and ordering functionality for GRP only affects (2), that is groups receive different integer-id’s depending on whether the groups are sorted sort = TRUE, and in which order (order = 1 ascending or order = -1 descending). This in-turn changes the order of values/rows in the output of collapse functions (the row/value corresponding to group 1 always comes out on top). The default setting with sort = TRUE and order = 1 results in groups being sorted in ascending order. This is equivalent to performing grouped operations in data.table using keyby, whereas sort = FALSE is equivalent to data.table grouping with by.

Evidently GRP is an S3 generic function with one default method supporting vector and list input and several conversion methods. The most important of these is the conversion of factors to ‘GRP’ objects and vice-versa. To obtain a ‘GRP’ object from a factor, one simply gets the number of groups calling ng <- length(levels(f)) (1) and then computes the count of each level (3) using tabulate(f,ng). The integer group-id (2) is already given by the factor itself after removing the levels and class attributes. The levels are put in a list and moved to position (4) in the ‘GRP’ object, which is reserved for the unique groups. Going from factor to ‘GRP’ object thus only requires a tabulation of the levels, whereas creating a factor from a ‘GRP’ object using as.factor.GRP does not involve any computations, but may involve interactions if multiple grouping columns were used (which are then interacted to produce unique factor levels) or as.character conversions if the grouping column(s) were numeric (which are potentially expensive).

Note: For faster factor generation and a factor-light class ‘qG’ which avoids the coercion of factor levels to character also see qF and qG.

Value

A list-like object of class ‘GRP’ containing information about the number of groups, the observations (rows) belonging to each group, the size of each group, the unique group names / definitions, whether the groups are ordered or not and (optionally) the ordering vector used to perform the ordering. The object is structured as follows:
### Description

Efficiently select and replace (or add) a subset of columns from (to) a data frame. This can be done by data type, or using column names, indices, logical vectors, functions or regular expressions.

The performance is generally faster than `\[\]`. It is also secure w.r.t. redefinitions of `\[\.data.frame\]` or `\<-\.data.frame\` for other classes (i.e. data.table’s, tibbles etc.) and prevents the loss of attributes, but does not offer a lot of security in terms of performing all kinds of costly checks on the data.frame’s or when lists of unequal-length columns are offered as replacements.

### Examples

```r
## default method
GRP(mtcars$ cyl)

## convert factor to GRP object
GRP(iris$Species)

## get GRP object from a dplyr grouped tibble
library(dplyr)
mtcars %>% group_by(cyl, vs, am) %>% GRP
```

---

### List-index Element-name | Content type | Content description
---

1. N.groups | integer(1) | Number of Groups
2. group.id | integer(NROW(X)) | An integer group-identifier
3. group.sizes | integer(N.groups) | Vector of group sizes
4. groups | unique(X) or NULL | Unique groups (same format as input, sorted if sort = TRUE), or NULL if return.groups = FALSE
5. group.vars | character | The names of the grouping variables
6. ordered | logical(2) | [1] TRUE if sort = TRUE, [2] TRUE if X already sorted
7. order | integer(NROW(X)) or NULL | Ordering vector from data.table:::forderv or NULL (the default)
8. call | call | The GRP() call, obtained from match.call()
Usage

## Select and replace columns by data type

num_vars(x, return = c("data","names","indices","named_indices"))

num_vars(x) <- value

nv(x, return = c("data","names","indices","named_indices")) # Short for num_vars

cat_vars(x) <- value

cat_vars(x, return = c("data","names","indices","named_indices"))

char_vars(x) <- value

char_vars(x, return = c("data","names","indices","named_indices"))

fact_vars(x, return = c("data","names","indices","named_indices"))

fact_vars(x) <- value

logi_vars(x, return = c("data","names","indices","named_indices"))

logi_vars(x) <- value

logi_vars(x, return = c("data","names","indices","named_indices"))

Date_vars(x, return = c("data","names","indices","named_indices"))

Date_vars(x) <- value

## Select and replace columns by names, indices, logical vectors, regular expressions or using other functions to identify columns

going_vars(x, vars, return = c("data","names","indices","named_indices"),
regex = FALSE, ...)

get_vars(x, vars, return = c("data","names","indices","named_indices"),
regex = FALSE, ...) <- value

get_vars(x, vars, regex = FALSE, ...) <- value # Short for get_vars<-

## Add columns at any position within a data.frame

add_vars(x, ..., pos = "end")

add_vars(x, pos = "end") <- value # Short for add_vars

Arguments

x a data.frame.

value a data.frame or list of columns whose dimensions exactly match those of the extracted subset of x. If only 1 variable is in the subset of x, value can also be an atomic vector or matrix, provided that NROW(value) == nrow(x).

vars a vector of column names, indices (can be negative), a suitable logical vector, a vector of regular expressions matching column names if regex = TRUE, or a function returning TRUE or FALSE when applied to the columns of x.

return an integer or string specifying what to return. The options are:
1. "data"
2. "names"
3. "indices"
4. "named_indices"

Note: replacement functions only replace data, not column names or indices (ordering). However column names are replaced together with the data.

- **regex**: logical. **TRUE** will do regular expression search on the column names of x using a (vector of) regular expression(s) passed to `vars`.
- **pos**: the position where columns are added in the data.frame. "end" (default) will add columns at the end (right) of the data.frame, "front" will add columns in front (left). Alternatively one can pass a vector of positions (matching `length(value)` if value is a list). In that case the other columns will be shifted around the new ones while maintaining their order.

... for `get_vars`: further arguments passed to `grep`, if `regex = TRUE`. For `add_vars`: Same as value. A single argument passed may also be a vector or matrix, multiple arguments must each be a list (they are combined using `c(...)`).

**See Also**

- **Collapse Overview**

**Examples**

```r
## Wold Development Data
head(num_vars(wlddev)) # Select numeric variables
head(get_vars(wlddev, is.numeric)) # Same thing
head(cat_vars(wlddev)) # Select categorical (non-numeric) vars
head(get_vars(wlddev, is.categorical)) # Same thing
num_vars(wlddev) <- num_vars(wlddev) # Replace Numeric Variables by themselves
get_vars(wlddev, is.numeric) <- get_vars(wlddev, is.numeric) # Same thing

head(get_vars(wlddev, 9:12)) # Select columns 9 through 12, 2x faster
head(get_vars(wlddev, -(9:12))) # All except columns 9 through 12
head(get_vars(wlddev, c("PCGDP", "LIFEEX", "GINI", "ODA"))) # Select using column names
head(get_vars(wlddev, c("[[:upper:]]", regex = TRUE))) # Same thing: match upper-case var. names

get_vars(wlddev, 9:12) <- get_vars(wlddev, 9:12) # 6x faster wlddev[9:12] <- wlddev[9:12]
add_vars(wlddev) <- STD(gv(wlddev, 9:12), wlddev$iso3c) # Add Standardized columns 9 through 12
head(wlddev) # gv and av are shortcuts

get_vars(wlddev, 13:16) <- NULL # Efficient Deleting added columns again
av(wlddev, "front") <- STD(gv(wlddev, 9:12), wlddev$iso3c) # Again adding in Front
head(wlddev)
get_vars(wlddev, 1:4) <- NULL # Deleting
av(wlddev, c(10, 12, 14, 16)) <- W(wlddev, ~iso3c, cols = 9:12, # Adding next to original variables
```

A3-select-replace-vars
Conversion common data objects quickly, without method dispatch and extensive checks:

- qDF and qDT convert vectors, matrices, higher-dimensional arrays and suitable lists to data.frame and data.table respectively.
- qM converts vectors, higher-dimensional arrays, data.frames and suitable lists to matrix.
- mctl and mrtl are exported C++ functions that column- or row-wise convert a matrix to list, data.frame or data.table. They are used internally by qDF and qDT, dapply, BY, etc...
- qF converts atomic vectors to factor.
- qG is a programmers function that converts atomic vectors to a quick-group (class attribute 'qG') - a kind of factor light which is just an integer vector with an attribute 'N.groups' indicating the number of groups. (Thus saving time consumed by converting the levels of a numeric vector to character).

Usage

\[
\begin{align*}
qDF(X, row.names.col = FALSE) \\
qDT(X, row.names.col = FALSE) \\
qM(X) \\
mctl(X, names = FALSE, ret = 0L) \\
mrtl(X, names = FALSE, ret = 0L) \\
qF(x, ordered = TRUE, na.exclude = TRUE) \\
qG(x, ordered = TRUE, na.exclude = TRUE)
\end{align*}
\]

Arguments

- **X** a vector, matrix, higher-dimensional array, data.frame or list. mctl and mrtl only take matrices.
- **x** a atomic vector or factor.
- **row.names.col** should a column capturing names or row.names be added? i.e. when converting atomic objects to data.frame or data.frame to data.table. Can be logical TRUE, which will add a column "row.names" in front, or can supply a name for the column i.e. "column1".
- **ordered** logical. TRUE sorts the levels and returns an ordered factor.
- **names** logical. Should the list be named?
ret  integer. 3 return options: 0L - return a list, 1L - return a data.frame, 2L - return a data.table.

na.exclude logical. TRUE preserves missing values in character or numeric vectors (i.e. no underlying integer is generated to represent NA, and functions like na.rm will remove missing values from the factor). FALSE creates an integer for NA and also attaches a class 'na.included', which will make computations with collapse functions faster (since no NA checking is required). Note that in both cases a 'NA' level is attached, so this option only concerns the underlying integer vector and the response to functions dealing with missing values (like na.rm).

Value

qDF - returns a data.frame
qDT - returns a data.table
qM - returns a matrix
mctl, mrtl - return a list, data.frame or data.table
qF - returns a factor
qG - returns a quick-group (= integer vector)

See Also

GRP, Collapse Overview

Examples

mtcarsM <- qM(mtcars) # Matrix from data.frame
mtcarsDT <- qDT(mtcarsM) # data.table from matrix columns
mrtl(mtcarsM, TRUE, 2L) # data.table from matrix rows, etc...
qDF(mtcarsM, "cars") # Adding a row.names column when converting from matrix
qDT(mtcars, "cars") # Saving row.names when converting data.frame to data.table

cylF <- qF(mtcars$cyl) # Factor from atomic vector
cylG <- qG(mtcars$cyl) # Quick-group from atomic vector
cylG # See the simple structure of this object
Usage

# Main function: allows formula and data input to `by` argument
collap(X, by, FUN = fmean, catFUN = fmode, cols = NULL, custom = NULL,
      keep.by = TRUE, keep.col.order = TRUE, sort.row = TRUE, parallel = FALSE,
      mc.cores = 1L, return = c("wide","list","long","long_dupl"),
      give.names = "auto", ...)

# Auxiliary function: allows column names and indices input to `by` argument
collapv(X, by, FUN = fmean, catFUN = fmode, cols = NULL, custom = NULL,
       keep.by = TRUE, keep.col.order = TRUE, sort.row = TRUE, parallel = FALSE,
       mc.cores = 1L, return = c("wide","list","long","long_dupl"),
       give.names = "auto", ...)

# Auxiliary function: allows dplyr 'grouped_df' input
collapg(X, FUN = fmean, catFUN = fmode, cols = NULL, custom = NULL,
        keep.group_vars = TRUE, keep.col.order = TRUE, sort.row = TRUE, parallel = FALSE,
        mc.cores = 1L, return = c("wide","list","long","long_dupl"),
        give.names = "auto", ...)

Arguments

X a data.frame, or an object coercible to data.frame using qDF.

by a one-or two sided formula, i.e. ~ group1 or var1 + var2 ~ group1 + group2,
or alternatively a factor, GRP object, atomic vector (internally converted to factor)
or a list of vectors / factors (internally converted to a GRP object) used to group
X. collapv additionally takes names or indices of grouping columns (could also
use a logical vector or a selector function such as is.categorical).

FUN a function, list of functions (i.e. list(fsum,fmean,fsd) or list(myfun1 =
function(x)...,sd = sd)), or a character vector of function names, which are
automatically applied only to numeric variables.

catFUN same as FUN, but applied only to categorical (non-numeric) typed columns (is.categorical).

cols select columns to aggregate using a function, column names or indices. Note:
cols is ignored if a two-sided formula is passed to by.

custom a named list specifying a fully customized aggregation task. The names of the
list are function names and the content columns to aggregate using this func-
tion (same input as cols). For example custom = list(fmean = 1:6,fsd =
7:9,fmode = 10:11) tells collap to aggregate columns 1-6 of X using the mean,
columns 7-9 using the standard deviation etc. Note: custom lets collap ignore
any inputs passed to FUN, catFUN or cols.

keep.by, keep.group_vars logical. FALSE will omit grouping variables from the output.

keep.col.order logical. Retain original column order post-aggregation.

sort.row logical. Sort rows by the groups.

parallel logical. Use parallel::mclapply instead of lapply for multi-function or cus-
tom aggregation.
mc.cores

integer. Argument to `parallel::mclapply` setting the number of cores to use.

return

character. Control the output format when aggregating with multiple functions or performing custom aggregation. "wide" (default) returns a wider data frame with added columns for each additional function. "list" returns a list of data frame's - one for each function. "long" adds a column "Function" and row-binds the results from different functions using `data.table::rbindlist`. "long.dupl" is a special option for aggregating multi-type data using multiple `FUN` but only one `catFUN` or vice-versa. In that case the format is long and data aggregated using only one function is duplicated. See Examples to understand this!

give.names

logical. Create unique names of aggregated columns by adding a prefix 'FUN. '. 'auto' will automatically create such prefixes whenever multiple functions are applied to a column or custom is used.

... additional arguments passed to all functions supplied to `FUN`, `catFUN` or custom.

Details
collap automatically checks each function passed to it whether it is a Fast Statistical Function (i.e. whether the function name is contained in `.FAST_STAT_FUN`). If the function is a fast function, collap only does the grouping and then calls the function to carry out the grouped computations. If the function is not one of `.FAST_STAT_FUN`, `BY` is called internally to perform the computation. The resulting computations from each function are put into a list and recombined to produce the desired output format as controlled by the return argument. When multiple functions are used with collap, setting `parallel = TRUE` and the number of cores with `mc.cores` will instruct collap to execute these function calls in parallel using `parallel::mclapply`. If only a single function is used which is not a `.FAST_STAT_FUN`, the parallel and `mc.cores` arguments are handed down to `BY`. See Examples.

Value

`X` aggregated by by.

See Also

`BY`, Fast Statistical Functions, Collapse Overview

Examples

```r
## World Development Panel Data

# Simple and Multi-Type Aggregation ------------------------------
head(collap(wlddev, ~ country + decade)) # Aggregate by country and decade
head(collap(wlddev, ~ country + decade, cols = is.numeric)) # Aggregate only numeric columns
head(collap(wlddev, ~ country + decade, cols = 9:12)) # Only the 4 series
head(collap(wlddev, PCGDP + LIFEEX ~ country + decade)) # Only GDP and life-expectancy
head(collap(wlddev, PCGDP + LIFEEX ~ country + decade, fsum)) # Using the sum instead
head(collap(wlddev, PCGDP + LIFEEX ~ country + decade, sum, na.rm = TRUE))
head(collap(wlddev, wlddev[c("country","decade")], fsum, ...) # same, exploring different inputs
```
cols = 9:10))
head(collap(wlddev[9:10], wlddev[c("country","decade"], fsum))
head(collapv(wlddev, c("country","decade"), fsum)) # ... names/indices with collapv
head(collapv(wlddev, c(1,5), fsum))

g <- GRP(wlddev, ~ country + decade) # Precomputing the grouping
head(collap(g, keep.by = FALSE)) # This is slightly faster now

# Aggregate categorical data using not the mode but the last element
head(collap(wlddev, ~ country + decade, fmean, flast))
head(collap(wlddev, ~ country + decade, catFUN = flast, # Aggregate only categorical data
cols = is.categorical))

# Weighted aggregation ----------------------------------------------
weights <- abs(rnorm(nrow(wlddev))) # Adding a random weight vector
head(collap(wlddev, ~ country + decade, w = weights)) # Takes weighted mean for numeric..
# ... and weighted mode for categorical data. The weight vector may also have missing values

# Multi-Function Aggregation ----------------------------------------
head(collap(wlddev, ~ country + decade, list(fmean, fNobs), # Saving mean and Nobs
cols = 9:12))
head(collap(wlddev, ~ country + decade,
list(mean = mean,
 Nobs = function(x,...) sum(!is.na(x)),
cols = 9:12, na.rm = TRUE))
head(collap(wlddev, ~ country + decade, # same using base R -> slower
list(fmean, fNobs), cols = 9:12, return = "list")
head(collap(wlddev, ~ country + decade,
list(fmean, fNobs), cols = 9:12, return = "long")
head(collap(wlddev, ~ country + decade,
list(fmean, fNobs), return = "long_dupl")) # and duplicating it 2 times
head(collap(wlddev, ~ country + decade,
list(fmean, fNobs), list(fmode, flast), # categorical data
 keep.col.order = FALSE))
head(collap(wlddev, ~ country + decade,
c("fmean","fsum","fNobs","fsd","fvar"), # more functions, string input,
c("fmode","ffirst","flast","fNdistinct"), # (choose more than 1 cores,
 parallel = TRUE, mc.cores = 1L, # depending on your machine)
 keep.col.order = FALSE))

# Custom Aggregation ----------------------------------------------
head(collap(wlddev, ~ country + decade, # custom aggregation
 custom = list(fmean = 9:12, fsd = 9:10, fmode = 7:8)))
head(collap(wlddev, ~ country + decade, # using column names
 custom = list(fmean = "PCGDP", fsd = c("LIFEEX","GINI"),
 flast = "date")))
collapse provides an ensemble of functions to perform common data transformations efficiently and user friendly:

- **dapply** applies functions to **rows or columns** of matrices and data.frame’s.
- **BY** is an S3 generic for **Split-Apply-Combine computing** and can perform aggregation as well as grouped transformations. (for aggregation please also see **collap** and **Fast Statistical Functions**).
- **TRA** is an S3 generic to efficiently perform (groupwise) **replacement and sweeping out of statistics**. Supported operations are:

<table>
<thead>
<tr>
<th>Integer-id</th>
<th>String-id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;replace_fill&quot;</td>
<td>replace and overwrite missing values</td>
</tr>
<tr>
<td>2</td>
<td>&quot;replace&quot;</td>
<td>replace but preserve missing values</td>
</tr>
<tr>
<td>3</td>
<td>&quot;-&quot;</td>
<td>subtract</td>
</tr>
<tr>
<td>4</td>
<td>&quot;+&quot;</td>
<td>subtract group-statistics but add group-frequency weighted average of group statistics</td>
</tr>
<tr>
<td>5</td>
<td>&quot;/&quot;</td>
<td>divide</td>
</tr>
<tr>
<td>6</td>
<td>&quot;%&quot;</td>
<td>compute percentages</td>
</tr>
<tr>
<td>7</td>
<td>&quot;+&quot;</td>
<td>add</td>
</tr>
<tr>
<td>8</td>
<td>&quot;*&quot;</td>
<td>multiply</td>
</tr>
<tr>
<td>9</td>
<td>&quot;%%&quot;</td>
<td>modulus</td>
</tr>
<tr>
<td>10</td>
<td>&quot;-%%&quot;</td>
<td>subtract modulus</td>
</tr>
</tbody>
</table>

All of collapse’s **Fast Statistical Functions** have a built-in TRA argument for faster access (i.e. you can compute (groupwise) statistics and use them to transform your data with a single function call).

- **fscale/STD** is an S3 generic to perform (groupwise and / or weighted) **scaling / standardizing** of data and is orders of magnitude faster than base::scale.
- **fwithin/W** is an S3 generic to efficiently perform (groupwise and / or weighted) **within-transformations / demeaning / centering** of data. Similarly **fbetween/B** computes (groupwise and / or weighted) **between-transformations / averages**.
- **fHDwithin/HDW**, shorthand for 'higher-dimensional within transform', is an S3 generic to efficiently **center data on multiple groups** and partial-out linear models (possibly involving many levels of fixed effects and interactions). In other words, **fHDwithin/HDW** efficiently computes **residuals** from (potentially complex) linear models. Similarly **fHDbetween/HDB**, short-
hand for 'higher-dimensional between transformation’, computes the corresponding means or fitted values.

- \texttt{fFtest} is a fast implementation of the R-Squared based F-test, to test exclusion restrictions on linear models potentially involving multiple large factors (fixed effects). It internally utilizes \texttt{fHDwithin} to project out factors while counting the degrees of freedom.

- \texttt{flag/L/F}, \texttt{fdiff/D} and \texttt{fgrowth/G} are S3 generics to compute sequences of lags/leads and suitably lagged and iterated differences and growth rates on time-series and panel data. More in Time-Series and Panel-Series.

- STD, W, B, HDW, HDB, L, D and G are parsimonious wrappers around the f- functions above representing the corresponding transformation ‘operators’. They have additional capabilities when applied to data-frames (i.e. variable selection, formula input, auto-renaming and id-variable preservation), and are easier to employ in regression formulas, but are otherwise identical in functionality.

### Table of Functions

<table>
<thead>
<tr>
<th>Function / S3 Generic</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dapply}</td>
<td>No methods, works with matrices and data frames</td>
<td>apply functions to apply Split-Apply-Combine computing</td>
</tr>
<tr>
<td>\texttt{BY}</td>
<td>default, matrix, data.frame, grouped_df</td>
<td>scale / standardize</td>
</tr>
<tr>
<td>\texttt{TRA}</td>
<td>default, matrix, data.frame, grouped_df</td>
<td>demean / center data</td>
</tr>
<tr>
<td>\texttt{fscale/STD}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>compute means / averaging</td>
</tr>
<tr>
<td>\texttt{fwithin/W}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>high-dimensional centering and lm residuals</td>
</tr>
<tr>
<td>\texttt{fbetween/B}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>high-dimensional averages and lm fitted values</td>
</tr>
<tr>
<td>\texttt{fHDwithin/HDW}</td>
<td>default, matrix, data.frame, pseries, pdata.frame</td>
<td>fast F-test of exclusion</td>
</tr>
<tr>
<td>\texttt{fHDbetween/HDB}</td>
<td>default, matrix, data.frame, pseries, pdata.frame</td>
<td>(sequences of lags for high-dimensional data)</td>
</tr>
<tr>
<td>\texttt{fFtest}</td>
<td>No methods, it’s a standalone test to which data needs to be supplied.</td>
<td>(sequences of lags)</td>
</tr>
<tr>
<td>\texttt{flag/L/F}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>(sequences of lags and leads)</td>
</tr>
<tr>
<td>\texttt{fdiff/D}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>(sequences of lagged/leaded differences)</td>
</tr>
<tr>
<td>\texttt{fgrowth/G}</td>
<td>default, matrix, data.frame, pseries, pdata.frame, grouped_df</td>
<td>(sequences of iterated growth rates)</td>
</tr>
</tbody>
</table>

### See Also

Collapse Overview, Fast Statistical Functions, \texttt{collap}, Time-Series and Panel-Series

---

**Description**

collapse provides the following functions to work with time-dependent data:

- \texttt{flag}, and the lag- and lead- operators L and F are S3 generics to efficiently compute sequences of lags and leads on ordered or unordered time-series and panel data.
• \texttt{fdiff, fgrowth}, and the operators \texttt{D} and \texttt{G} are S3 generics to efficiently compute sequences of suitably lagged and iterated differences and growth rates or log-differences on ordered or unordered time-series and panel data. They can also be used to compute forward (leaded) differences or growth rates.

• \texttt{psmat} is an S3 generic to efficiently expand panel-vectors or \texttt{plm::pseries} and \texttt{data.frame}'s or \texttt{plm::pdata.frame}'s to \textbf{panel-series matrices and 3D arrays}, respectively.

• \texttt{psacf, pspacf} and \texttt{psccf} are S3 generics to compute estimates of the \textbf{auto-, partial auto- and cross- correlation or covariance functions} for panel-vectors or \texttt{plm::pseries}, and multivariate versions for \texttt{data.frame}'s or \texttt{plm::pdata.frame}'s.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{S3 Generic} & \textbf{Methods} & \textbf{Description} \\
\hline
\texttt{flag/L/F} & default, matrix, data.frame, pseries, pdata.frame, grouped\_df & compute (sequences of) lags and leads \\
\texttt{fdiff/D} & default, matrix, data.frame, pseries, pdata.frame, grouped\_df & compute (sequences of lagged and iterated) differences \\
\texttt{fgrowth/G} & default, matrix, data.frame, pseries, pdata.frame, grouped\_df & compute (sequences of lagged and iterated) growth rates or log-differences \\
\texttt{psmat} & default, pseries, data.frame, pdata.frame & convert panel-data to matrix/array \\
\texttt{psacf} & default, pseries, data.frame, pdata.frame & compute ACF on panel-data \\
\texttt{pspacf} & default, pseries, data.frame, pdata.frame & compute PACF on panel-data \\
\texttt{psccf} & default, pseries, data.frame, pdata.frame & compute CCF on panel-data \\
\hline
\end{tabular}
\end{table}

\textbf{See Also}
\texttt{Collapse Overview, Data Transformations}

\texttt{A8-list-processing collapse List Processing}

\textbf{Description}

\texttt{collapse} provides the following set of functions to work with lists of R objects:

• \textbf{Search and Identification}
  \begin{itemize}
  \item \texttt{is.regular} checks whether an R object is either atomic or a list. A (nested) list composed of regular objects at each level of the list-tree is unlistable to an atomic vector, checked by \texttt{is.unlistable}.
  \item \texttt{ldepth} determines the level of nesting of the list (i.e. the maximum number of nodes of the list-tree).
  \item \texttt{has.elem} searches elements in a list using element names, regular expressions applied to element names, or a function applied to the elements, and returns \texttt{TRUE} if any matches were found.
  \end{itemize}

• \textbf{Subsetting}
- **atomic_elem** examines the top-level of a list and returns a sublist with the atomic elements. Conversely **list_elem** returns the sublist of elements which are themselves lists or list-like objects.

- **reg_elem** and **irreg_elem** are recursive versions of the former. **reg_elem** extracts the regular part of the list-tree (leading to atomic elements in the final nodes), while **irreg_elem** extracts the ‘irregular’ part of the list tree leading to non-atomic elements in the final nodes. *(Tipp: try calling both on an `lm` object). Naturally for all lists \( l \), `is.unlistable(reg_elem(l))` evaluates to `TRUE`...

- **get_elem** extracts elements from a list using element names, regular expressions applied to element names, a function applied to the elements, or element-indices used to subset the lowest-level sub-lists. By default the result is presented as a simplified list containing all matching elements. With the keep.tree option however **get_elem** can also be used to subset lists i.e. maintain the full tree but cut off non-matching branches.

**Apply Functions**

- **rapply2d** is a recursive version of `base::lapply` with two key differences to `base::rapply`:
  1. Data frames are considered as atomic objects, not as (sub-)lists, and (2) the result is not simplified.

**Unlisting / Row-Binding**

- **unlist2d** efficiently unlists unlistable lists in 2-dimensions and creates a `data.frame` (or `data.table`) representation of the list (unlike `base::unlist` which returns an atomic vector). This is done by recursively flattening and row-binding R objects in the list (using `data.table::rbindlist`) while creating identifier columns for each level of the list-tree and (optionally) saving the row-names of the objects in a separate column. **unlist2d** can thus also be understood as a recursive generalization of `do.call(rbind,l)`, for lists of vectors, data.frames, arrays or heterogeneous objects.

**Table of Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is.regular</td>
<td>`function(x) is.atomic(x)</td>
</tr>
<tr>
<td>is.unlistable</td>
<td>checks if list is unlistable</td>
</tr>
<tr>
<td>ldepth</td>
<td>level of nesting / maximum depth of list-tree</td>
</tr>
<tr>
<td>has_elem</td>
<td>checks if list contains a certain element</td>
</tr>
<tr>
<td>get_elem</td>
<td>subset list / extract certain elements</td>
</tr>
<tr>
<td>get_elem</td>
<td>subset list / extract certain elements</td>
</tr>
<tr>
<td>reg_elem</td>
<td>subset / extract regular part of list</td>
</tr>
<tr>
<td>irreg_elem</td>
<td>subset / extract non-regular part of list</td>
</tr>
<tr>
<td>atomic_elem</td>
<td>top-level subset atomic elements</td>
</tr>
<tr>
<td>list_elem</td>
<td>top-level subset list/list-like elements</td>
</tr>
<tr>
<td>rapply2d</td>
<td>recursively apply functions to lists of data objects</td>
</tr>
<tr>
<td>unlist2d</td>
<td>recursively unlist/row-bind lists of data objects in 2D, to <code>data.frame</code> or <code>data.table</code></td>
</tr>
</tbody>
</table>
See Also

Collapse Overview

---

A9-summary-statistics collapse Summary Statistics

Description
collapse provides the following functions to efficiently summarize data:

- **qsu**, shorthand for quick-summary, is an extremely fast summary command inspired by the (xt)summarize command in the STATA statistical software. It computes a set of 7 statistics (nobs, mean, sd, min, max, skewness and kurtosis) using a numerically stable one-pass method. Statistics can be computed weighted, by groups, and also within-and between entities (for multilevel / panel-data).
- **descr** computes a concise and detailed description of a data.frame, including frequency tables for categorical variables and various statistics and quantiles for numeric variables. It is inspired by Hmisc::describe, but about 10x faster.
- **pwcor**, **pwcov** and **pwNobs** compute pairwise correlations, covariances and observation counts, respectively. Pairwise correlations and covariances can be computed together with observation counts and p-values, and output as 3D array (default) or list of matrices. A major feature of **pwcor** and **pwcov** is the print method displaying all of these statistics in a single correlation table.

Table of Functions

<table>
<thead>
<tr>
<th>Function / S3 Generic</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qsu</td>
<td>default, matrix, data.frame, pseries, pdata.frame</td>
<td>Fast (grouped, weighted, panel-decomposed) summary statistics</td>
</tr>
<tr>
<td>descr</td>
<td>No methods, for data.frame’s or lists of vectors</td>
<td>Detailed statistical description of data.frame</td>
</tr>
<tr>
<td>pwcor</td>
<td>No methods, for matrices or data.frame’s</td>
<td>Pairwise correlations</td>
</tr>
<tr>
<td>pwcov</td>
<td>No methods, for matrices or data.frame’s</td>
<td>Pairwise covariances</td>
</tr>
<tr>
<td>pwNobs</td>
<td>No methods, for matrices or data.frame’s</td>
<td>Pairwise observation counts</td>
</tr>
</tbody>
</table>

See Also

Fast Statistical Functions, Collapse Overview

---

AA1-recode-replace Recode and Replace Values in Matrix-Like Objects
Description

- Recode can be used to replace multiple values in vectors, matrices or data.frames, using either exact (==) or regular expression matching.
- replace_non_finite replaces NaN/Inf/-Inf (or optionally only Inf/-Inf) with a value (default is NA).
- replace_outliers replaces values falling outside a 1- or 2-sided numeric threshold or outside a certain number of column-standard deviations with a value (default is NA).

Usage

Rcode(X, ..., copy = FALSE, reserve.na.nan = TRUE, regex = FALSE)

replace_nonfinite(X, value = NA, replace.nan = TRUE)

replace_outliers(X, limits, value = NA, single.limit = c("SDs","min","max"))

Arguments

X a vector, matrix or data.frame.

... comma-separated recode arguments of the form: name = newname, `2` = 0, `NaN` = 0, `NA` = 0, `Inf` = NA, `-Inf` = NA, etc...

limits either a vector of two-numeric values c(minval,maxval) constituting a two-sided outlier threshold, or a single numeric value constituting either factor of standard deviations (default), or the minimum or maximum of a one-sided outlier threshold. See also single.limit.

value a single (scalar) value to replace matching elements with. Default is NA.

copy logical. For reciprocal or sequential replacements of the form a = b, b = c make a copy of X to prevent a being replaced with b and then all b-values being replaced with c again. In general Recode does the replacements one-after the other, starting with the first.

reserve.na.nan logical. TRUE identifies NA and NaN as special numeric values and does the correct replacement. FALSE will treat NA/NaN as strings, and thus not match numeric NA/NaN. Note: This is not an issue for Inf/-Inf, which are matched in both numeric and character variables.

regex logical. If TRUE, all recode-argument names are (sequentially) passed to grepl as a pattern to search X. All matches are replaced.

replace.nan logical. TRUE (default) replaces NaN/Inf/-Inf. FALSE replaces only Inf/-Inf.

single.limit a character or integer (only applies if length(limits) == 1):

- 1 -"SDs" specifies that limits will be interpreted as a (two-sided) threshold in column standard-deviations. The underlying code is equivalent to X[abs(fscale(X)) > limits] <-value.
- 2 -"min" specifies that limits will be interpreted as a (one-sided) minimum threshold. The underlying code is equivalent to X[X < limits] <-value.
- 3 -"max" specifies that limits will be interpreted as a (one-sided) maximum threshold. The underlying code is equivalent to X[X > limits] <-value.
Note

Recode is not suitable for recoding factors or other classed objects / columns, it simply does
X[X == value] <-replacement in a more efficient way. For classed objects, see for example
dplyr::recode.

See Also

Small (Helper) Functions, Collapse Overview

Examples

Recode(c("a","b","c"), a = "b", b = "c")
Recode(c("a","b","c"), a = "b", b = "c", copy = TRUE)
Recode(month.name, ber = NA, regex = TRUE)
mtcr <- Recode(mtcars, `0` = 2, `4` = Inf, `1` = NaN)
replace_nonfinite(mtcr)
replace_nonfinite(mtcr, replace.nan = FALSE)
replace_outliers(mtcars, c(2, 100)) # replace all values below 2 and above 100 w. NA
replace_outliers(mtcars, 2, single.limit = "min") # replace all value smaller than 2 with NA
replace_outliers(mtcars, 100, single.limit = "max") # replace all value larger than 100 with NA
replace_outliers(mtcars, 2) # replace all values above or below 2 column-
# standard-deviations from the column-mean w. NA
replace_outliers(
# Passing a grouped_df, pseries or pdata.frame
num_vars(dplyr::group_by(iris, Species)), 2) # allows to remove outliers according to
# in-group standard-deviation. see ?fscale

Description

Elementary exported convenience functions in the collapse package. that help to deal with variable
names, labels, missing values, matching and object checking etc... For recoding and replacing
values see Recode.

Usage

vlabels(X, attrn = "label") # Get labels of variables in X, in attr(X[[i]], attrn)
vlabels(X, attrn = "label") <- value # Set labels of variables in X
vclasses(X) # Get classes of variables in X
vtypes(X) # Get data storage types of variables in X (calling typeof)
namlab(X, class = FALSE, attrn = "label") # Return data.frame of names, labels and classes
add_stub(X, stub, pre = TRUE) # Add a stub (i.e. prefix or postfix) to column names
rm_stub(X, stub, ...) # Remove stub from column names (using base::sub)
x %!in% table # The opposite of %in%
ckmatch(x, table, ...) # Check-match: Throws an informative error if non-matched
e = "Unknown columns:"
fnlevels(x) # Faster version of nlevels(x) (for factors)
funique(x, ordered = TRUE) # Faster unique(x) and sort(unique(x)) for vectors
finteraction(...) # Faster interactions, unused levels are dropped
na_rm(x) # Remove missing values from vector and return vector
na_insert(X, prop = 0.1) # Insert missing values at random in vectors, matrices DF's
all_identical(...) # Check exact equality of multiple objects or list-elements
all_obj_equal(...) # Check near equality of multiple objects or list-elements
seq_row(X) # Integer sequences along rows of X
seq_col(X) # Integer sequences along columns of X
setRownames(object = nm, # Set rownames of object and return object
nm = seq_row(object))
setColnames(object = nm, nm) # Set colnames of object and return object
setDimnames(object = dn, dn) # Set dimension names of object and return object
is.categorical(x) # The opposite of is.numeric
is.Date(x) # Check if object is of class "Date", "POSIXlt" or "POSIXct"
as.numeric_factor(X) # Coerce a factor, or all factor columns in X, to numeric
as.character_factor(X) # Coerce a factor, or all factor columns in X, to character

Arguments

X a matrix or data.frame.
object a matrix, array or data.frame.
x, table a atomic vector.
attrn character. Name of attribute to store labels or retrieve labels from.
value a matching character vector of variable labels.
class logical. Also show the classes of variables in X in a column?
stub a single character stub, i.e. "log.", which by default will be pre-applied to all
variables or column names in X.
pre logical. FALSE will post-apply stub.
nm a suitable vector of row- or column-names.
dn a suitable list of dimension names.
ordered logical. TRUE (default) sorts the output, FALSE is slightly faster.
prop specify the proportion of observations randomly replaced with NA.
e The error message thrown by ckmatch.
... for all_identical / all_obj_equal / finteraction: either multiple comma-
separated objects (= vectors or factors in the case of finteraction) or a single
list of objects. For rm_stub: Other arguments passed to stub.

See Also

Rcode, Collapse Overview
Examples

```r
## Variable labels
namlab(wlddev, class = TRUE)
vlabels(wlddev)
vlabels(wlddev) <- vlabels(wlddev)

## Stub-renaming
log_mtc <- add_stub(log(mtcars), "log.")
rm_stub(log_mtc, "log.")  # use "log." or ".log$" to only match beginning or end

## Checking exact equality of multiple objects
all_identical(iris, iris, iris, iris)
l <- replicate(100, fmean(num_vars(iris), iris$Species), simplify = FALSE)
all_identical(l)

## Factor to numeric conversions:
identical(mtcars, as.numeric_factor(dapply(mtcars, qF)))

## Fast interactions
finteraction(mtcars$cyl, mtcars$vs)
finteraction(mtcars)
```

Description

BY is an S3 generic that efficiently applies functions over vectors or matrix- and data.frame columns by groups, and returns various output formats. A simple parallelism is also available.

Usage

```r
BY(X, ...)
```

## Default S3 method:
BY(X, g, FUN, ..., use.g.names = TRUE, sort = TRUE,
   expand.wide = FALSE, parallel = FALSE, mc.cores = 1L,
   return = c("same","list"))

## S3 method for class 'matrix'
BY(X, g, FUN, ..., use.g.names = TRUE, sort = TRUE,
   expand.wide = FALSE, parallel = FALSE, mc.cores = 1L,
   return = c("same","matrix","data.frame","list"))

## S3 method for class 'data.frame'
BY(X, g, FUN, ..., use.g.names = TRUE, sort = TRUE,
   expand.wide = FALSE, parallel = FALSE, mc.cores = 1L,
   return = c("same","matrix","data.frame","list"))
## S3 method for class 'grouped_df'

```r
BY(X, FUN, ..., use.g.names = FALSE, keep.group_vars = TRUE,
   expand.wide = FALSE, parallel = FALSE, mc.cores = 1L,
   return = c("same","matrix","data.frame","list"))
```

### Arguments

- **X**: a atomic vector, matrix or data frame.
- **g**: a factor, `GRP` object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a `GRP` object) used to group x.
- **FUN**: a function, can be scalar- or vector-valued.
- **...**: further arguments to `FUN`.
- **use.g.names**: make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
- **sort**: logical. Sort the groups? Internally passed to `GRP` or `qF`, and only effective if `g` is not already a factor or `GRP` object.
- **expand.wide**: logical. If `FUN` is a vector-valued function returning a vector of fixed length > 1 (such as the `quantile` function), `expand.wide` can be used to return the result in a wider format (instead of stacking the resulting vectors of fixed length above each other in each output column).
- **parallel**: logical. `TRUE` implements simple parallel execution by internally calling `parallel::mclapply` instead of `base::lapply`.
- **mc.cores**: integer. Argument to `parallel::mclapply` indicating the number of cores to use for parallel execution. Can use `parallel::detectCores()` to select all available cores. See also `?parallel::mclapply`.
- **return**: an integer or string indicating the type of object to return. The default 1 -"same" returns the same object type (i.e. passing a matrix returns a matrix and passing a data frame returns a data frame). 2 -"matrix" always returns the output as matrix, 3 -"data.frame" always returns a data frame and 4 -"list" returns the raw (uncombined) output. *Note:* 4 -"list" works together with `expand.wide` to return a list of matrices.
- **keep.group_vars**: `grouped_df` method: Logical. FALSE removes grouping variables after computation.

### Details

`BY` is a frugal reimplementation of the Split-Apply-Combine computing paradigm. It is faster than `base::tapply`, `base::by`, `base::aggregate` and `plyr`, and preserves data attributes just like `dplyr`.

I note at this point that the philosophy of `collapse` is to move beyond this rather slow computing paradigm, which is why the `Fast Statistical Functions` were implemented. However sometimes tasks
need to be performed that involve more complex and customized operations on data, and for these cases BY is a good solution.

BY is built principally as a wrapper around `lapply(split(x,g),FUN,...)`, but strongly optimizes on attribute checking compared to base R. For more details examine the code yourself or look at the documentation for `dapply` which works very similar (the only difference really is the splitting performed in BY).

BY is used internally in `collap` (collapse's main aggregation command) for functions that are not 

Fast Statistical Functions.

Value

`X` where `FUN` was applied to every column split by `g`.

See Also

dapply, collap, Fast Statistical Functions, Data Transformations, Collapse Overview

Examples

```r
v <- iris$Sepal.Length  # A numeric vector
f <- iris$Species        # A factor. Vectors/lists will internally be converted to factor

## default vector method
BY(v, f, sum)            # Sum by species
BY(v, f, scale)          # Scale by species (please use fscale instead)
BY(v, f, scale, use.g.names = FALSE)  # Omitting auto-generated names
BY(v, f, quantile)       # Species quantiles: by default stacked
BY(v, f, quantile, expand.wide = TRUE)  # Wide format

## matrix method
m <- qM(num_vars(iris))  # Also return as matrix
BY(m, f, sum)            # Also return as data.frame ... also works for computations below
BY(m, f, scale)
BY(m, f, scale, use.g.names = FALSE)
BY(m, f, quantile)
BY(m, f, quantile, expand.wide = TRUE)
BY(m, f, quantile, expand.wide = TRUE, # Return as list of matrices
    return = "list")

## data.frame method
BY(num_vars(iris), f, sum)  # Also returns a data.frame
BY(num_vars(iris), f, sum, return = 2) # Return as matrix ... also works for computations below
BY(num_vars(iris), f, scale)
BY(num_vars(iris), f, scale, use.g.names = FALSE)
BY(num_vars(iris), f, quantile)
BY(num_vars(iris), f, quantile, expand.wide = TRUE)
BY(num_vars(iris), f, quantile, expand.wide = TRUE, return = "list")

## grouped tibble method

```
library(dplyr)
giris <- group_by(iris, Species)
giris %>% BY(sum) # Compute sum
  giris %>% BY(sum, use.g.names = TRUE, # Use row.names and
              keep.group_vars = FALSE) # remove 'Species' and groups attribute
  giris %>% BY(sum, return = "matrix") # Return matrix
  giris %>% BY(sum, return = "matrix", # Matrix with row.names
              use.g.names = TRUE)
  giris %>% BY(log) # Take logs
  giris %>% BY(log, use.g.names = TRUE, # Use row.names and
              keep.group_vars = FALSE) # remove 'Species' and groups attribute
  giris %>% BY(quantile) # Compute quantiles (output is stacked)
  giris %>% BY(quantile, # Much better, also keeps 'Species'
               expand.wide = TRUE)

dapply

Data Apply

Description

dapply efficiently applies functions to columns or rows of matrices and data frame’s and (default) returns an object of the same type and with the same attributes, or converts to the other type. A simple parallelism is also available.

Usage

dapply(X, FUN, ..., MARGIN = 2, parallel = FALSE, mc.cores = 1L,
       return = c("same", "matrix", "data.frame"), drop = TRUE)

Arguments

X  a matrix or data frame.
FUN  a function, can be scalar- or vector-valued.
... further arguments to FUN.
MARGIN  integer. The margin which FUN will be applied over. Default 2 indicates columns while 1 indicates rows. See also Details.
parallel  logical. TRUE implements simple parallel execution by internally calling parallel::mclapply instead of base::lapply.
mc.cores  integer. Argument to parallel::mclapply indicating the number of cores to use for parallel execution. Can use parallel::detectCores() to select all available cores. See also ?parallel::mclapply.
return  an integer or string indicating the type of object to return. The default 1 -"same" returns the same object type (i.e. passing a matrix returns a matrix and passing a data frame returns a data frame). 2 -"matrix" always returns the output as matrix and 3 -"data.frame" always returns a data frame.
drop  logical. If the result has only one row or one column, drop = TRUE will drop dimensions and return a (named) atomic vector.
Details

dapply is an efficient command to apply functions to rows or columns of data without loosing information (attributes) about the data or changing the classes or format of the data. It is principally an efficient wrapper around base::lapply and works as follows:

- Save the attributes of X.
- If MARGIN = 2 (columns), convert matrices to plain lists of columns using mctl and remove all attributes from data frames.
- If MARGIN = 1 (rows), convert matrices to plain lists of rows using mrtl. For data frames remove all attributes, efficiently convert to matrix using do.call(rbind,X) and also convert to list of rows using mrtl.
- Call base::lapply or parallel::mclapply on these plain lists (which is faster than calling lapply on an object with attributes).
- depending on the requested output type, use base::matrix, base::unlist or do.call(cbind,...) to convert the result back to a matrix or list of columns.
- modify the relevant attributes accordingly and efficiently attach to the object again (no further checks).

This performance gain from working with plain lists makes dapply not much slower than calling lapply itself on a data frame. Because of the conversions involved, row-operations require some memory, but are still faster than base::apply.

Value

X where FUN was applied to every row or column.

See Also

BY, collap, Fast Statistical Functions, Data Transformations, Collapse Overview

Examples

dapply(mtcars, log) # Take natural log of each variable
dapply(mtcars, log, return = "matrix") # Return as matrix
m <- as.matrix(mtcars)
dapply(m, log) # Same thing
dapply(m, log, return = "data.frame") # Return data frame from matrix
dapply(mtcars, sum); dapply(m, sum) # Computing sum of each column, return as vector
dapply(mtcars, sum, drop = FALSE) # This returns a data.frame of 1 row
dapply(mtcars, sum, MARGIN = 1) # Compute row-sum of each column, return as vector
dapply(m, sum, MARGIN = 1) # Same thing for matrices, faster than apply(m, 1, sum)
dapply(m, sum, MARGIN = 1, drop = FALSE) # Gives matrix with one column
dapply(m, quantile, MARGIN = 1) # Compute row-quantiles
dapply(m, quantile) # Column-quantiles
dapply(mtcars, quantile, MARGIN = 1) # Same for data frames, output is also a data.frame
dapply(mtcars, quantile)

# Let's now take a more complex classed object, like a dplyr grouped tibble
library(dplyr)
**dapply**

```r
gmtcars <- group_by(mtcars,cyl,vs,am)
dapply(gmtcars, log) # Still gives a grouped tibble back
dapply(gmtcars, log, MARGIN = 1)
dapply(gmtcars, quantile, MARGIN = 1) # Also works for quantiles
dapply(gmtcars, log, return = "matrix") # Output as matrix
```

---

### descr

**Detailed Statistical Description of Data Frame**

**Description**

descr offers concise description of each variable in a data.frame. It is built as a wrapper around qsu, but by default also computes frequency tables with percentages for categorical variables, and quantiles and the number of distinct values for numeric variables (next to the mean, sd, min, max, skewness and kurtosis computed by qsu).

**Usage**

```r
descr(X, Ndistinct = TRUE, higher = TRUE, table = TRUE, 
Qprobs = c(0.01, 0.05, 0.25, 0.5, 0.75, 0.95, 0.99),
cols = NULL, label.attr = "label", ...)
```

```r
## S3 method for class 'descr'
print(x, n = 6, perc = TRUE, summary = TRUE, ...)
## S3 method for class 'descr'
as.data.frame(x, ...)
```

**Arguments**

- **X**
  - a data.frame or list of atomic vectors. Atomic vectors, matrices or arrays can be passed but will first be coerced to data.frame using qDF.

- **Ndistinct**
  - logical. TRUE (default) computes the number of distinct values on all variables using fNdistinct.

- **higher**
  - logical. Argument is passed down to qsu: TRUE (default) computes the skewness and the kurtosis.

- **table**
  - logical. TRUE (default) calls table on all categorical variables (excluding Date variables).

- **Qprobs**
  - probabilities for quantiles to compute on numeric variables, passed down to quantile. If something non-numeric is passed (i.e. NULL, FALSE, NA, "" etc.), no quantiles are computed.

- **cols**
  - select columns to describe using column names, indices or a function (i.e. is.numeric).

- **label.attr**
  - character. The name of a label attribute to display for each variable (if variables are labeled).

- **...**
  - other arguments passed to qsu.default.
descr\n
\section*{Details}

descr was heavily inspired by Hmisc::describe, but computes about 10x faster. The performance is comparable to base::summary. descr was built as a wrapper around qsu, to enrich the set of statistics computed by qsu for both numeric and categorical variables.

qsu itself is yet about 10x faster than descr, and is optimized for grouped, panel-data and weighted statistics. It is possible to also compute grouped, panel-data and/or weighted statistics with descr by passing group-ids to g, panel-ids to pid or a weight vector to w. These arguments are handed down to qsu.default and only affect the statistics natively computed by qsu, i.e. passing a weight vector produces a weighted mean, sd, skewness and kurtosis but not weighted quantiles.

The list-object returned from descr can be converted to a tidy data.frame using as.data.frame. This representation will not include frequency tables computed for categorical variables, and the method cannot handle arrays of statistics (applicable when g or pid arguments are passed to descr, in that case as.data.frame.descr will throw an appropriate error).

\section*{Value}

A 2-level nested list, the top-level containing the statistics computed for each variable, which are themselves stored in a list containing the class, the label, the basic statistics and quantiles / tables computed for the variable. The object is given a class 'descr' and also has the number of observations in the dataset attached as an 'N' attribute, as well as an attribute 'arstat' indicating whether the object contains arrays of statistics.

\section*{See Also}

qsu, pwcor, Fast Statistical Functions, Collapse Overview

\section*{Examples}

\begin{verbatim}
## Standard Use
descr(iris)
descr(wlddev)
descr(GGDC10S)

as.data.frame(descr(wlddev))

## Passing Arguments down to qsu: For Panel-Data Statistics
descr(iris, pid = iris$species)
descr(wlddev, pid = wlddev$iso3c)

## Grouped Statistics
\end{verbatim}
descr(iris, g = iris$species)
descr(GGDC10S, g = GGDC10S$Region)

**extract-list**

Find and Extract / Subset List Elements

**Description**

A suite of functions to subset or extract from (potentially complex) lists and list-like structures. Subsetting may occur according to certain data types, using identifier functions, element names or regular expressions to search the list for certain objects.

- **atomic_elem** and **list_elem** are non-recursive functions to extract and replace the atomic and sub-list elements at the top-level of the list tree.
- **reg_elem** is the recursive equivalent of **atomic_elem** and returns the 'regular' part of the list - with atomic elements in the final nodes. See **is.regular** and **is.unlistable**. **irreg_elem** returns all the non-regular elements (i.e. call and terms objects, formulas, etc...). See Examples.
- **get_elem** returns the part of the list responding to either an identifier function, regular expression or exact element names, or indices applied to all final objects. **has_elem** checks for the existence of the searched element and returns TRUE if a match is found. See Examples.

**Usage**

```r
## Non-recursive (top-level) subsetting and replacing
atomic_elem(l, return = c("sublist","names","indices","named_indices"),
keep.class = FALSE)
atomic_elem(l) <- value
list_elem(l, return = c("sublist","names","indices","named_indices"),
keep.class = FALSE)
list_elem(l) <- value

## Recursive separation of regular (atomic) and irregular (non-atomic) parts
reg_elem(l, recursive = TRUE, keep.tree = FALSE, keep.class = FALSE)
irreg_elem(l, recursive = TRUE, keep.tree = FALSE, keep.class = FALSE)

## Extract elements using a function or regular expression
get_elem(l, elem, recursive = TRUE, DF.as.list = TRUE, keep.tree = FALSE, keep.class = FALSE, regex = FALSE, ...)

## Check for the existence of elements
has_elem(l, elem, recursive = TRUE, DF.as.list = TRUE, regex = FALSE, ...)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>a list.</td>
</tr>
<tr>
<td>value</td>
<td>a list of the same length as the extracted subset of l.</td>
</tr>
<tr>
<td>elem</td>
<td>a function returning TRUE or FALSE when applied to elements of l, or a character vector of element names or regular expressions (if regex = TRUE). get_elem also supports a vector or indices which will be used to subset all final objects.</td>
</tr>
</tbody>
</table>
| return   | an integer or string specifying what to return. The options are:  
|          | 1. "sublist"  
|          | 2. "names"  
|          | 3. "indices"  
|          | 4. "named_indices" |

Note: replacement functions currently only elements, not element names or indices (ordering). However element names are replaced together with the data.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive</td>
<td>logical. should the list search be recursive (i.e. go through all the elements), or just at the top-level?</td>
</tr>
<tr>
<td>DF.as.list</td>
<td>logical. treat data.frame’s like (sub-)lists or like atomic elements?</td>
</tr>
<tr>
<td>keep.tree</td>
<td>logical. TRUE always returns the entire list tree leading up to all matched results, while FALSE drops the top-level part of the tree if possible.</td>
</tr>
<tr>
<td>keep.class</td>
<td>logical. for classed objects: Should the class be retained?</td>
</tr>
<tr>
<td>regex</td>
<td>logical. should regular expression search be used on the list names, or only exact matches?</td>
</tr>
</tbody>
</table>

Details

A list is made up of regular and irregular elements. I defined regular elements as all elements that are either atomic or a list (see is.regular). reg_elem with recursive = TRUE therefore extracts the subset of the list tree leading up to atomic elements in the final nodes. This part of the list tree is unlistable - calling is.unlistable(reg_elem(l)) will be TRUE for all lists l. Conversely, all elements left behind by reg_elem will be picked up by irreg_elem (if available). Thus is.unlistable(irreg_elem(l)) is always FALSE for lists with irregular elements (otherwise irreg_elem returns an empty list).

If keep.tree = TRUE, reg_elem, irreg_elem and get_elem always return the entire list tree, but cut off all of the branches not leading to the desired result. If keep.tree = FALSE, top-level parts of the tree are omitted so far this is possible. For example in a nested list with three levels and one data-matrix in one of the final branches, get_elem(l,is.matrix,keep.tree = TRUE) will return a list (lres) of depth 3, from which the matrix can be accessed as lres[[1]][[1]][[1]]. This however does not make much sense. get_elem(l,is.matrix,keep.tree = FALSE) will therefore figure out that it can drop the entire tree and return just the matrix. keep.tree = FALSE makes additional optimizations if matching elements are at far-apart corners in a nested structure, by only
between-fwithin-B-W

preserving the hierarchy if elements are above each other on the same branch. Thus for a list
l <- list(list(2, list("a", 1)), list(1, list("b", 2))) calling get_elem(l, is.character)
will just return list("a", "b").

See Also

List Processing, Collapse Overview

Examples

l <- list(list(2, list("a", 1)), list(1, list("b", 2)))
has_elem(l, is.logical)
has_elem(l, is.character)
get_elem(l, is.character)
get_elem(l, is.character, keep.tree = TRUE)

l <- lm(mpg ~ cyl + vs, data = mtcars)
str(reg_elem(l))
str(irreg_elem(l))
get_elem(l, is.matrix)
get_elem(l, "residuals")
get_elem(l, "fit", regex = TRUE)
has_elem(l, "tol")
get_elem(l, "tol")

fbetween-fwithin-B-W  Fast Between (Averaging) and Within (Centering) Transformations

Description

fbetween and fwithin are S3 generics to efficiently obtain between-transformed (averaged) or
within-transformed (demeaned) data. These operations can be performed groupwise and/or weighted.
B and W are wrappers around fbetween and fwithin representing the 'between-operator' and the
'within-operator'. B / W provide more flexibility than fbetween / fwithin when applied to data
frames (i.e. column subsetting, formula input, auto-renaming and id-variable-preservation capabil-
ities...), but are otherwise identical.

(fbetween and fwithin are simple programmers functions in style of the Fast Statistical Functions
while B and W are more practical to use in regression formulas or for ad-hoc computations on data
frames.)

Usage

fbetween(x, ...)
fwithin(x, ...)
B(x, ...)
W(x, ...)
### Default S3 method:
fbetween(x, g = NULL, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### Default S3 method:
fwithin(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, ...)
### Default S3 method:
B(x, g = NULL, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### Default S3 method:
W(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, ...)

### S3 method for class 'matrix'
fbetween(x, g = NULL, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### S3 method for class 'matrix'
fwithin(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, ...)
### S3 method for class 'matrix'
B(x, g = NULL, w = NULL, na.rm = TRUE, fill = FALSE, stub = "B.", ...)
### S3 method for class 'matrix'
W(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, stub = "W.", ...)

### S3 method for class 'data.frame'
fbetween(x, g = NULL, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### S3 method for class 'data.frame'
fwithin(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, ...)
### S3 method for class 'data.frame'
B(x, by = NULL, w = NULL, cols = is.numeric, na.rm = TRUE,
   fill = FALSE, stub = "B.", keep.by = TRUE, keep.w = TRUE, ...)
### S3 method for class 'data.frame'
W(x, by = NULL, w = NULL, cols = is.numeric, na.rm = TRUE,
   mean = 0, stub = "W.", keep.by = TRUE, keep.w = TRUE, ...)

# Methods for compatibility with plm:

### S3 method for class 'pseries'
fbetween(x, effect = 1L, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### S3 method for class 'pseries'
fwithin(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, ...)
### S3 method for class 'pseries'
B(x, effect = 1L, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### S3 method for class 'pseries'
W(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, ...)

### S3 method for class 'pdata.frame'
fbetween(x, effect = 1L, w = NULL, na.rm = TRUE, fill = FALSE, ...)
### S3 method for class 'pdata.frame'
fwithin(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, ...)
### S3 method for class 'pdata.frame'
B(x, effect = 1L, w = NULL, cols = is.numeric, na.rm = TRUE,
   fill = FALSE, stub = "B.", keep.ids = TRUE, keep.w = TRUE, ...)
## S3 method for class 'pdata.frame'
W(x, effect = 1L, w = NULL, cols = is.numeric, na.rm = TRUE, mean = 0, stub = "W.", keep.ids = TRUE, keep.w = TRUE, ...)

# Methods for compatibility with dplyr:

## S3 method for class 'grouped_df'
fbetween(x, w = NULL, na.rm = TRUE, fill = FALSE, keep.group_vars = TRUE, keep.w = TRUE, ...)

## S3 method for class 'grouped_df'
within(x, w = NULL, na.rm = TRUE, mean = 0, keep.group_vars = TRUE, keep.w = TRUE, ...)

## S3 method for class 'grouped_df'
B(x, w = NULL, na.rm = TRUE, fill = FALSE, stub = "B.", keep.group_vars = TRUE, keep.w = TRUE, ...)

## S3 method for class 'grouped_df'
W(x, w = NULL, na.rm = TRUE, mean = 0, stub = "W.", keep.group_vars = TRUE, keep.w = TRUE, ...)

Arguments

x | a numeric vector, matrix, data.frame, panel-series (plm::pseries), panel-data.frame (plm::pdata.frame) or grouped tibble (dplyr::grouped_df).
g | a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.
by | B and W data.frame method: Same as g, but also allows one- or two-sided formulas i.e. ~ group1 or var1 + var2 ~ group1 + group2. See Examples.
w | a numeric vector of (non-negative) weights. B/W data.frame and pdata.frame methods also allow a one-sided formula i.e. ~ weightcol. The grouped_df (dplyr) method supports lazy-evaluation. See Examples.
cols | data.frame method: Select columns to center/average using a function, column names or indices. Default: All numeric variables. Note: cols is ignored if a two-sided formula is passed to by.
na.rm | logical. skip missing values in x when computing averages. If na.rm = FALSE and a NA or NaN is encountered, the average for that group will be NA, and all data points belonging to that group will also be NA.
effect | plm methods: Select which panel identifier should be used as grouping variable. 1L means first variable in the plm:::index, 2L the second etc. if more than one integer is supplied, the corresponding index-variables are interacted.
stub | a prefix or stub to rename all transformed columns. FALSE will not rename columns.
fill | option to fbetween/B: Logical. TRUE will overwrite missing values in x with the respective average. By default missing values in x are preserved.
mean | option to within/W: The mean to center on, default is 0, but a different mean can be supplied and will be added to the data after the centering is performed. A special option when performing grouped centering is mean = "overall.mean". In that case the overall mean of the data will be added after subtracting out group means.
keep.by, keep.ids, keep.group_vars

B and W data.frame, pdata.frame and grouped_df methods: Logical. Retain grouping / panel-identifier columns in the output. For data frames this only works if grouping variables were passed in a formula.

keep.w

B and W data.frame, pdata.frame and grouped_df methods: Logical. Retain column containing the weights in the output. Only works if w is passed as formula / lazy-expression.

... arguments to be passed to or from other methods.

Details

Without groups, fbetween/B replaces all data points in x with their mean or weighted mean (if w is supplied). Similarly fwithin/W subtracts the mean from all data points i.e. centers the data on the mean.

With groups supplied to g, the replacement / centering performed by fbetween/B | fwithin/W becomes groupwise. I like to think of this in terms of panel data: If x is a vector in such a dataset, x_{it} denotes a single data-point belonging to group i in time-period t (t need not be a time-period). Then \( x_{i.} \) denotes x, averaged over t. fbetween/B now returns \( x_{i.} \) and fwithin/W returns \( x - x_{i.} \). Thus for any data x and any grouping vector g: \( B(x, g) + W(x, g) = x_{i.} + x - x_{i.} = x \). In terms of variance, fbetween/B only retains the variance between group averages, while fwithin/W, by subtracting out group means, only retains the variance within those groups.

The data replacement performed by fbetween/B can keep (default) or overwrite missing values (option fill = TRUE) in x. fwithin/W can center data simply (default), or add back a mean after centering (option mean = value), or add the overall mean in groupwise computations (option mean = "overall.mean"). Let \( x_{..} \) denote the overall mean of x, then fwithin/W with mean = "overall.mean" returns \( x - x_{i.} + x_{..} \) instead of \( x - x_{i.} \). This is useful to get rid of group-differences but preserve the overall level of the data (as simple groupwise centering will set the overall mean of the data to 0, or any other arbitrary value passed to mean). In regression analysis, centering with mean = "overall.mean" will only change the constant term. See Examples.

Value

fbetween/B returns x with every element replaced by its (groupwise) mean \( (x_{i.}) \). fwithin/W returns x where every element was subtracted its (groupwise) mean \( (x - x_{i.} \) or \( x - x_{i.} + \text{mean} \) or \( x - x_{i.} + x_{..} \)). See Details.

See Also

fHDbetween/HDB and fHDwithin/HDW, fscale/STD, TRA, Data Transformations, Collapse Overview

Examples

```r
## Simple centering and averaging
fbetween(mtcars)
B(mtcars)
fwithin(mtcars)
W(mtcars)
```
### Description

`fdiff` and `fgrowth` are S3 generics to compute (sequences of) suitably lagged / leaded and iterated differences and growth rates / log-differences, respectively. D and G are wrappers around `fdiff` and `fgrowth` representing the 'difference-operator' and the 'growth-operator'. D / G provide more flexibility than `fdiff` / `fgrowth` when applied to data frames, but are otherwise identical. (`fdiff` and `fgrowth` are programmers functions in style of the Fast Statistical Functions while D & G are more practical to use in regression formulas or for computations on data frames.)

### Usage

```r
fdiff(x, n = 1, diff = 1, ...)
fgrowth(x, n = 1, diff = 1, ...)
D(x, n = 1, diff = 1, ...)
```
G(x, n = 1, diff = 1, ...)  

## Default S3 method:  
fdiff(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
stubs = TRUE, ...)  

## Default S3 method:  
fgrowth(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
logdiff = FALSE, stubs = TRUE, ...)  

## Default S3 method:  
D(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
stubs = TRUE, ...)  

## Default S3 method:  
G(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
logdiff = FALSE, stubs = TRUE, ...)  

## S3 method for class 'matrix'  
fdiff(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
stubs = TRUE, ...)  

## S3 method for class 'matrix'  
fgrowth(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
logdiff = FALSE, stubs = TRUE, ...)  

## S3 method for class 'matrix'  
D(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
stubs = TRUE, ...)  

## S3 method for class 'matrix'  
G(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
logdiff = FALSE, stubs = TRUE, ...)  

## S3 method for class 'data.frame'  
fdiff(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
stubs = TRUE, ...)  

## S3 method for class 'data.frame'  
fgrowth(x, n = 1, diff = 1, g = NULL, t = NULL, fill = NA,  
logdiff = FALSE, stubs = TRUE, ...)  

## S3 method for class 'data.frame'  
D(x, n = 1, diff = 1, by = NULL, t = NULL, cols = is.numeric,  
fill = NA, stubs = TRUE, keep.ids = TRUE, ...)  

## S3 method for class 'data.frame'  
G(x, n = 1, diff = 1, by = NULL, t = NULL, cols = is.numeric,  
fill = NA, logdiff = FALSE, stubs = TRUE, keep.ids = TRUE, ...)  

# Methods for compatibility with plm:  

## S3 method for class 'pseries'  
fdiff(x, n = 1, diff = 1, fill = NA, stubs = TRUE, ...)  

## S3 method for class 'pseries'  
fgrowth(x, n = 1, diff = 1, fill = NA, logdiff = FALSE, stubs = TRUE, ...)  

## S3 method for class 'pseries'
D(x, n = 1, diff = 1, fill = NA, stubs = TRUE, ...)
## S3 method for class 'pseries'
G(x, n = 1, diff = 1, fill = NA, logdiff = FALSE, stubs = TRUE, ...)
## S3 method for class 'pdata.frame'

fdiff(x, n = 1, diff = 1, fill = NA, stubs = TRUE, ...)
## S3 method for class 'pdata.frame'

fgrowth(x, n = 1, diff = 1, fill = NA, logdiff = FALSE, stubs = TRUE, ...)
## S3 method for class 'pdata.frame'

D(x, n = 1, diff = 1, cols = is.numeric, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'pdata.frame'

G(x, n = 1, diff = 1, cols = is.numeric, fill = NA, logdiff = FALSE, stubs = TRUE, keep.ids = TRUE, ...)

# Methods for compatibility with dplyr:

## S3 method for class 'grouped_df'

fdiff(x, n = 1, diff = 1, t = NULL, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'grouped_df'

fgrowth(x, n = 1, diff = 1, t = NULL, fill = NA, logdiff = FALSE, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'grouped_df'

D(x, n = 1, diff = 1, t = NULL, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'grouped_df'

G(x, n = 1, diff = 1, t = NULL, fill = NA, logdiff = FALSE, stubs = TRUE, keep.ids = TRUE, ...)

Arguments

x a numeric vector, matrix, data.frame, panel-series (plm::pseries), panel-data.frame (plm::pdata.frame) or grouped tibble (dplyr::grouped_df).

n a integer vector indicating the number of lags or leads.

diff a vector of integers > 1 indicating the order of differencing / taking growth rates or log-differences.

g a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.

by data.frame method: Same as g, but also allows one- or two-sided formulas i.e. ~ group1 or var1 + var2 ~ group1 + group2. See Examples.

t same input as g, to indicate the time-variable. For safe computation of differences/growth rates on unordered time-series and panels. Note: data.frame method also allows name, index or one-sided formula i.e. ~time. grouped_df method also allows lazy-evaluation i.e. time (no quotes).

cols data.frame method: Select columns to difference/compute growth rates using a function, column names or indices. Default: All numeric variables. Note: cols is ignored if a two-sided formula is passed to by.
fill

value to insert when vectors are shifted. Default is NA.

logdiff

logical. compute log-differences instead of exact growth rates. See Details.

stubs

logical. TRUE will rename all differenced columns by adding a prefix "LnDdiff." / "FnDdiff." and a prefix "LnGdiff." / "FnGdiff." for growth rates.

keep.ids

data.frame / pdata.frame / grouped_df methods: Logical. Drop all panel-identifiers from the output (which includes all variables passed to by or t). Note: For panel-data.frame’s and grouped tibbles identifiers are dropped, but the ‘index’ / ‘groups’ attributes are kept.

... arguments to be passed to or from other methods.

Details

By default, fdiff/D|fgrowth/G return x with all columns differenced | converted to growth rates. Differences are computed as repeat(diff){x[i] -x[i-\n]}, growth rates as repeat(diff){(x[i] -x[i-\n])/(x[i-\n])*100} and log-differences as repeat(diff){(log(x[i]) -log(x[i-\n]))*100}. Setting diff = 2 thus returns differences of differences | growth rates of growth rates etc... and setting n = 2 returns simple differences computed by subtracting twice-lagged x from x. It is also possible to compute forward differences | growth rates by passing negative n values. n also supports sequences of integers (lags), and diff supports positive sequences of integers (differences):

If more than one value is passed to n and/or diff, the data is expanded-wide as follows: If x is an atomic vector or time-series, a (time-series) matrix is returned with columns ordered first by lag, then by difference. If x is a matrix or data.frame, each column is expanded in like manor such that the output has ncol(x)*length(n)*length(diff) columns ordered first by column name, then by lag, then by difference.

With groups/panel-identifiers supplied to g/by, fdiff/D|fgrowth/G efficiently compute panel-differences | growth rates by inserting fill elements in the right places. If t is left empty, the data needs to be ordered such that all values belonging to a group are consecutive and in the right order. It is not necessary that the groups themselves occur in the right order. If time-variable(s) are supplied to t, the panel is fully identified and differences | growth rates can be securely computed even if the data is completely unordered (in that case data is shifted around and fill values are inserted in such a way that if the data were sorted afterwards the result would be identical to computing on sorted data). Internally this works by using the grouping- and time-variables to create an ordering and then accessing the panel-vector(s) through this ordering. If the data is just a bit unordered, such computations are nearly as fast as computations on ordered data (without t), however, if the data is very unordered, it can take significantly longer. Since most panel-data come perfectly or pretty ordered, I recommend always supplying t to be on the safe-side.

It is also possible to compute differences | growth rates on unordered vectors / time-series (thus utilizing t but leaving g/by empty).

The methods applying to plm objects (panel-series and panel-data.frames) automatically utilize the panel-identifiers attached to these objects and thus securely compute fully identified panel-differences. If these objects have > 2 panel-identifiers attached to them, the last identifier is assumed to be the time-variable, and the others are taken as grouping-variables and interacted.

Value

fdiff/D returns x differenced diff times using lags n of itself. fgrowth/G returns x where the
growth rate or log-difference was taken diff times using lags n of itself. Computations can be grouped by g/by and/or ordered by t. See Details and Examples.

See Also
flag/L/F, Time-Series and Panel-Series, Collapse Overview

Examples

```r
## Simple Time-Series: Airpassengers
D(AirPassengers) # 1st difference, same as fdiff(AirPassengers)
D(AirPassengers,-1) # forward difference
G(AirPassengers) # growth rate, same as fgrowth(AirPassengers)
G(AirPassengers, logdiff = TRUE) # log-difference
D(AirPassengers,1,2) # second difference
G(AirPassengers,1,2) # growth rate of growth rate
D(AirPassengers,12) # seasonal difference (data is monthly)
G(AirPassengers,12) # seasonal growth rate (data is monthly)

D(AirPassengers,-2:2,1:3) # sequence of leaded/lagged and iterated differences

# let's do some visual analysis
plot(AirPassengers) # plot the series - seasonal pattern is evident
plot(stl(AirPassengers, "periodic")) # Seasonal decomposition
plot(D(AirPassengers,c(1,12),1:2)) # plotting ordinary and seasonal first and second differences
plot(G(AirPassengers,c(1,12),1:2)) # same using growth rates
plot(stl(window(G(AirPassengers,12), # Taking seasonal growth rate removes most seasonal variation
                1950), "periodic"))

## Time-Series Matrix of 4 EU Stock Market Indicators, recorded 260 days per year
plot(G(EuStockMarkets,c(0,260))) # Plot series and annual growth rates
summary(lm(L260G1.DAX ~., G(EuStockMarkets,260))) # Annual growth rate of DAX regressed on the
# growth rates of the other indicators

## World Development Panel Data
head(fgrowth(num_vars(wlddev), 1, 1, # Computes growth rates of numeric variables
             wlddev$country, wlddev$year)) # fdiff/fgrowth require externally inputs...
head(G(wlddev, 1, 1, ~country, ~year)) # Growth of numeric variables, id's attached
head(G(wlddev, 1, 1, ~country)) # Without t: Works because data is ordered
head(G(wlddev, 1, 1, PCGDP + LIFEEX ~ country, ~year)) # Growth of GDP per Capita & Life Expectancy
head(G(wlddev, 0:1, 1, ~country, ~year, cols = 9:10)) # Same, also retaining original series
head(G(wlddev, 0:1, 1, ~country, ~year, 9:10, # Dropping id columns
     keep.ids = FALSE))

# Dynamic Panel-Data Models:
summary(lm(G(PCGDP,1,1,iso3c,year) ~ L(PCGDP,1,iso3c,year) + # GDP growth regressed on it's lagged level
         G(LIFEEX,1,1,iso3c,year), data = wlddev))

g = qF(wlddev$country) # Omitting t and precomputing g allows for a
summary(lm(G(PCGDP,1,1,g) ~ L(PCGDP,1,g) + # bit more parsimonious specification
           G(LIFEEX,1,1,g), wlddev))
```
summary(lm(G1.PCGDP ~., # Now adding level and lagged level of
L(G(wlddev,0:1,1, ~ country, ~year,9:10),0:1, # LIFEEX and lagged growth rates
~ country, ~year, keep.ids = FALSE)[-1]))

## Using plm can make things easier, but avoid attaching or 'with' calls:
pwlddev <- plm::pdata.frame(wlddev, index = c("country","year"))
head(G(pwlddev, 0:1, 1, 9:10)) # Again growth rates of LIFEEX and PCGDP
PCGDP <- pwlddev$PCGDP # A panel-Series of GDP per Capita
D(PCGDP) # Differencing the panel series.
summary(lm(G1.PCGDP ~., # Running the dynamic model again ->
data = L(G(pwlddev,0:1,1,9:10),0:1,
keep.ids = FALSE)[-1])))

# One could be tempted to also do something like this, but THIS DOES NOT WORK!!!:
# lm drops the attributes (-r with(pwlddev, PCGDP) drops attr. so G.default and L.matrix are used)
summary(lm(G(PCGDP) ~ L(G(PCGDP,0:1)) + L(G(LIFEEX,0:1),0:1), pwlddev))

# To make it work, one needs to create pseries (note: attach(pwlddev) also won't work)
LIFEEX <- pwlddev$LIFEEX
summary(lm(G(PCGDP) ~ L(G(PCGDP,0:1)) + L(G(LIFEEX,0:1),0:1)))) # THIS WORKS !

## Using dplyr:
library(dplyr)
wlddev %>% group_by(country) %>% # Adding a first and second difference
  select(PCGDP,LIFEEX) %>% D(0:1,1:2) # Also using t (safer)
wlddev %>% group_by(country) %>%
  select(year,PCGDP,LIFEEX) %>% D(0:1,1:2,year) # Growth rates, dropping id's
wlddev %>% group_by(country) %>%
  select(year,PCGDP,LIFEEX) %>% G(0:1,1:2,year, keep.ids = FALSE)

ffirst-flast

**Fast (Grouped) First and Last Value for Matrix-Like Objects**

**Description**

`ffirst` and `flast` are S3 generic functions that (column-wise) returns the first and last values in `x`, (optionally) grouped by `g`. The `TRA` argument can further be used to transform `x` using its (groupwise) first and last values.

**Usage**

```r
ffirst(x, ...)
flast(x, ...)
```

## Default S3 method:
`ffirst(x, g = NULL, TRA = NULL, na.rm = TRUE,
use.g.names = TRUE, ...)`
Arguments

**x**

A vector, matrix, data.frame or grouped tibble (`dplyr::grouped_df`).

**g**

A factor, `GRP` object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a `GRP` object) used to group `x`.

**TRA**

An integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "-" | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "*" | 9 - "%%" | 10 - "-%%". See `TRA`.

**na.rm**

Logical. Skip missing values and choose the first / last non-missing value i.e. if the first (1) / last (n) value is NA, take the second (2) / second-to-last (n-1) value etc...

**use.g.names**

Make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.

**drop**

Matrix and data.frame method: drop dimensions and return an atomic vector if `g = NULL` and `TRA = NULL`.

**keep.group_vars**

grouped_df method: Logical. FALSE removes grouping variables after computation.

**...**

Arguments to be passed to or from other methods.
Value

`ffirst` returns the first value in x, grouped by g, or (if TRA is used) x transformed by its first value, grouped by g. Similarly `flast` returns the last value in x, ...

See Also

Fast Statistical Functions, Collapse Overview

Examples

```r
## default vector method
ffirst(airquality$Ozone) # Simple first value
ffirst(airquality$Ozone, airquality$Month) # Grouped first value
ffirst(airquality$Ozone, airquality$Month, na.rm = FALSE) # Grouped first, but without skipping initial NA's

## data.frame method
ffirst(airquality)
ffirst(airquality, airquality$Month)
ffirst(airquality, airquality$Month, na.rm = FALSE) # Again first Ozone measurement in month 6 is NA

## matrix method
aqm <- qM(airquality)
ffirst(aqm)
ffirst(aqm, airquality$Month) # etc...

## method for grouped tibbles - for use with dplyr
library(dplyr)
airquality %>% group_by(Month) %>% ffirst
airquality %>% group_by(Month) %>% select(Ozone) %>% ffirst(na.rm = FALSE)

# Note: All examples generalize to flast!
```

fFtest

*Fast F-test of Linear Models (with Factors)*

Description

`fFtest` computes an R-squared based F-test for the exclusion of the variables in exc, where the full (unrestricted) model is defined by variables supplied to both exc and X. The test is efficient and designed for cases where both exc and X may contain multiple factors and continuous variables.

Usage

```r
fFtest(y, exc, X = NULL, full.df = TRUE, ...)
```
Arguments

- **y**: a numeric vector: The dependent variable.
- **exc**: a numeric vector, factor, numeric matrix or list / data.frame of numeric vectors and/or factors: Variables to test / exclude.
- **X**: a numeric vector, factor, numeric matrix or list / data.frame of numeric vectors and/or factors: Covariates to include in both the restricted (without exc) and unrestricted model. If left empty (X = NULL), the test amounts to the F-test of the regression of y on exc.
- **full.df**: logical. If TRUE (default), the degrees of freedom are calculated as if both restricted and unrestricted models were estimated using lm() (i.e. as if factors were expanded to matrices of dummies). FALSE only uses one degree of freedom per factor.
- **...**: other arguments passed to lfe::demeanlist, the workhorse function underlying fHDwithin.

Details

Factors and continuous regressors are efficiently projected out using fHDwithin, and the option full.df regulates whether a degree of freedom is subtracted for each used factor level (equivalent to dummy-variable estimator / expanding factors), or only one degree of freedom per factor (fixed-effects estimation / treating factors as variables). The test automatically removes missing values and considers only the complete cases of y, exc and X. Unused factor levels in exc and X are dropped.

Value

A 5 x 3 numeric matrix of statistics. The columns contain statistics:

1. the R-squared of the model
2. the numerator degrees of freedom i.e. the number of variables (k) and used factor levels if full.df = TRUE
3. the denominator degrees of freedom: N - k - 1.
4. the F-statistic
5. the corresponding P-value

The rows show these statistics for:

1. the Full (unrestricted) Model (y ~ exc + X)
2. the Restricted Model (y ~ X)
3. the Exclusion Restriction of exc. The R-squared shown is simply the difference of the full and restricted R-Squared’s, not the R-Squared of the model y ~ exc.

If X = NULL, only a vector of the same 5 statistics testing the model (y ~ exc) is shown.

See Also

fHDbetween/HDB and fHDwithin/HDW, Data Transformations, Collapse Overview
Examples

```r
## We could use fttest as a seasonality test:
fttest(AirPassengers, qF(cycle(AirPassengers)))  # Testing for level-seasonality
fttest(AirPassengers, qF(cycle(AirPassengers)), poly(seq_along(AirPassengers), 3))  # Seasonality test around a cubic trend

## A more classical example with only continuous variables
fttest(mtcars$mpg, mtcars[c("cyl","vs")], mtcars[c("hp","carb")])

## Now encoding cyl and vs as factors
fttest(mtcars$mpg, dapply(mtcars[c("cyl","vs")], qF), mtcars[c("hp","carb")])

## Using iris data: A factor and a continuous variable excluded
fttest(iris$Sepal.Length, iris[4:5], iris[2:3])

## Testing the significance of country-FE in regression of GDP on life expectancy
fttest(wlddev$PCGDP, wlddev$iso3c, wlddev$LIFEEX)

## Ok, country-FE are significant, what about adding time-FE
fttest(wlddev$PCGDP, qF(wlddev$year), wlddev[c("iso3c","LIFEEX")])

# Same test done using lm:
data <- na.omit(get_vars(wlddev, c("iso3c","year","PCGDP","LIFEEX")))
full <- lm(PCGDP ~ LIFEEX + iso3c + qF(year), data)
rest <- lm(PCGDP ~ LIFEEX + iso3c, data)
anova(rest, full)
```
## Default S3 method:
fHDbetween(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)
## Default S3 method:
fHDwithin(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)
## Default S3 method:
HDB(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)
## Default S3 method:
HDW(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)

## S3 method for class 'matrix'
fHDbetween(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)
## S3 method for class 'matrix'
fHDwithin(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, ...)
## S3 method for class 'matrix'
HDB(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, stub = "HDB.", ...)
## S3 method for class 'matrix'
HDW(x, fl, w = NULL, na.rm = TRUE, fill = FALSE, stub = "HDW.", ...)

## S3 method for class 'data.frame'
fHDbetween(x, f1, w = NULL, na.rm = TRUE, fill = FALSE,
  variable.wise = FALSE, ...)
## S3 method for class 'data.frame'
fHDwithin(x, f1, w = NULL, na.rm = TRUE, fill = FALSE,
  variable.wise = FALSE, ...)
## S3 method for class 'data.frame'
HDB(x, f1, w = NULL, cols = is.numeric, na.rm = TRUE, fill = FALSE,
  variable.wise = FALSE, stub = "HDB.", ...)
## S3 method for class 'data.frame'
HDW(x, f1, w = NULL, cols = is.numeric, na.rm = TRUE, fill = FALSE,
  variable.wise = FALSE, stub = "HDW.", ...)

# Methods for compatibility with plm:

## S3 method for class 'pseries'
fHDbetween(x, w = NULL, na.rm = TRUE, fill = TRUE, ...)
## S3 method for class 'pseries'
fHDwithin(x, w = NULL, na.rm = TRUE, fill = TRUE, ...)
## S3 method for class 'pseries'
HDB(x, w = NULL, na.rm = TRUE, fill = TRUE, ...)
## S3 method for class 'pseries'
HDW(x, w = NULL, na.rm = TRUE, fill = TRUE, ...)

## S3 method for class 'pdata.frame'
fHDbetween(x, w = NULL, na.rm = TRUE, fill = TRUE,
  variable.wise = TRUE, ...)
## S3 method for class 'pdata.frame'
fHDwithin(x, w = NULL, na.rm = TRUE, fill = TRUE,
  variable.wise = TRUE, ...)
## S3 method for class 'pdata.frame'
HDB(x, w = NULL, cols = is.numeric, na.rm = TRUE, fill = TRUE, variable.wise = TRUE, stub = "HDB.", ...)  
## S3 method for class 'pdata.frame'
HDW(x, w = NULL, cols = is.numeric, na.rm = TRUE, fill = TRUE, variable.wise = TRUE, stub = "HDW.", ...)

### Arguments

- **x**
a numeric vector, matrix, data.frame, panel-series (plm::pseries) or panel-data.frame (plm::pdata.frame).

- **fl**
a numeric vector, factor, matrix, data.frame or list (which may or may not contain factors). In the data.frame method fl can also be a one-or two sided \texttt{lm()} formula with variables contained in x. Interactions (:) and full interactions (*) are supported! See Examples.

- **w**
a vector of (non-negative) weights. Currently only weighted centering on multiple factors is supported, not weighted linear models.

- **cols**
  
  *data.frame methods:* Select columns to center (partial-out) or predict using column names, indices or a function. Unless specified otherwise all numeric columns are selected. If NULL, all variables are selected.

- **na.rm**
remove missing values from both x and fl. by default rows with missing values in x or fl are removed. In that case an attribute "na.rm" is attached containing the rows removed.

- **fill**
If na.rm = TRUE, fill = TRUE will not remove rows with missing values in x or fl, but fill them with NA's.

- **variable.wise**
  
  *data.frame methods:* Setting variable.wise = TRUE will process each column individually i.e. use all non-missing cases in each column and in fl (fl is only checked for missing values if na.rm = TRUE). This is a lot less efficient but uses all data available in each column.

- **stub**
a prefix / stub to rename all transformed columns. FALSE will not rename columns.

- **...**
further arguments passed to \texttt{lfe::demeanlist} (if fl contains factors), or to / from other methods.

### Details

\texttt{fHDbetween/HDB} and \texttt{fHDwithin/HDW} can be understood as generalizations of \texttt{lfe::demeanlist} to continuous-data and formula input, and more choices dealing with missing values. They are powerful tools for complex high-dimensional linear prediction problems involving large factors and datasets, but can just as well handle ordinary regression problems. Intended areas of use are to efficiently obtain residuals and predicted values from data, and to prepare data for complex linear models involving multiple levels of fixed effects. Such models can now be fitted using \texttt{lm()} on data prepared with \texttt{fHDwithin} / HDW (relying on bootstrapped SE's for inference, or implementing the appropriate corrections). See Examples.

If fl is a vector or matrix, the result are identical to \texttt{lmi.e. fHDbetween / HDB returns fitted(lm(x ~ fl))} and \texttt{fHDwithin / HDW residuals(lm(x ~ fl))}. If fl is a list containing factors, all vari-
ables in x and non-factor variables in fl are centered on these factors using the method of alternating projections implemented by lfe::demeanlist. Afterwards the centered data is regressed on the centered predictors. If fl is just a list of factors, fHDbetween/HDW returns the centered data and fHDwithin/HDB the corresponding means. Take as a most general example a list fl = list(fct1,fct2,...,var1,var2,...) where fcti are factors and vari are continuous variables. The output of fHDwithin/HDW | fHDbetween/HDB will then be identical to calling resid | fitted on lm(x ~ fct1 + fct2 + ... + var1 + var2 + ...). The computations performed by fHDwithin/HDW and fHDbetween/HDB are however much faster and more memory efficient than lm because factors are not passed to stats::model.matrix and expanded to matrices of dummies but projected out using lfe::demeanlist.

The formula interface to the data.frame method (only supported by the operators HDW | HDB) provides ease of use and allows for additional modelling complexity. For example it is possible to project out formulas like HDW(data,~ fct1*var1 + fct2:var3 + var1:var2:var3 + poly(var5,3)*fct5) containing simple (: ) or full (*) interactions of factors with continuous variables or polynomials of continuous variables, and two-or three-way interactions of factors and continuous variables. If the formula is one-sided as in the example above (the space left of (~) is left empty), the formula is applied to all variables selected through cols. The specification provided in cols (default: all numeric variables not used in the formula) can be overridden by supplying one-or more dependent variables. For example HDW(data,var1 + var2 ~ fct1 + fct2) will return a data.frame with var1 and var2 centered on fct1 and fct2.

The special methods for plm::pseries and plm::pdata.frame center a panel-series or variables in a panel-data.frame on all panel-identifiers. By default in these methods fill = TRUE and variable.wise = TRUE, so missing values are kept. This change in the default arguments was done to ensure a coherent framework of functions and operators applied to plm panel-data classes.

Value

HDB returns fitted values of regressing x on fl. HDW returns residuals. See Details and Examples.

Note

**Caution with full (*) and factor-continuous variable interactions:** In general full interactions specified with (*) can be very slow on large data, and lfe::demeanlist is also not very speedy on interaction between factors and continuous variables, so these structures should be used with caution (don’t just specify an interaction like that on a large dataset, start with smaller data and see how long computations take. Upon further updates of lfe::demeanlist, performance might improve).

On the differences between fHDwithin/HDW... and fwithin/W...:

- fHDwithin/HDW can center data on multiple factors and also partial out continuous variables while fwithin/W only centers on one factor, but does that very efficiently...
- HDW(data,~ qF(group1) + qF(group2)) simultaneously centers numeric variables in data on group1 and group2, while W(data,~ group1 + group2) centers data on the interaction of group1 and group2. The equivalent operation in HDW would be: HDW(data,~ qF(group1):qF(group2)).
- W always does computations on the variable-wise complete observations (in both matrices and data.frames), whereas by default HDW removes all cases missing in either x or fl. In short, W(data,~ group1 + group2) is actually equivalent to HDW(data,~ qF(group1):qF(group2),variable.wise = TRUE). HDW(data,~ qF(group1):qF(group2)) would remove any missing cases.
• `fbetween` and `fwithin` have options to fill missing cases using group-averages and to add the overall mean back to group-demeaned data. These options are not available in `fHDbetween` and `fHDwithin`. Since `HDB` and `HDW` by default remove missing cases, they also don’t have options to keep grouping-columns as in `B` and `W`.

See Also

`fbetween` and `fwithin`, `fscale`, `TRA`, `fFtest`, `Data Transformations`, `Collapse Overview`

Examples

```r
HDW(mtcars$mpg, mtcars$carb)  # Simple regression problems..
HDW(mtcars$mpg, mtcars[-1])
HDW(mtcars$mpg, qM(mtcars[-1]))
HDW(qM(mtcars[3:4]), mtcars[1:2])
HDW(iris[1:2], iris[3:4])     # Partionalling columns 3 and 4 out of columns 1 and 2
HDW(iris[1:2], iris[3:5])     # Adding the Species factor -> fixed effect
HDW(wlddev, PCGDP + LIFEEX ~ iso3c + qF(year)) # Partialling out 2 fixed effects (iso3c is factor)
HDW(wlddev, PCGDP + LIFEEX ~ iso3c + qF(year), variable.wise = TRUE) # Variable-wise computations
HDW(wlddev, PCGDP + LIFEEX ~ iso3c + qF(year) + ODA) # Adding ODA as a continuous regressor
HDW(wlddev, PCGDP + LIFEEX ~ iso3c:qF(decade) + qF(year) + ODA) # Country-decade and year FE’s

# More complex examples (Currently only recommended for smaller data)
lm(HDW.mpg ~ HDW.hp, data = HDW(mtcars, ~ factor(cyl)*carb + vs + wt:gear + wt:gear:carb))
lm(mpg ~ hp + factor(cyl)*carb + vs + wt:gear + wt:gear:carb, data = mtcars)

lm(HDW.mpg ~ HDW.hp, data = HDW(mtcars, ~ factor(cyl)*carb + vs + wt:gear))
lm(mpg ~ hp + factor(cyl)*carb + vs + wt:gear, data = mtcars)

lm(HDW.mpg ~ HDW.hp, data = HDW(mtcars, ~ cyl*carb + vs + wt:gear))
lm(mpg ~ hp + cyl*carb + vs + wt:gear, data = mtcars)

lm(HDW.mpg ~ HDW.hp, data = HDW(mtcars, mpg + hp ~ cyl*carb + factor(cyl)*poly(drat,2)))
lm(mpg ~ hp + cyl*carb + factor(cyl)*poly(drat,2), data = mtcars)
```

Description

`flag` is an S3 generic to compute (sequences of) lags and leads. `L` and `F` are wrappers around `flag` representing the lag- and lead-operators, such that `L(x,-1) = F(x,1) = F(x)` and `L(x,-3:3) = F(x,3:-3)`. `L` & `F` provide more flexibility than `flag` when applied to data frames (i.e. column subsetting, formula input and id-variable-preservation capabilities...), but are otherwise identical.

(`flag` is more of a programmers function in style of the `Fast Statistical Functions` while `L` & `F` are more practical to use in regression formulas or for computations on data frames.)
Usage

flag(x, n = 1, ...)
L(x, n = 1, ...)
F(x, n = 1, ...)

## Default S3 method:
flag(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)
## Default S3 method:
L(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)
## Default S3 method:
F(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)

## S3 method for class 'matrix'
flag(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)
## S3 method for class 'matrix'
L(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)
## S3 method for class 'matrix'
F(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)

## S3 method for class 'data.frame'
flag(x, n = 1, g = NULL, t = NULL, fill = NA, stubs = TRUE, ...)
## S3 method for class 'data.frame'
L(x, n = 1, by = NULL, t = NULL, cols = is.numeric,
    fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'data.frame'
F(x, n = 1, by = NULL, t = NULL, cols = is.numeric,
    fill = NA, stubs = TRUE, keep.ids = TRUE, ...)

# Methods for compatibility with plm:

## S3 method for class 'pseries'
flag(x, n = 1, fill = NA, stubs = TRUE, ...)
## S3 method for class 'pseries'
L(x, n = 1, fill = NA, stubs = TRUE, ...)
## S3 method for class 'pseries'
F(x, n = 1, fill = NA, stubs = TRUE, ...)

## S3 method for class 'pdata.frame'
flag(x, n = 1, fill = NA, stubs = TRUE, ...)
## S3 method for class 'pdata.frame'
L(x, n = 1, cols = is.numeric, fill = NA, stubs = TRUE,
    keep.ids = TRUE, ...)
## S3 method for class 'pdata.frame'
F(x, n = 1, cols = is.numeric, fill = NA, stubs = TRUE,
    keep.ids = TRUE, ...)

# Methods for compatibility with dplyr:
flag-L-F

## S3 method for class 'grouped_df'
flag(x, n = 1, t = NULL, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'grouped_df'
L(x, n = 1, t = NULL, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)
## S3 method for class 'grouped_df'
F(x, n = 1, t = NULL, fill = NA, stubs = TRUE, keep.ids = TRUE, ...)

Arguments

x
   a vector, matrix, data.frame, panel-series (plm::pseries), panel-data.frame (plm::pdata.frame)
   or grouped tibble (dplyr::grouped_df). Note: Data must not be numeric.

n
   an integer vector indicating the lags/leads to compute.

g
   a factor, GRP object, atomic vector (internally converted to factor) or a list of
   vectors / factors (internally converted to a GRP object) used to group x.

by
   data.frame method: Same as g, but also allows one- or two-sided formulas i.e.
   ~ group1 or var1 + var2 ~ group1 + group2. See Examples.

t
   same input as g, to indicate the time-variable. For safe computation of lags/leads
   on unordered time-series and panels. Note: Data frame method also allows one-
   sided formula i.e. ~time, and grouped_df method also allows lazy-evaluation
   i.e. time (no quotes).

cols
   data.frame method: Select columns to lag/lead using a function, column names
   or indices. Default: All numeric variables. Note: cols is ignored if a two-sided
   formula is passed to by.

fill
   value to insert when vectors are shifted. Default is NA.

stubs
   logical. TRUE will rename all lagged / leaded columns by adding a stub or prefix
   "Ln." / "Fn."

keep.ids
   data.frame / pdata.frame / grouped_df methods: Logical. Drop all panel-identifiers
   from the output (which includes all variables passed to by or t). Note: For
   panel-data.frame’s and grouped tibbles identifiers are dropped, but the 'index' /
   'groups' attributes are kept.

... arguments to be passed to or from other methods.

Details

If a single integer is passed to n, and g/by and t are left empty, flag/L/F just returns x with all
columns lagged / leaded by n. If length(n)>1, and x is an atomic vector, flag/L/F returns a matrix
with lags / leads computed in the same order as passed to n. If instead x is a matrix / data.frame,
a matrix / data.frame with ncol(x)*length(n) columns is returned where columns are sorted first
by variable and then by lag (so all lags computed on a variable are grouped together). x can be of
any standard data type.

With groups/panel-identifiers supplied to g/by, flag/L/F efficiently computes a panel-lag by shift-
ing the entire vector(s) but inserting fill elements in the right places. If t is left empty, the data
needs to be ordered such that all values belonging to a group are consecutive and in the right or-
der. It is not necessary that the groups themselves occur in the right order. If a time-variable is
supplied to t (or a list of time-variables uniquely identifying the time-dimension), the panel is fully
identified and lags / leads can be securely computed even if the data is completely unordered (in
that case data is shifted around and fill values are inserted in such a way that if the data were sorted afterwards the result would be identical to computing lags / leads on sorted data). Internally this works by using the grouping- and time-variable(s) to create an ordering and then accessing the panel-vector(s) through this ordering. If the data is just a bit unordered, such computations are nearly as fast as computations on ordered data (without t), however, if the data is very unordered, it can take significantly longer. Since most panel-data come perfectly or pretty ordered, I recommend always supplying t to be on the safe-side.

It is also possible to compute lags / leads on unordered time-series (thus utilizing t but leaving g/by empty), although this is probably more rare to encounter than unordered panels. The methods applying to plm objects (panel-series and panel-data.frames) automatically utilize the panel-identifiers attached to these objects and thus securely compute fully identified panel-lags. If these objects have > 2 panel-identifiers attached to them, the last identifier is assumed to be the time-variable, and the others are taken as grouping-variables and interacted. I note that flag/L/F is significantly faster than plm::lag/plm::lead since the latter is written in R and based on a Split-Apply-Combine logic.

Value

x lagged / leaded n-times, grouped by g/by, ordered by t. See Details and Examples.

See Also

fdiff/D, fgrowth/G, Time-Series and Panel-Series, Collapse Overview

Examples

```r
## Simple Time-Series: Airpassengers
L(AirPassengers)  # 1 lag
F(AirPassengers)  # 1 lead

all_identical(L(AirPassengers), flag(AirPassengers),
             F(AirPassengers, -1))

L(AirPassengers, -1:3)  # 1 lead and 3 lags - output as matrix

tsp(EuStockMarkets)  # Data is recorded on 260 days per year
freq <- frequency(EuStockMarkets)
plot(stl(EuStockMarkets[, "DAX"], freq))  # There is some obvious seasonality
L(EuStockMarkets, -1:3*freq)  # 1 annual lead and 3 annual lags
summary(lm(DAX ~ ., data = L(EuStockMarkets, -1:3*freq)))  # DAX regressed on its own annual lead, # lags and the lead/lags of the other series

## World Development Panel Data
head(flag(wlddev, 1, wlddev$iso3c, wlddev$year))  # This lags all variables,
head(L(wlddev, 1, ~iso3c, ~year))  # This lags all numeric variables
head(L(wlddev, 1, ~iso3c))  # Without t: Works because data is ordered
head(L(wlddev, 1, PCGDP + LIFEEX ~ iso3c, ~year))  # This lags GDP per Capita & Life Expectancy
head(L(wlddev, 0:2, ~iso3c, ~year, cols = 9:10))  # Same, also retaining original series
head(L(wlddev, 1:2, PCGDP + LIFEEX ~ iso3c, ~year, ~year, ~year))  # Two lags, dropping id columns
```
### Description

`fmax` and `fmin` are generic functions that compute the (column-wise) maximum and minimum value of all values in `x`, (optionally) grouped by `g`. The `TRA` argument can further be used to transform `x` using its (grouped) maximum or minimum value.

### Usage

```r
default S3 method:
fmax(x, g = NULL, TRA = NULL, na.rm = TRUE, 
use.g.names = TRUE, ...)
default S3 method:
fmin(x, g = NULL, TRA = NULL, na.rm = TRUE, 
use.g.names = TRUE, ...)
```

## S3 method for class 'matrix'
fmax(x, g = NULL, TRA = NULL, na.rm = TRUE,
```

---

**fmax-fmin**  
Fast (Grouped) Maxima and Minima for Matrix-Like Objects

---

# Different ways of regressing GDP on its's lags and life-Expectancy and it's lags
summary(lm(PCGDP ~ ., L(wlddev, 0:2, iso3c, ~year, 9:10, keep.ids = FALSE)))  # 1 - Precomputing
summary(lm(PCGDP ~ L(PCGDP,1:2,iso3c,year) + L(LIFEEX,0:2,iso3c,year), wlddev))  # 2 - Ad-hoc
summary(lm(PCGDP ~ L(PCGDP,1:2,iso3c) + L(LIFEEX,0:2,iso3c), wlddev))  # 3 - same no year

```r
g = qF(wlddev$iso3c); t = qF(wlddev$year)
```
# 4- Precomputing
summary(lm(PCGDP ~ L(PCGDP,1:2,g,t) + L(LIFEEX,0:2,g,t), wlddev))  # panel-id's

## Using plm:
pwlddev <- plm::pdata.frame(wlddev, index = c("iso3c","year"))
head(L(pwlddev, 0:2, 9:10))  # Again 2 lags of GDP and LIFEEX
PCGDP <- pwlddev$PCGDP  # A panel-Series of GDP per Capita
L(PCGDP)  # Lagging the panel series
summary(lm(PCGDP ~ L(pwlddev, 0:2, 9:10, keep.ids = FALSE)))  # Running the lm again: WORKS!
# THIS DOES NOT WORK: Unfortunately lm drops the attributes of the columns,
# so L.default is used here and ordinary lags are computed. (with and attach don't retain attr.)
summary(lm(PCGDP ~ L(PCGDP,1:2) + L(LIFEEX,0:2), pwlddev))
LIFEEX <- pwlddev$LIFEEX  # To make it work, create pseries
summary(lm(PCGDP ~ L(PCGDP,1:2) + L(LIFEEX,0:2)))  # THIS WORKS !!

## Using dplyr:
library(dplyr)
```
```r
wlddev %>% group_by(iso3c) %>% select(PCGDP,LIFEEX) %>% L(0:2)
wlddev %>% group_by(iso3c) %>% select(year,PCGDP,LIFEEX) %>% L(0:2,year) # Also using t (safer)
```
use.g.names = TRUE, drop = TRUE, ...)
## S3 method for class 'data.frame'
fmax(x, g = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = TRUE, drop = TRUE, ...)
## S3 method for class 'data.frame'
fmin(x, g = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = TRUE, drop = TRUE, ...)
## S3 method for class 'grouped_df'
fmax(x, TRA = NULL, na.rm = TRUE,
    use.g.names = FALSE, keep.group_vars = TRUE, ...)
## S3 method for class 'grouped_df'
fmin(x, TRA = NULL, na.rm = TRUE,
    use.g.names = FALSE, keep.group_vars = TRUE, ...)

Arguments

x a numeric vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).
g a factor, GRP object, atomic vector (internally converted to factor) or a list of
    vectors / factors (internally converted to a GRP object) used to group x.
TRA an integer or quoted operator indicating the transformation to perform: 1 - "re-
    place_fill" | 2 - "replace" | 3 - "." | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "%" | 9
    - "%" | 10 - ".%". See TRA.
na.rm logical. Skip missing values in x. Defaults to TRUE and implemented at very
    little computational cost. If na.rm = FALSE a NA is returned when encountered.
use.g.names make group-names and add to the result as names (vector method) or row-names
    (matrix and data.frame method). No row-names are generated for data.tables
    and grouped tibbles.
drop matrix and data.frame method: drop dimensions and return an atomic vector if
    g = NULL and TRA = NULL.
keep.group_vars grouped_df method: Logical. FALSE removes grouping variables after computa-
    tion.
... arguments to be passed to or from other methods.

Details

Missing-value removal as controlled by the na.rm argument is done at no extra cost since in C++
any logical comparison involving NA or NaN evaluates to FALSE. Large performance gains can never-
theless be achieved in the presence of missing values if na.rm = FALSE, since then the correspond-
ing computation is terminated once a NA is encountered and NA is returned (unlike base::max and
base::min which just run through without any checks).
This all seamlessly generalizes to grouped computations, which are performed in a single pass (without splitting the data) and therefore extremely fast.

When applied to data frame’s with groups or drop = FALSE, fmax and fmin preserve all column attributes (such as variable labels) but do not distinguish between classed and unclassed objects. The attributes of the data.frame itself are also preserved.

Value

fmax returns the maximum value of x, grouped by g, or (if TRA is used) x transformed by its maximum value, grouped by g. Analogous, fmin returns the minimum value ...

See Also

Fast Statistical Functions, Collapse Overview

Examples

```r
## default vector method
mpg <- mtcars$mpg
fmax(mpg) # maximum value
fmin(mpg) # minimum value (all examples below use fmax but apply to fmin)
fmax(mpg, TRA = "%") # Simple transformation: Take percentage of maximum value
fmax(mpg, mtcars$cyl) # Grouped maximum value
fmax(mpg, mtcars[c(2,8:9)]) # More groups...
g <- GRP(mtcars, ~ cyl + vs + am) # Precomputing groups gives more speed !!
fmax(mpg, g)
fmax(mpg, g, TRA = "%") # Groupwise percentage of maximum value
fmax(mpg, g, TRA = "replace") # Groupwise replace by maximum value

## data.frame method
fmax(mtcars)
fmax(mtcars, TRA = "%")
fmax(mtcars, g)
fmax(mtcars, g, use.g.names = FALSE) # No row-names generated

## matrix method
m <- qM(mtcars)
fmax(m)
fmax(m, TRA = "%")
fmax(m, g) # etc...

## method for grouped tibbles - for use with dplyr
library(dplyr)
mtcars %>% group_by(cyl, vs, am) %>% fmax
mtcars %>% group_by(cyl, vs, am) %>% fmax("%")
mtcars %>% group_by(cyl, vs, am) %>% select(mpg) %>% fmax
```
**fmean**

*Fast (Grouped, Weighted) Mean for Matrix-Like Objects*

**Description**

`fmean` is a generic function that computes the (column-wise) mean of `x`, (optionally) grouped by `g` and/or weighted by `w`. The `TRA` argument can further be used to transform `x` using its (grouped, weighted) mean.

**Usage**

```r
fmean(x, ...)  
## Default S3 method:  
fmean(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,  
    use.g.names = TRUE, ...)  
## S3 method for class 'matrix'  
fmean(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,  
    use.g.names = TRUE, drop = TRUE, ...)  
## S3 method for class 'data.frame'  
fmean(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,  
    use.g.names = TRUE, drop = TRUE, ...)  
## S3 method for class 'grouped_df'  
fmean(x, w = NULL, TRA = NULL, na.rm = TRUE,  
    use.g.names = FALSE, keep.group_vars = TRUE, keep.w = TRUE, ...)```

**Arguments**

- `x`  
a numeric vector, matrix, data.frame or grouped tibble (by `::grouped_df`).
- `g`  
a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group `x`.
- `w`  
a numeric vector of (non-negative) weights, may contain missing values.
- `TRA`  
an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - ":" | 4 - ":+" | 5 - ":-" | 6 - ":%=" | 7 - ":+%" | 8 - ":%" | 9 - ":%+%" | 10 - ":-%%". See `TRA`.
- `na.rm`  
logical. Skip missing values in `x`. Defaults to TRUE and implemented at very little computational cost. If `na.rm = FALSE` a NA is returned when encountered.
- `use.g.names`  
make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
- `drop`  
matrix and data.frame method: drop dimensions and return an atomic vector if `g = NULL` and `TRA = NULL`. 


keep.group_vars

grouped_df method: Logical. FALSE removes grouping variables after computation.

keep.w

grouped_df method: Logical. Retain summed weighting variable after computation (if contained in grouped_df).

... arguments to be passed to or from other methods.

Details

Missing-value removal as controlled by the na.rm argument is done very efficiently by simply skipping them in the computation (thus setting na.rm = FALSE on data with no missing values doesn’t give extra speed). Large performance gains can nevertheless be achieved in the presence of missing values if na.rm = FALSE, since then the corresponding computation is terminated once a NA is encountered and NA is returned (unlike base::mean which just runs through without any checks).

The weighted mean is computed as \( \frac{\text{sum}(x \times w)}{\text{sum}(w)} \). If na.rm = TRUE, missing values will be removed from both x and w i.e. utilizing only \( x[\text{complete.cases}(x,w)] \) and \( w[\text{complete.cases}(x,w)] \).

This all seamlessly generalizes to grouped computations, which are performed in a single pass (without splitting the data) and therefore extremely fast.

When applied to data frame’s with groups or drop = FALSE, fmean preserves all column attributes (such as variable labels) but does not distinguish between classed and unclassed object (thus applying fmean to a factor column will give a 'malformed factor' error). The attributes of the data.frame itself are also preserved.

Value

The (w weighted) mean of x, grouped by g, or (if TRA is used) x transformed by its mean, grouped by g.

See Also

fmedian, fmode, Fast Statistical Functions, Collapse Overview

Examples

## default vector method
mpg <- mtcars$mpg
fmean(mpg)  # Simple mean
fmean(mpg, w = mtcars$hp)  # Weighted mean: Weighted by hp
fmean(mpg, TRA = "-")  # Simple transformation: demeaning (See also ?W)
fmean(mpg, mtcars$cyl)  # Grouped mean
fmean(mpg, mtcars[8:9])  # another grouped mean.
g <- GRP(mtcars[c(2,8:9)])
fmean(mpg, g)  # Pre-computing groups speeds up the computation
fmean(mpg, g, mtcars$hp)  # Grouped weighted mean
fmean(mpg, g, TRA = "-")  # Demeaning by group
fmean(mpg, g, mtcars$hp, ":-"")  # Group-demeaning using weighted group means

## data.frame method
fmean(mtcars)
fmedian

fmean(mtcars, g)
fmean(mtcars, g, TRA = "-") # etc...

## matrix method
m <- qM(mtcars)
fmean(m)
fmean(m, g)
fmean(m, g, TRA = "-") # etc...

## method for grouped tibbles - for use with dplyr
library(dplyr)

mtcars %>% group_by(cyl, vs, am) %>% fmean # Ordinary
mtcars %>% group_by(cyl, vs, am) %>% fmean(hp) # Weighted
mtcars %>% group_by(cyl, vs, am) %>% fmean(hp, "-") # Weighted Transform
mtcars %>% group_by(cyl, vs, am) %>%
  select(mpg, hp) %>% fmean(hp, "-") # Only mpg

---

fmedian  

*Fast (Grouped) Median Value for Matrix-Like Objects*

**Description**

fmedian is a generic function that computes the (column-wise) median value of all values in x, (optionally) grouped by g. The TRA argument can further be used to transform x using its (grouped) median value.

**Usage**

fmedian(x, ...)

## Default S3 method:
fmedian(x, g = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, ...)

## S3 method for class 'matrix'
fmedian(x, g = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
fmedian(x, g = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'grouped_df'
fmedian(x, TRA = NULL, na.rm = TRUE,
  use.g.names = FALSE, keep.group_vars = TRUE, ...)

Arguments

- **x**
a numeric vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).

- **g**
a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group `x`.

- **TRA**
an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "-" | 4 - "-+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "*" | 9 - "%%" | 10 - "-%-%". See `TRA`.

- **na.rm**
logical. Skip missing values in `x`. Defaults to TRUE and implemented at very little computational cost. If `na.rm = FALSE` a NA is returned when encountered.

- **use.g.names**
make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.

- **drop**
matrix and data.frame method: drop dimensions and return an atomic vector if `g = NULL` and `TRA = NULL`.

- **keep.group_vars**
grouped_df method: Logical. FALSE removes grouping variables after computation.

- **...**
arguments to be passed to or from other methods.

Details

Median value estimation is done using `std::nth_element` in C++, which is an efficient partial sorting algorithm. A downside of this is that vectors need to be copied first and then partially sorted, thus `fmedian` currently requires additional memory equal to the size of the object (`x`).

Grouped computations are currently performed by mapping the data to a sparse-array directed by `g` and then partially sorting each row (group) of that array. For reasons I don’t fully understand this requires less memory than a full deep copy which is done with no groups.

When applied to data frame’s with groups or `drop = FALSE`, `fmedian` preserves all column attributes (such as variable labels) but does not distinguish between classed and unclassed objects. The attributes of the data.frame itself are also preserved.

Value

The median value of `x`, grouped by `g`, or (if `TRA` is used) `x` transformed by its median value, grouped by `g`.

See Also

`fmean`, `fmode`, Fast Statistical Functions, Collapse Overview

Examples

```r
## default vector method
mpg <- mtcars$mpg
fmedian(mpg) # Simple median value
fmedian(mpg, TRA = "-") # Simple transformation: Subtract median value
```
fmode

fmedian(mpg, mtcars$ cyl) # Grouped median value
fmedian(mpg, mtcars[c(2,8:9)]) # More groups...
g <- GRP(mtcars, ~ cyl + vs + am) # Precomputing groups gives more speed !!
fmedian(mpg, g)
fmedian(mpg, g, TRA = "-") # Groupwise subtract median value

## data.frame method
fmedian(mtcars)
fmedian(mtcars, TRA = "-"
)fmedian(mtcars, g)
fmedian(mtcars, g, use.g.names = FALSE) # No row-names generated

## matrix method
m <- qM(mtcars)
fmedian(m)
fmedian(m, TRA = "-"
)fmedian(m, g) # etc...

## method for grouped tibbles - for use with dplyr
library(dplyr)
mtcars %>% group_by(cyl, vs, am) %>% fmedian
mtcars %>% group_by(cyl, vs, am) %>% fmedian("-")
mtcars %>% group_by(cyl, vs, am) %>% select(mpg) %>% fmedian

fmode

Fast (Grouped, Weighted) Statistical Mode for Matrix-Like Objects

Description

fmode is a generic function and returns the (column-wise) statistical mode i.e. the most frequent value of x, (optionally) grouped by g and/or weighted by w. The TRA argument can further be used to transform x using its (grouped, weighted) mode.

Usage

fmode(x, ...)

## Default S3 method:
fmode(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = TRUE, ...)

## S3 method for class 'matrix'
fmode(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
fmode(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = TRUE, drop = TRUE, ...)
## S3 method for class 'grouped_df'

fmode(x, w = NULL, TRA = NULL, na.rm = TRUE,
         use.g.names = FALSE, keep.group_vars = TRUE, keep.w = TRUE, ...)

Arguments

- **x**: a vector, matrix, data.frame or grouped tibble (`dplyr::grouped_df`).
- **g**: a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.
- **w**: a numeric vector of (non-negative) weights, may contain missing values.
- **TRA**: an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "." | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "+" | 9 - "%%" | 10 - "-%". See TRA.
- **na.rm**: logical. Skip missing values in x. Defaults to TRUE and implemented at very little computational cost. If na.rm = FALSE, NA is treated as any other value.
- **use.g.names**: make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
- **drop**: *matrix and data.frame method*: drop dimensions and return an atomic vector if g = NULL and TRA = NULL.
- **keep.group_vars**: *grouped_df method*: Logical. FALSE removes grouping variables after computation.
- **keep.w**: *grouped_df method*: Logical. Retain max of weighting variable after computation (if contained in grouped_df).
- **...**: arguments to be passed to or from other methods.

Details

fmode implements a pretty fast algorithm to find the statistical mode utilizing index- hashing implemented in the Rcpp::sugar::IndexHash class.

If all values are distinct, the first value is returned. If there are multiple distinct values having the top frequency, the first value established as having the top frequency when passing through the data from element 1 to element n is returned. If na.rm = FALSE, NA is not removed but treated as any other value (i.e. it’s frequency is counted). If all values are NA, NA is always returned.

The weighted mode is computed by summing up the weights for all distinct values and choosing the value with the largest sum. If na.rm = TRUE, missing values will be removed from both x and w i.e. utilizing only x[complete.cases(x,w)] and w[complete.cases(x,w)].

This all seamlessly generalizes to grouped computations, which are currently performed by mapping the data to a sparse-array directed by g and then going group-by-group.

fmode preserves all the attributes of the objects it is applied to (apart from names or row-names which are adjusted as necessary). If a data frame is passed to fmode and drop = TRUE, base::unlist will be called on the result, which might or might not be sensible depending on the data at hand.
Value

The statistical mode of \( x \), grouped by \( g \), or (if \( \text{TRA} \) is used) \( x \) transformed by its mode, grouped by \( g \). See also Details.

See Also

\( \text{fmean, fmedian, Fast Statistical Functions, Collapse Overview} \)

Examples

```r
## World Development Data
attach(wlddev)
## default vector method
fmode(PCGDP) # Numeric mode
fmode(PCGDP, iso3c) # Grouped numeric mode
fmode(PCGDP, iso3c, LIFEEX) # Grouped and weighted numeric mode
fmode(region) # Factor mode
fmode(date) # Date mode (defaults to first value since panel is balanced)
fmode(country) # Character mode (also defaults to first value)
fmode(OECD) # Logical mode
# ...all the above can also be performed grouped and weighted

## matrix method
m <- qM(airquality)
fmode(m)
fmode(m, na.rm = FALSE) # NA frequency is also counted
fmode(m, airquality$Month) # Groupwise
fmode(m, w = airquality$Day) # Weighted: Later days in the month are given more weight
fmode(m>50, airquality$Month) # Groupwise logical mode
# etc ...

## data.frame method
fmode(wlddev) # Gives one row
fmode(wlddev, drop = TRUE) # calling unlist -> coerce to character vector
fmode(wlddev, iso3c) # Grouped mode
fmode(wlddev, iso3c, LIFEEX) # Grouped and weighted mode

detach(wlddev)
```

---

\( \text{fNdistinct} \)

**Fast (Grouped) Distinct Value Count for Matrix-Like Objects**

Description

\( \text{fNdistinct} \) is a generic function that (column-wise) computes the number of distinct values in \( x \), (optionally) grouped by \( g \). It is significantly faster than \( \text{length(unique}(x)) \). The \( \text{TRA} \) argument can further be used to transform \( x \) using its (grouped) distinct value count.
Usage

fNdistinct(x, ...)

## Default S3 method:
fNdistinct(x, g = NULL, TRA = NULL, na.rm = TRUE,
        use.g.names = TRUE, ...)

## S3 method for class 'matrix'
fNdistinct(x, g = NULL, TRA = NULL, na.rm = TRUE,
        use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
fNdistinct(x, g = NULL, TRA = NULL, na.rm = TRUE,
        use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'grouped_df'
fNdistinct(x, TRA = NULL, na.rm = TRUE,
        use.g.names = FALSE, keep.group_vars = TRUE, ...)

Arguments

x a vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).

g a factor, GRP object, atomic vector (internally converted to factor) or a list of
  vectors / factors (internally converted to a GRP object) used to group x.

TRA an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "." | 4 - "-" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "*" | 9 - "-+%" | 10 - "-%%". See TRA.

na.rm logical. TRUE: Skip missing values in x (faster computation). FALSE: Also consider 'NA' as one distinct value.

use.g.names make group-names and add to the result as names (vector method) or row-names
  (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.

drop matrix and data.frame method: drop dimensions and return an atomic vector if
  g = NULL and TRA = NULL.

keep.group_vars grouped_df method: Logical. FALSE removes grouping variables after computa-
  tion.

... arguments to be passed to or from other methods.

Details

fNdistinct implements a fast algorithm to find the number of distinct values utilizing index- hashing implemented in the Rcpp::sugar::IndexHash class.

If na.rm = TRUE (the default), missing values will be skipped yielding substantial performance gains in data with many missing values. If na.rm = TRUE, missing values will simply be treated as any
other value and read into the hash-map. Thus with the former, a numeric vector \( c(1.25, \text{NaN}, 3.56, \text{NA}) \) will have a distinct value count of 2, whereas the latter will return a distinct value count of 4.

Grouped computations are currently performed by mapping the data to a sparse-array directed by \( g \) and then hash-mapping each group. This is often not much slower than using a larger hash-map for the entire data when \( g = \text{NULL} \).

fNdistinct preserves all attributes of non-classed vectors / columns, and only the 'label' attribute (if available) of classed vectors / columns (i.e. dates or factors). When applied to data frames and matrices, the row-names are adjusted as necessary.

### Value

Integer. The number of distinct values in \( x \), grouped by \( g \), or (if TRA is used) \( x \) transformed by its distinct value count, grouped by \( g \).

### See Also

fNobs, Fast Statistical Functions, Collapse Overview

### Examples

```r
## default vector method
fNdistinct(airquality$Solar.R)  # Simple distinct value count
fNdistinct(airquality$Solar.R, airquality$Month)  # Grouped distinct value count

## data.frame method
fNdistinct(airquality)
fNdistinct(airquality$Solar.R, airquality$Month)
fNdistinct(wlddev)  # Works with data of all types!
head(fNdistinct(wlddev, wlddev$iso3c))

## matrix method
aqm <- qM(airquality)
fNdistinct(aqm)  # Also works for character or logical matrices
fNdistinct(aqm, airquality$Month)

## method for grouped tibbles - for use with dplyr:
library(dplyr)
airquality %>% group_by(Month) %>% fNdistinct
wlddev %>% group_by(country) %>%
  select(PCGDP,LIFEEX,GINI,ODA) %>% fNdistinct
```

---

fNobs is a generic function that (column-wise) computes the number of non-missing values in \( x \), (optionally) grouped by \( g \). It is much faster than \( \text{sum}(!\text{is.na}(x)) \). The \( \text{TRA} \) argument can further be used to transform \( x \) using its (grouped) observation count.
Usage

fNobs(x, ...)

## Default S3 method:
fNobs(x, g = NULL, TRA = NULL, use.g.names = TRUE, ...)

## S3 method for class 'matrix'
fNobs(x, g = NULL, TRA = NULL, use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
fNobs(x, g = NULL, TRA = NULL, use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'grouped_df'
fNobs(x, TRA = NULL, use.g.names = FALSE, keep.group_vars = TRUE, ...)

Arguments

x      a vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).
g      a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.
TRA      an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - ":-" | 4 - ":+" | 5 - ":/" | 6 - ":%" | 7 - ":+" | 8 - "*:1" | 9 - "%:*" | 10 - "%:*". See TRA.
use.g.names      make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
don         matrix and data.frame method: drop dimensions and return an atomic vector if g = NULL and TRA = NULL.
keep.group_vars   grouped_df method: Logical. FALSE removes grouping variables after computation.

Details

fNobs preserves all attributes of non-classed vectors / columns, and only the 'label' attribute (if available) of classed vectors / columns (i.e. dates or factors). When applied to data frames and matrices, the row-names are adjusted as necessary.

Value

Integer. The number of non-missing observations in x, grouped by g, or (if TRA is used) x transformed by its number of non-missing observations, grouped by g.

See Also

fNdistinct, Fast Statistical Functions, Collapse Overview
Examples

## default vector method
fNobs(airquality$Solar.R) # Simple Nobs
fNobs(airquality$Solar.R, airquality$Month) # Grouped Nobs

## data.frame method
fNobs(airquality)
fNobs(airquality, airquality$Month)
fNobs(wlddev) # Works with data of all types!
head(fNobs(wlddev, wlddev$iso3c))

## matrix method
aqm <- qM(airquality)
fNobs(aqm)
# Also works for character or logical matrices
fNobs(aqm, airquality$Month)

## method for grouped tibbles - for use with dplyr
library(dplyr)
airquality %>% group_by(Month) %>% fNobs
wlddev %>% group_by(country) %>%
  select(PCGDP,LIFEEX,GINI,ODA) %>% fNobs

---

fprod

Fast (Grouped, Weighted) Product for Matrix-Like Objects

Description

fprod is a generic function that computes the (column-wise) product of all values in x, (optionally) grouped by g and/or weighted by w. The TRA argument can further be used to transform x using its (grouped) product.

Usage

fprod(x, ...)

## Default S3 method:
fprod(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, ...)

## S3 method for class 'matrix'
fprod(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, drop = TRUE, ...)

## S3 method for class 'data.frame'
fprod(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
  use.g.names = TRUE, drop = TRUE, ...)
`fprod` function and its arguments:

### Arguments

- **x**: a numeric vector, matrix, data.frame or grouped tibble (`dplyr::grouped_df`).
- **g**: a factor, `GRP` object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a `GRP` object) used to group `x`.
- **w**: a numeric vector of (non-negative) weights, may contain missing values.
- **TRA**: an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "+" | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "+" | 9 - "+" | 10 - "+". See `TRA`.
- **na.rm**: logical. Skip missing values in `x`. Defaults to `TRUE` and implemented at very little computational cost. If `na.rm = FALSE` a NA is returned when encountered.
- **use.g.names**: make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
- **drop**: matrix and data.frame method: drop dimensions and return an atomic vector if `g = NULL` and `TRA = NULL`.
- **keep.group_vars**: grouped_df method: Logical. FALSE removes grouping variables after computation.
- **keep.w**: grouped_df method: Logical. Retain product of weighting variable after computation (if contained in grouped_df).
- **...**: arguments to be passed to or from other methods.

### Details

Non-grouped product computations internally utilize long-doubles in C++, for additional numeric precision.

Missing-value removal as controlled by the `na.rm` argument is done very efficiently by simply skipping them in the computation (thus setting `na.rm = FALSE` on data with no missing values doesn’t give extra speed). Large performance gains can nevertheless be achieved in the presence of missing values if `na.rm = FALSE`, since then the corresponding computation is terminated once a NA is encountered and NA is returned (unlike `base::prod` which just runs through without any checks).

This all seamlessly generalizes to grouped computations, which are performed in a single pass (without splitting the data) and therefore extremely fast.

The weighted product is computed as `prod(x * w)`. If `na.rm = TRUE`, missing values will be removed from both `x` and `w` i.e. utilizing only `x[complete.cases(x, w)]` and `w[complete.cases(x, w)]`.

When applied to data frame’s with groups or `drop = FALSE`, `fprod` preserves all column attributes (such as variable labels) but does not distinguish between classed and unclassed objects. The attributes of the data.frame itself are also preserved.
Value
The product of x, grouped by g, or (if TRA is used) x transformed by its product, grouped by g.

See Also
fsum, Fast Statistical Functions, Collapse Overview

Examples

```r
## default vector method
mpg <- mtcars$mpg
fprod(mpg)  # Simple product
fprod(mpg, w = mtcars$hp)  # Weighted product
fprod(mpg, TRA = "/")  # Simple transformation: Divide by product
fprod(mpg, mtcars$cyl)  # Grouped product
fprod(mpg, mtcars$cyl, mtcars$hp)  # Weighted grouped product
fprod(mpg, mtcars[c(2,8:9)])  # More groups...
g <- GRP(mtcars, ~ cyl + vs + am)  # Precomputing groups gives more speed !!
fprod(mpg, g)
fprod(mpg, g, TRA = "/")  # Groupwise divide by product

## data.frame method
fprod(mtcars)
fprod(mtcars, TRA = "/")
fprod(mtcars, g)
fprod(mtcars, g, use.g.names = FALSE)  # No row-names generated

## matrix method
m <- qM(mtcars)
fprod(m)
fprod(m, TRA = "/")
fprod(m, g)  # etc...

## method for grouped tibbles - for use with dplyr
library(dplyr)
mtcars %>% group_by(cyl,vs,am) %>% fprod(hp)  # Weighted grouped product
mtcars %>% group_by(cyl,vs,am) %>% fprod(TRA = "/")
mtcars %>% group_by(cyl,vs,am) %>% select(mpg) %>% fprod
```

Description
fscale is a generic function to efficiently standardize (scale and center) data. STD is a wrapper around fscale representing the ‘standardization operator’, with more options than fscale when applied to matrices and data frames. Standardization can be simple or groupwise, ordinary or weighted.

Note: For centering without scaling see fwithin/W.
Usage

fscale(x, ...)  
STD(x, ...)

### Default S3 method:
fscale(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)
STD(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)

### S3 method for class 'matrix'
fscale(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)
STD(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, sd = 1,
    stub = "STD.", ...)

### S3 method for class 'data.frame'
fscale(x, g = NULL, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)
STD(x, by = NULL, w = NULL, cols = is.numeric, na.rm = TRUE,
    mean = 0, sd = 1, stub = "STD.", keep.by = TRUE, keep.w = TRUE, ...)

# Methods for compatibility with plm:

### S3 method for class 'pseries'
fscale(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)
STD(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)

### S3 method for class 'pdata.frame'
fscale(x, effect = 1L, w = NULL, na.rm = TRUE, mean = 0, sd = 1, ...)
STD(x, effect = 1L, w = NULL, cols = is.numeric, na.rm = TRUE,
    mean = 0, sd = 1, stub = "STD.", keep.ids = TRUE, keep.w = TRUE, ...)

# Methods for compatibility with dplyr:

### S3 method for class 'grouped_df'
fscale(x, w = NULL, na.rm = TRUE, mean = 0, sd = 1,
    keep.group_vars = TRUE, keep.w = TRUE, ...)
STD(x, w = NULL, na.rm = TRUE, mean = 0, sd = 1,
    stub = "STD.", keep.group_vars = TRUE, keep.w = TRUE, ...)

Arguments

x             a numeric vector, matrix, data.frame, panel-series (plm::pseries), panel-data.frame (plm::pdata.frame) or grouped tibble (dplyr::grouped_df).
g            a factor, GRP object, atomic vector (internally converted to factor) or a list of
vectors / factors (internally converted to a GRP object) used to group x.

by

STD data.frame method: Same as g, but also allows one- or two-sided formulas i.e. ~ group1 or var1 + var2 ~ group1 + group2. See Examples.

cols
data.frame method: Select columns to scale using a function, column names or indices. Default: All numeric variables. Note: cols is ignored if a two-sided formula is passed to by.

w

a numeric vector of (non-negative) weights. STD data.frame and pdata.frame methods also allow a one-sided formula i.e. ~ weightcol. The grouped_df (dplyr) method supports lazy-evaluation. See Examples.

na.rm

logical. skip missing values in x or w when computing means and sd’s.

effect

plm methods: Select which panel identifier should be used as grouping variable. 1L means first variable in the plm::index, 2L the second etc. if more than one integer is supplied, the corresponding index-variables are interacted.

stub

a prefix or stub to rename all transformed columns. FALSE will not rename columns.

mean

the mean to center on (default is 0). If mean = FALSE, no centering will be performed. In that case the scaling is mean-preserving. A numeric value different from 0 (i.e. mean = 5) will be added to the data after subtracting out the mean(s), such that the data will have a mean of 5. A special option when performing grouped scaling and centering is mean = "overall.mean". In that case the overall mean of the data will be added after subtracting out group means.

sd

the standard deviation to scale the data to (default is 1). A numeric value different from 0 (i.e. sd = 3) will scale the data to have a standard deviation of 3. A special option when performing grouped scaling is sd = "within.sd". In that case the within standard deviation (= the standard deviation of the group-centered series) will be calculated and applied to each group. The results is that the variance of the data within each group is harmonized without forcing a certain variance (such as 1).

keep.by, keep.ids, keep.group_vars
data.frame, pdata.frame and grouped_df methods: Logical. Retain grouping / panel-identifier columns in the output. For STD.data.frame this only works if grouping variables were passed in a formula.

keep.w
data.frame, pdata.frame and grouped_df methods: Logical. Retain column containing the weights in the output. Only works if w is passed as formula / lazy-expression.

...arguments to be passed to or from other methods.

Details

If g = NULL, fscale by default (column-wise) subtracts the mean or weighted mean (if w is supplied) from all data points in x, and then divides this difference by the standard deviation or frequency-weighted standard deviation (if w is supplied). The result is that all columns in x will have mean 0 and standard deviation 1. Alternatively, data can be scaled to have a mean of mean and a standard deviation of sd. If mean = FALSE the data is only scaled (not centered) such that the mean of the data is preserved.
Means and standard deviations are computed using Welford’s numerically stable online algorithm. With groups supplied to \( g \), this standardizing becomes groupwise, so that in each group (in each column) the data points will have mean \( \text{mean} \) and standard deviation \( \text{sd} \). Naturally if \( \text{mean} = \text{FALSE} \) then each group is just scaled and the mean is preserved. For centering without scaling see \texttt{fwithin}.

If \( \text{na.rm} = \text{FALSE} \) and a \( \text{NA} \) or \( \text{NaN} \) is encountered, the mean and \( \text{sd} \) for that group will be \( \text{NA} \), and all data points belonging to that group will also be \( \text{NA} \) in the output.

If \( \text{na.rm} = \text{TRUE} \), means and \( \text{sd} \)’s are computed (column-wise) on the available data points, and also the weight vector can have missing values. In that case (\( w \) also has missing values), the weighted mean \( \text{an sd are computed on (column-wise)} \, \text{complete.cases} (x,w) \), and \( x \) is scaled using these statistics. \textit{Note} that \texttt{fscale} will not insert a missing value in \( x \) if the weight for that value is missing, rather, that value will be scaled using a weighted mean and standard-deviated computed without itself! (The intention here is that a few (randomly) missing weights shouldn’t break the computation when \( \text{na.rm} = \text{TRUE} \), but it is not meant for weight vectors with many missing values. If you don’t like this behavior, you should prepare your data using \( x[\text{is.na}(w),] \leftarrow \text{NA} \), or impute your weight vector for non-missing \( x \)).

Special options for grouped scaling are \texttt{mean = "overall.mean"} and \texttt{sd = "within.sd"}. The former group-centers vectors on the overall mean of the data (see \texttt{fwithin} for more details) and the latter scales the data in each group to have the within-group standard deviation (= the standard deviation of the group-centered data). Thus scaling a grouped vector with options \texttt{mean = "overall.mean"} and \texttt{sd = "within.sd"} amounts to removing all differences in the mean and standard deviations between these groups. In weighted computations, \texttt{mean = "overall.mean"} will subtract weighted group-means from the data and add the overall weighted mean of the data, whereas \texttt{sd = "within.sd"} will compute the weighted within-standard deviation and apply it to each group.

**Value**

\( x \) standardized (mean = \( \text{mean} \), standard deviation = \( \text{sd} \)), grouped by \( g / \text{by} \), weighted with \( w \). See Details.

**See Also**

\texttt{fwithin/}, \texttt{W}, \texttt{Fast Statistical Functions}, \texttt{TRA}, \texttt{Data Transformations}, \texttt{Collapse Overview}

**Examples**

```r
## Simple Scaling & Centering / Standardizing
fscale(mtcars) # Doesn't rename columns
STD(mtcars) # By default adds a prefix
qsu(STD(mtcars)) # See that is works
qsu(STD(mtcars, mean = 5, sd = 3)) # Assigning a mean of 5 and a standard deviation of 3
qsu(STD(mtcars, mean = FALSE)) # No centering: Scaling is mean-preserving

## Panel-Data
head(fscale(get_vars(wlddev,9:12), wlddev$iso3c)) # Standardizing 4 series within each country
head(STD(wlddev, ~iso3c, cols = 9:12)) # Same thing using STD, id's added
pwcor(fscale(get_vars(wlddev,9:12), wlddev$iso3c)) # Correlating panel-series after standardizing
fmean(get_vars(wlddev, 9:12)) # This calculates the overall means
```
fsum(\text{within}(\text{get-vars}(\text{wlddev}, 9:12), \text{wlddev}\$\text{iso3c})) \quad \# \text{This calculates the within standard deviations}
qsu(\text{fscale}(\text{get-vars}(\text{wlddev}, 9:12), \text{wlddev}\$\text{iso3c}), \quad \# \text{This group-centers on the overall mean and}
\quad \text{mean = "overall.mean", sd = "within.sd"}, \quad \# \text{group-scales to the within standard deviation}
\quad \text{by = wlddev}\$\text{iso3c}) \quad \# \rightarrow \text{data harmonized in the first 2 moments}

## Using plm
\text{pwlddev} \leftarrow \text{plm}\text{::pdata.frame(}\text{wlddev, index = c("iso3c","year")})
\text{head(STD(pwlddev))} \quad \# \text{Standardizing all numeric variables by country}
\text{head(STD(pwlddev, effect = 2L))} \quad \# \text{Standardizing all numeric variables by year}

## Weighted Standardizing
weights = \text{abs(rnorm(nrow(}\text{wlddev})))
\text{head(fscale(\text{get-vars(}\text{wlddev}, 9:12), \text{wlddev}\$\text{iso3c, weights}))}
\text{head(STD(}\text{wlddev, -iso3c, weights, 9:12}))

# Using dplyr
library(dplyr)
\text{wlddev} \%\% \text{group_by(}\text{iso3c}) \%\% \text{select(}\text{PCGDP, LIFEEX}) \%\% \text{STD}
\text{wlddev} \%\% \text{group_by(}\text{iso3c}) \%\% \text{select(}\text{PCGDP, LIFEEX}) \%\% \text{STD(}\text{weights}) \quad \# \text{weighted standardizing}
\text{wlddev} \%\% \text{group_by(}\text{iso3c}) \%\% \text{select(}\text{PCGDP, LIFEEX, ODA}) \%\% \text{STD(ODA}) \quad \# \text{weighting by ODA}
\# \ldots \text{keeps the weight column unless keep.w = FALSE}

---

\textbf{fsum}

\textit{Fast (Grouped, Weighted) Sum for Matrix-Like Objects}

\textbf{Description}

\texttt{fsum} is a generic function that computes the (column-wise) sum of all values in \texttt{x}, (optionally) grouped by \texttt{g} and/or weighted by \texttt{w} (i.e. to calculate survey totals). The \texttt{TRA} argument can further be used to transform \texttt{x} using its (grouped, weighted) sum.

\textbf{Usage}

\texttt{fsum(x, \ldots)}

\texttt{## Default S3 method:}
\texttt{fsum(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE, use.g.names = TRUE, \ldots)}

\texttt{## S3 method for class 'matrix'}
\texttt{fsum(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE, use.g.names = TRUE, drop = TRUE, \ldots)}

\texttt{## S3 method for class 'data.frame'}
\texttt{fsum(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE, use.g.names = TRUE, drop = TRUE, \ldots)}

\texttt{## S3 method for class 'grouped_df'}
fsum(x, w = NULL, TRA = NULL, na.rm = TRUE,
    use.g.names = FALSE, keep.group_vars = TRUE, keep.w = TRUE, ...)

Arguments

x  a numeric vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).
g  a factor, GRP object, atomic vector (internally converted to factor) or a list of
    vectors / factors (internally converted to a GRP object) used to group x.
w  a numeric vector of (non-negative) weights, may contain missing values.
TRA an integer or quoted operator indicating the transformation to perform: 1 - "re-
    place_fill" | 2 - "replace" | 3 - "." | 4 - "+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "+%" | 9
    - "%" | 10 - ".%". See TRA.
na.rm logical. Skip missing values in x. Defaults to TRUE and implemented at very
    little computational cost. If na.rm = FALSE a NA is returned when encountered.
use.g.names make group-names and add to the result as names (vector method) or row-names
    (matrix and data.frame method). No row-names are generated for data.tables
    and grouped tibbles.
drop  matrix and data.frame method: drop dimensions and return an atomic vector if
    g = NULL and TRA = NULL.
keep.group_vars grouped_df method: Logical. FALSE removes grouping variables after computa-
    tion.
keep.w grouped_df method: Logical. Retain summed weighting variable after computa-
    tion (if contained in grouped_df).
... arguments to be passed to or from other methods.

Details

Missing-value removal as controlled by the na.rm argument is done very efficiently by simply skipping
them in the computation (thus setting na.rm = FALSE on data with no missing values doesn’t
give extra speed). Large performance gains can nevertheless be achieved in the presence of miss-
ing values if na.rm = FALSE, since then the corresponding computation is terminated once a NA is
encountered and NA is returned (unlike base::sum which just runs through without any checks).

The weighted sum (i.e. survey total) is computed as sum(x * w). If na.rm = TRUE, missing values
will be removed from both x and w. i.e. utilizing only x[complete.cases(x,w)] and w[complete.cases(x,w)].

This all seamlessly generalizes to grouped computations, which are performed in a single pass
(without splitting the data) and therefore extremely fast. See Benchmark and Examples below.

When applied to data frame’s with groups or drop = FALSE, fsum preserves all column attributes
(such as variable labels) but does not distinguish between classed and unclassed objects. The at-
tributes of the data.frame itself are also preserved.

Value

The (w weighted) sum of x, grouped by g, or (if TRA is used) x transformed by its sum, grouped by

g.
See Also

fprod, Fast Statistical Functions, Collapse Overview

Examples

### default vector method

mpg <- mtcars$mpg
fsum(mpg)  # Simple sum
fsum(mpg, w = mtcars$hp)  # Weighted sum (total): Weighted by hp
fsum(mpg, TRA = "%")  # Simple transformation: obtain percentages of mpg
fsum(mpg, mtcars$cyl)  # Grouped sum
fsum(mpg, mtcars$cyl, mtcars$hp)  # Weighted grouped sum (total)
fsum(mpg, mtcars$[c(2,8:9)])  # More groups...
g <- GRP(mtcars, ~ cyl + vs + am) # Precomputing groups gives more speed!!
fsum(mpg, g)
fmean(mpg, g) == fsum(mpg, g) / fNobs(mpg, g)
fsum(mpg, g, TRA = "%")  # Percentages by group

### data.frame method

fsum(mtcars)
fsum(mtcars, TRA = "%")
fsum(mtcars, g)
fsum(mtcars, g, TRA = "%")

### matrix method

m <- qM(mtcars)
fsum(m)
fsum(m, TRA = "%")
fsum(m, g)
fsum(m, g, TRA = "%")

### method for grouped tibbles - for use with dplyr

library(dplyr)
mtcars %>% group_by(cyl,vs,am) %>% fsum(hp)  # Weighted grouped sum (total)
mtcars %>% group_by(cyl,vs,am) %>% fsum(TRA = "%")
mtcars %>% group_by(cyl,vs,am) %>% select(mpg) %>% fsum

Benchmark

### Let's run some benchmarks and compare fsum against data.table and base::rowsum

# Starting with small data
mtcDT <- qDT(mtcars)
f <- qF(mtcars$cyl)

library(microbenchmark)
microbenchmark(mtcDT[, lapply(.SD, sum), by = f],
               rowsum(mtcDT, f, reorder = FALSE),
               fsum(mtcDT, f, na.rm = FALSE), unit = "relative")
# My results:
expr    min     lq    mean  median   uq   max  neval  cld
mtcDT[, lapply(.SD, sum), by = f] 145.436928 123.542134 88.681111 98.336378 71.880479 85.217726 100  c
rowsum(mtcDT, f, reorder = FALSE) 2.833333 2.798203 2.489064 2.937889 2.425724 2.181173 100  b
fsum(mtcDT, f, na.rm = FALSE) 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 100  a

# Now larger data
tdata <- qDT(replicate(100, rnorm(1e5), simplify = FALSE)) # 100 columns with 100.000 obs
f <- qF(sample.int(1e4, 1e5, TRUE)) # A factor with 10.000 groups

microbenchmark(tdata[, lapply(.SD, sum), by = f],
rowsum(tdata, f, reorder = FALSE),
  fsum(tdata, f, na.rm = FALSE), unit = "relative")

# My results:
expr min lq mean median uq max neval cld
tdata[, lapply(.SD, sum), by = f] 2.646992 2.975489 2.834771 3.081313 3.120070 1.276647 100  c
rowsum(tdata, f, reorder = FALSE) 1.747567 1.753313 1.629036 1.758043 1.839348 0.272093 100  b
fsum(tdata, f, na.rm = FALSE) 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 100  a

tdata

fvar-fsd

Fast (Grouped, Weighted) Variance and Standard Deviation for Matrix-Like Objects

Description

fvar and fsd are generic functions that compute the (column-wise) variance and standard deviation of x, optionally grouped by g and/or frequency-weighted by w. The TRA argument can further be used to transform x using its (grouped, weighted) variance/sd.

Usage

fvar(x, ...)
fsd(x, ...)## Default S3 method:
fvar(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
     use.g.names = TRUE, stable.algo = TRUE, ...)
## Default S3 method:
fsd(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
     use.g.names = TRUE, drop = TRUE, stable.algo = TRUE, ...)
## S3 method for class 'matrix'
fvar(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
     use.g.names = TRUE, drop = TRUE, stable.algo = TRUE, ...)
## S3 method for class 'matrix'
fsd(x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE,
     use.g.names = TRUE, drop = TRUE, stable.algo = TRUE, ...)
Arguments

- **x**: a numeric vector, matrix, data.frame or grouped tibble (dplyr::grouped_df).
- **g**: a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group `x`.
- **w**: a numeric vector of (non-negative) weights, may contain missing values.
- **TRA**: an integer or quoted operator indicating the transformation to perform: 1 - "replace_fill" | 2 - "replace" | 3 - "-" | 4 - "-+" | 5 - "/" | 6 - "%" | 7 - "+" | 8 - "*" | 9 - "%%" | 10 - "-\%\%". See `TRA`.
- **na.rm**: logical. Skip missing values in `x`. Defaults to `TRUE` and implemented at very little computational cost. If `na.rm = FALSE` a NA is returned when encountered.
- **use.g.names**: make group-names and add to the result as names (vector method) or row-names (matrix and data.frame method). No row-names are generated for data.tables and grouped tibbles.
- **drop**: matrix and data.frame method: drop dimensions and return an atomic vector if `g = NULL` and `TRA = NULL`.
- **keep.group_vars**: grouped_df method: Logical. FALSE removes grouping variables after computation.
- **keep.w**: grouped_df method: Logical. Retain summed weighting variable after computation (if contained in grouped_df).
- **stable.algo**: logical. TRUE (default) use Welford's numerically stable online algorithm. FALSE implements a faster but numerically unstable one-pass method. See Details.
- **...**: arguments to be passed to or from other methods.

Details

Welford's online algorithm used by default to compute the variance is well described [here](#) (the section Weighted incremental algorithm also shows how the weighted variance is obtained by this algorithm).
If stable.algo = FALSE, the variance is computed in one-pass as \((\text{sum}(x^2) - n \times \text{mean}(x)^2) / (n-1)\), where \(\text{sum}(x^2)\) is the sum of squares from which the expected sum of squares \(n \times \text{mean}(x)^2\) is subtracted, normalized by \(n-1\) (Bessel's correction). This is numerically unstable if \(\text{sum}(x^2)\) and \(n \times \text{mean}(x)^2\) are large numbers very close together, which will be the case for large \(n\), large \(x\)-values and small variances (catastrophic cancellation occurs, leading to a loss of numeric precision). Numeric precision is however still maximized through the internal use of long doubles in C++, and the fast algorithm can be up to 4-times faster compared to Welford's method.

The weighted variance is computed with frequency weights as \((\text{sum}(x^2 \times w) - \sum(w) \times \text{weighted.mean}(x,w)^2) / (\sum(w) - 1)\). If na.rm = TRUE, missing values will be removed from both \(x\) and \(w\) i.e. utilizing only \(x[\text{complete.cases}(x,w)]\) and \(w[\text{complete.cases}(x,w)]\).

Missing-value removal as controlled by the na.rm argument is done very efficiently by simply skipping the values (thus setting na.rm = FALSE on data with no missing values doesn’t give extra speed). Large performance gains can nevertheless be achieved in the presence of missing values if na.rm = FALSE, since then the corresponding computation is terminated once a NA is encountered and NA is returned.

This all seamlessly generalizes to grouped computations, which are performed in a single pass (without splitting the data) and therefore extremely fast.

When applied to data frame’s with groups or drop = FALSE, fvar/fsd preserves all column attributes (such as variable labels) but does not distinguish between classed and unclassed object (thus applying fvar/fsd to a factor column will give a ‘malformed factor’ error, and applying it to a date variable will give an error or a pretty weird date). The attributes of the data.frame itself are also preserved.

Value

fvar returns the variance of \(x\), grouped by \(g\), or (if TRA is used) \(x\) transformed by its variance, grouped by \(g\). fsd computes the standard deviation of \(x\) in like manor.

See Also

Fast Statistical Functions, Collapse Overview

Examples

```r
## default vector method
fvar(mtcars$mpg)  # Simple variance (all examples also hold for fvar!)
fsd(mtcars$mpg)    # Simple standard deviation
fsd(mtcars$mpg, w = mtcars$hp) # Weighted sd: Weighted by hp
fsd(mtcars$mpg, TRA = "/") # Simple transformation: scaling (See also ?fscale)
fsd(mtcars$mpg, mtcars$cyl) # Grouped sd
fsd(mtcars$mpg, mtcars$cyl, mtcars$hp) # Grouped weighted sd
fsd(mtcars$mpg, mtcars$cyl, TRA = "/") # Scaling by group
fsd(mtcars$mpg, mtcars$cyl, mtcars$hp, "/") # Group-scaling using weighted group sds

## data.frame method
fsd(iris)          # This works, although 'Species' is a factor variable
fsd(mtcars, drop = FALSE) # This works, all columns are numeric variables
fsd(iris[-5], iris[[5]]) # By Species: iris[[5]] is still a list, and thus passed to GRP()
fsd(iris[-5], iris[[5]]) # Same thing much faster: fsd recognizes 'Species' is a factor
```
fSD(iris[-5], iris[[5]], TRA = "/") # Data scaled by species (see also fScale)

## matrix method
m <- qM(mtcars)
fSD(m)
fSD(m, mtcars$cyl) # etc...

## method for grouped tibbles - for use with dplyr:
library(dplyr)
mtcars %>% group_by(cyl,vs,am) %>% fSD
mtcars %>% group_by(cyl,vs,am) %>% fSD(keep.group_vars = FALSE) # remove grouping columns
mtcars %>% group_by(cyl,vs,am) %>% fSD(hp) # Weighted by hp
mtcars %>% group_by(cyl,vs,am) %>% fSD(hp, "/") # Weighted scaling transformation

GGDC10S  
Groningen Growth and Development Centre 10-Sector Database

Description
The GGDC 10-Sector Database provides a long-run internationally comparable dataset on sectoral productivity performance in Africa, Asia, and Latin America. Variables covered in the data set are annual series of value added (in local currency), and persons employed for 10 broad sectors.

Usage

data("GGDC10S")

Format
A data frame with 5027 observations on the following 16 variables.

Country char: Country (43 countries)
Regioncode char: ISO3 Region code
Region char: Region (6 World Regions)
Variable char: Variable (Value Added or Employment)
Year num: Year (67 Years, 1947-2013)
AGR num: Agriculture
MIN num: Mining
MAN num: Manufacturing
PU num: Utilities
CON num: Construction
WRT num: Trade, restaurants and hotels
TRA num: Transport, storage and communication
FIRE num: Finance, insurance, real estate and business services
GOV num: Government services
OTH num: Community, social and personal services
SUM num: Summation of sector GDP
Source

https://www.rug.nl/ggdc/productivity/10-sector/

References


See Also

wlddev, Collapse Overview

Examples

namlab(GGDC10S, class = TRUE)
qsu(GGDC10S, ~ Variable, ~ Variable + Country, vlabels = TRUE)

library(data.table)
library(ggplot2)

## Not run:
## World Regions Structural Change Plot

dat <- GGDC10S
get_vars(dat, 6:15) <- replace_outliers(sweep(get_vars(dat, 6:15), 1, dat$SUM, "/"), 0,
   single.limit = "min")
names(dat) <- stringr::str_wrap(vlabels(dat), width = 20)
dat$Variable <- Recode(dat$Variable,"VA"="Value Added Share","EMP"="Employment Share")
dat <- collap(dat, ~ Variable + Region + Year, cols = 6:15)
dat <- melt(qDT(dat), 1:3, variable.name = "Sector")

ggplot(aes(x = Year, y = value, fill = Sector), data = dat) +
   geom_area(position = "fill", alpha = 0.9) + labs(x = NULL, y = NULL) +
   theme_linedraw(base_size = 14) + facet_grid(Variable ~ Region, scales = "free_x") +
   scale_fill_manual(values = sub("#00FF66FF", "#00CC66", rainbow(10))) +
   scale_x_continuous(breaks = scales::pretty_breaks(n = 7), expand = c(0, 0))+
   scale_y_continuous(breaks = scales::pretty_breaks(n = 10), expand = c(0, 0),
   labels = scales::percent) +
   theme(axis.text.x = element_text(angle = 315, hjust = 0, margin = ggplot2::margin(t = 0)),
   strip.background = element_rect(colour = "grey30", fill = "grey30")

## End(Not run)

# A function to plot the structural change of an arbitrary country

plotGGDC <- function(ctrty) {
  dat <- qDT(GGDC10S)[Country == ctrty]
  dat <- cbind(get_vars(dat, c("Variable","Year")),
    replace_outliers(sweep(get_vars(dat, 6:15), 1, dat$SUM, "/"), 0, NA, "min"))
  dat$Variable <- Recode(dat$Variable,"VA"="Value Added Share","EMP"="Employment Share")
}

# Function to plot the structural change of an arbitrary country

plotGGDC(ctrty)
is.regular-is.unlistable

Regular Objects and Unlistable Lists

Description

A regular R object is an R object that is either atomic or a list - checked with is.regular. A (nested) list composed of regular objects at each level is unlistable - checked with is.unlistable.

Usage

is.regular(x)

is.unlistable(l)

Arguments

x a R object.

l a list.

Details

is.regular is simply defined as is.atomic(x) || is.list(x). is.unlistable is defined as all(unlist(rapply2d(l,is.regular),use.names = FALSE)). It could of course also be defined as all(rapply(l,is.atomic)), but the above is a lot more efficient if l contains data.frame's.

Value

logical(1) - TRUE or FALSE.

See Also

ldepth, has_elem, List Processing, Collapse Overview
Examples

is.regular(list(1,2))
is.regular(2)
is.regular(a ~ c)
l <- list(1, 2, list(3, 4, "b", FALSE))
is.regular(l)
is.unlistable(l)
l <- list(1, 2, list(3, 4, "b", FALSE, e ~ b))
is.regular(l)
is.unlistable(l)

ldepth

Determine the Depth / Level of Nesting of a List

Description

ldepth provides the depth of a list or list-like structure.

Usage

ldepth(l, DF.as.list = TRUE)

Arguments

l

a list.

DF.as.list

treat data.frame’s as sub-lists?

Details

The depth or level or nesting of a list or list-like structure (i.e. a classed object) is found by recursing down to the bottom of the list and adding an integer count of 1 for each level passed. For example the depth of a data.frame is 1. If a data.frame has list-columns, the depth is 2. However for reasons of efficiency, if l is not a data.frame and DF.as.list = TRUE, data.frame’s found inside l will not be checked for list column’s but assumed to have a depth of 1.

Value

A single integer indicating the depth of the list.

See Also

is.unlistable, has_elem, List Processing, Collapse Overview
Examples

```r
l = list(1, 2)
ldepth(l)
l = list(1, 2, mtcars)
ldepth(l)
l = list(l, DF.as.list = FALSE)
l = list(1, 2, list(4, 5, list(6, mtcars)))
ldepth(l)
l = list(l, DF.as.list = FALSE)
```

---

**psacf**  
Auto- and Cross- Covariance and -Correlation Function Estimation for Panel-Series

### Description

`psacf`, `pspacf` and `psccf` compute (and by default plot) estimates of the auto-, partial auto- and cross-correlation or covariance functions for panel-vectors and `plm::pseries`. They are analogues to `stats::acf`, `stats::pacf` and `stats::ccf`.

### Usage

```r
psacf(x, ...)  
pspacf(x, ...)  
psccf(x, y, ...)  
```

### Default S3 method:

```r
psacf(x, g, t = NULL, lag.max = NULL, type = c("correlation", "covariance","partial"),  
      plot = TRUE, gscale = TRUE, ...)  
pspacf(x, g, t = NULL, lag.max = NULL, plot = TRUE, gscale = TRUE, ...)  
psccf(x, y, g, t = NULL, lag.max = NULL, type = c("correlation", "covariance"),  
      plot = TRUE, gscale = TRUE, ...)  
```

### S3 method for class 'pseries'

```r
psacf(x, lag.max = NULL, type = c("correlation", "covariance","partial"),  
      plot = TRUE, gscale = TRUE, ...)  
pspacf(x, lag.max = NULL, plot = TRUE, gscale = TRUE, ...)  
psccf(x, y, lag.max = NULL, type = c("correlation", "covariance"),  
      plot = TRUE, gscale = TRUE, ...)  
```

### S3 method for class 'data.frame'

```r
psacf(x, by, t = NULL, cols = is.numeric, lag.max = NULL,  
      type = c("correlation", "covariance","partial"), plot = TRUE, gscale = TRUE, ...)  
```
Arguments

- **x, y**
  a numeric vector, panel-series (plm::pseries), data.frame or panel-data-frame (plm::pdata.frame).

- **g**
  a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x,y.

- **by**
  *data frame method*: Same input as `g`, but also allows one- or two-sided formulas using the variables in `x`, i.e. `~ idvar` or `var1 + var2 ~ idvar1 + idvar2`.

- **t**
  same input as `g`, to indicate the time-variable. For secure computations on unordered panel-vectors. Data frame method also takes one-sided formula i.e. `~time`.

- **cols**
  *data.frame method*: Select columns using a function, column names or indices. Note: cols is ignored if a two-sided formula is passed to by.

- **lag.max**
  maximum lag at which to calculate the acf. Default is `2*sqrt(length(x)/ng)` where `ng` is the number of groups in the panel-series / supplied to `g`.

- **type**
  character string giving the type of acf to be computed. Allowed values are "correlation" (the default), "covariance" or "partial".

- **plot**
  logical. If TRUE (the default) the acf is plotted.

- **gscale**
  logical. Do a groupwise scaling / standardization of `x,y` (using collapse::fscale and the groups supplied to `g`) before computing panel-autocovariances / correlations.

- **...**
  further arguments to be passed to stats::plot.acf.

Details

If `gscale = TRUE` data are standardized within each group (using collapse::fscale) such that the group-mean is 0 and the group-standard deviation is 1. This is strongly recommended for most panels to get rid of individual-specific heterogeneity which would corrupt the ACF computations.

After scaling, `psacf`, `pspacf` and `psccf` compute the ACF/CCF by creating a matrix of panel-lags of the series using collapse::flag and then correlating this matrix with the series (`x,y`) using stats::cor and pairwise-complete observations. This may require a lot of memory on large data, but is done because passing a sequence of lags to collapse::flag and thus calling collapse::flag and stats::cor one time is much faster than calling them `lag.max` times. The partial ACF is computed from the ACF in the same way as in stats::pacf.
Value

An object of class "acf", see ?stats::acf. The result is returned invisibly if plot is TRUE.

Note

For plm::pseries and plm::pdata.frame, the first index variable is taken to be the group-id and the second the time variable. If more than 2 index variables are attached to plm::pseries, the last one is taken as the time variable and the others are taken as group-id’s and interacted.

The pdata.frame method only works for properly subsetted objects of class 'pdata.frame'. A list of 'pseries' won’t work.

See Also

Time-Series and Panel-Series, Collapse Overview

Examples

```r
## World Development Panel Data
head(wlddev) # see also help(wlddev)
psacf(wlddev$PCGDP, wlddev$country, wlddev$year) # ACF of GDP per Capita
psacf(wlddev, PCGDP ~ country, ~year) # Same using data.frame method
psacf(wlddev$PCGDP, wlddev$country) # The Data is sorted, can omit t
pspacf(wlddev$PCGDP, wlddev$country) # Partial ACF
psccf(wlddev$PCGDP, wlddev$LIFEEX, wlddev$country) # CCF with Life-Expectancy at Birth

psacf(wlddev, PCGDP + LIFEEX + ODA ~ country, ~year) # ACF and CCF of GDP, LIFEEX and ODA
psacf(wlddev, ~ country, ~year, c(9:10,12)) # Same, using cols argument
pspacf(wlddev, ~ country, ~year, c(9:10,12)) # Partial ACF

## Using plm:
pwlddev <- plm::pdata.frame(wlddev, index = c("country","year"))# Creating a Panel-Data Frame
PCGDP <- pwlddev$PCGDP # Panel-Series of GDP per Capita
LIFEEX <- pwlddev$LIFEEX # Panel-Series of Life Expectancy
psacf(PCGDP) # Same as above, more parsimonious
pspacf(PCGDP)
psccf(PCGDP, LIFEEX)
psacf(pwlddev[,c(9:10,12)])
pspacf(pwlddev[,c(9:10,12)])
```

psmat

Matrix / Array from Panel-Series

Description

psmat efficiently expands a panel-vector or plm::pseries into a matrix. If a data frame or plm::pdata.frame is passed, psmat returns (default) a 3D array or a list of such matrices.
Usage

psmat(x, ...)

## Default S3 method:
psmat(x, g, t = NULL, transpose = FALSE, ...)

## S3 method for class 'pseries'
psmat(x, transpose = FALSE, ...)

## S3 method for class 'data.frame'
psmat(x, by, t = NULL, cols = NULL, transpose = FALSE, array = TRUE, ...)

## S3 method for class 'pdata.frame'
psmat(x, cols = NULL, transpose = FALSE, array = TRUE, ...)

## S3 method for class 'psmat'
plot(x, legend = FALSE, colours = legend, labs = NULL, ...)

Arguments

x a vector, panel-series (plm::pseries), data.frame or panel-data.frame (plm::pdata.frame).

g a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x. If the panel is balanced an integer indicating the number of groups can also be supplied. See Examples.

by data.frame method: Same input as g, but also allows one- or two-sided formulas using the variables in x, i.e. ~ idvar or var1 + var2 ~ idvar1 + idvar2.

t same inputs as g, to indicate the time-variable or second identifier(s). g and t together should fully identify the panel. If t = NULL, the data is assumed sorted and seq_col is used to generate rownames.

cols data.frame method: Select columns using a function, column names or indices.

Note: cols is ignored if a two-sided formula is passed to by.

transpose logical. TRUE generates the matrix such that g/by -> columns, t -> rows. Default is g/by -> rows, t -> columns.

array data.frame / pdata.frame methods: logical. TRUE returns a 3D array (if just one column is selected a matrix is returned). Otherwise always return a list of matrices.

... arguments to be passed to or from other methods, or for the plot method additional arguments passed to ts.plot.

legend logical. Automatically create a legend of panel-groups.

colours logical. Automatically colour by panel-groups.

labs provide a character-vector of variable labels / series titles when plotting an array.
Details

For \texttt{plm::pseries}, the first index variable is taken to be the group-id and the second the time variable. If more than 2 index variables are attached to \texttt{plm::pseries}, the last one is taken as the time variable and the others are taken as group-id’s and interacted.

Value

A matrix or 3D array containing the data in \(x\), where by default the rows constitute the groups-ids (g/by) and the columns the time variable or individual ids (t). 3D arrays contain the variables in the 3rd dimension. The objects have a class ‘psmat’, and also a ‘transpose’ attribute indicating whether \(\text{transpose} = \text{TRUE}\) or \(\text{transpose} = \text{FALSE}\).

Note

The \texttt{pdata.frame} method only works for properly subsetted objects of class ‘pdata.frame’. A list of ‘pseries’ won’t work. There also exist simple \texttt{aperm} and \(\text{[]}\) (subset) methods for ‘psmat’ objects. These differ from the default methods only by keeping the class and the ‘transpose’ attribute.

See Also

\texttt{Time-Series and Panel-Series, Collapse Overview}

Examples

```r
## World Development Panel Data
head(wlddev)  # View data
qsu(wlddev, pid = ~ iso3c, cols = 9:12, vlabels = TRUE)  # Sumarizing data
str(psmat(wlddev$PCGDP, wlddev$iso3c, wlddev$year))  # Generating matrix of GDP
r <- psmat(wlddev, PCGDP ~ iso3c, ~ year)  # Same thing using data.frame method
plot(r, main = vlabels(wlddev)[9], xlab = "Year")  # Plot the matrix
str(r)  # See strcture
str(psmat(wlddev$PCGDP, wlddev$iso3c))  # The Data is sorted, could omit t
str(psmat(wlddev$PCGDP, 216))  # This panel is also balanced, so
# ..indicating the number of groups would be sufficient to obtain a matrix

ar <- psmat(wlddev, ~ iso3c, ~ year, 9:12)  # Get array of transposed matrices
str(ar)
plot(ar)  # More legible and fancy plot
plot(psmat(collap(wlddev, ~region+year, cols = 9:12), ~region, ~year), legend = TRUE,
     labs = vlabels(wlddev)[9:12])

psml <- psmat(wlddev, ~ iso3c, ~ year, 9:12, array = FALSE)  # This gives list of ps-matrices
head(unlist2d(psml, "Variable", "Country", id.factor = TRUE))  # Using unlist2d, can generate DF

## Using plm simplifies things
pwlddev <- plm::pdata.frame(wlddev, index = c(”iso3c”, ”year”))  # Creating a Panel-Data Frame
PCGDP <- pwlddev$PCGDP  # A panel-Series of GDP per Capita
psmat(PCGDP)  # Same as above, more parsimonious
plot(psmat(PCGDP))
```

pwcor, pwcov, pwNobs  
Pairwise Correlations, Covariances and Observation Count

Description
Computes pairwise Pearsons correlations, covariances and observation counts. Pairwise correlations and covariances can be computed together with observation counts and p-values, and output as 3D array (default) or list of matrices. For an equivalent and faster implementation of `pwcor` see `Hmisc::rcorr` (written in Fortran). A major feature of `pwcor` and `pwcov` is their sophisticated print method.

Usage

```r
pwcor(X, ..., N = FALSE, P = FALSE, array = TRUE)
pwcov(X, ..., N = FALSE, P = FALSE, array = TRUE)
pwNobs(X)
```

Arguments

- `X`: a matrix or data.frame, for `pwcor` and `pwcov` all columns must be numeric.
- `x`: an object of class 'pwcor' / 'pwcov'.
- `N`: logical. TRUE also computes pairwise observation counts.
- `P`: logical. TRUE also computes pairwise p-values (same as `cor.test`).
- `array`: logical. If `N = TRUE` or `P = TRUE`, TRUE (default) returns output as 3D array whereas FALSE returns a list of matrices.
- `digits`: integer. The number of digits to round to in print.
- `sig.level`: numeric. P-value threshold below which a '*' is displayed above significant coefficients if `P = TRUE`.
- `show`: character. The part of the correlation / covariance matrix to display.
- `spacing`: integer. Controls the spacing between different reported quantities in the printout of the matrix: 0 - compressed, 1 - single space, 2 - double space.
- `...`: other arguments passed to `cor` or `cov`. Only sensible if `P = FALSE`.
Value

a numeric matrix, 3D array or list of matrices of the computed statistics. For \texttt{pwcor} and \texttt{pwcov} the object has a class 'pwcor' and 'pwcov', respectively.

See Also

\texttt{qsu}, \texttt{Collapse Overview}

Examples

```r
mna <- na_insert(mtcars)
pwcor(mna)
pwcov(mna)
pwNobs(mna)
pwcor(mna, N = TRUE)
pwcor(mna, P = TRUE)
pwcor(mna, N = TRUE, P = TRUE)
aperm(pwcor(mna, N = TRUE, P = TRUE))
print(pwcor(mna, N = TRUE, P = TRUE), digits = 3, sig.level = 0.01, show = "lower.tri")
pwcor(mna, N = TRUE, P = TRUE, array = FALSE)
print(pwcor(mna, N = TRUE, P = TRUE, array = FALSE), show = "lower.tri")
```

\texttt{qsu} \hspace{1cm} \textit{Fast (Grouped, Weighted) Summary Statistics for Cross-Sectional and Panel-Data}

Description

\texttt{qsu}, shorthand for quick-summary, is an extremely fast summary command inspired by the (xt)\texttt{summarize} command in the STATA statistical software.

It computes a set of 7 statistics (nobs, mean, sd, min, max, skewness and kurtosis) using a numerically stable one-pass method generalized from Welford's Algorithm. Statistics can be computed weighted, by groups, and also within-and between entities (for panel-data, see Details).

Usage

```r
## Default S3 method:
qsu(x, g = NULL, pid = NULL, w = NULL, higher = FALSE, array = TRUE, ...)

## S3 method for class 'matrix'
qsu(x, g = NULL, pid = NULL, w = NULL, higher = FALSE, array = TRUE, ...)

## S3 method for class 'data.frame'
qsu(x, by = NULL, pid = NULL, w = NULL, cols = NULL, 
    higher = FALSE, array = TRUE, vlabels = FALSE,...)
```
# Methods for compatibility with plm:

## S3 method for class 'pseries'
qsu(x, g = NULL, w = NULL, effect = 1L, higher = FALSE, array = TRUE, ...)

## S3 method for class 'pdata.frame'
qsu(x, by = NULL, w = NULL, cols = NULL, effect = 1L, higher = FALSE, array = TRUE, vlabels = FALSE, ...)

## S3 method for class 'qsu'
print(x, digits = 2, nonsci.digits = 9, na.print = "-", return = FALSE, print.gap = 2, ...)

### Arguments

- **x**: a numeric vector, matrix, data.frame, panel-series (plm::pseries) or panel-data.frame (plm::pdata.frame).
- **g**: a factor, GRP object, atomic vector (internally converted to factor) or a list of vectors / factors (internally converted to a GRP object) used to group x.
- **by** (p)data.frame method: Same as g, but also allows one- or two-sided formulas i.e. \(~ \text{group1 + group2 or var1 + var2 ~ group1 + group2}\). See Examples.
- **pid**: same input as g/by: Specify a panel-identifier to also compute statistics on between- and within-transformed data. data.frame method also supports one- or two-sided formulas. Transformations are taken independently from grouping with g/by (grouped statistics are computed on the transformed data). However, passing any LHS variables to pid will overwrite any LHS variables passed to by.
- **w**: a vector of (non-negative) weights. Adding weights will compute the weighted mean, sd, skewness and kurtosis, and transform the data using weighted individual means if pid is used.
- **cols**: select columns to summarize using column names, indices or a function (i.e. is.numeric). Two-sided formulas passed to by or pid overwrite cols.
- **higher**: logical. Add higher moments (skewness and kurtosis).
- **array**: logical. If computations have more than 2 dimensions (up to a maximum of 4D: variables, statistics, groups and panel-decomposition) output to array, else output (nested) list of matrices.
- **vlabels**: logical. Use variable labels in the summary. See vlabels.
- **effect** plm methods: Select which panel identifier should be used for between and within transformations of the data. 1L means first variable in the plm::index, 2L the second etc.. More than one variable can be supplied.
- **...**: arguments to be passed to or from other methods.
- **digits**: the number of digits to print after the comma/dot.
- **nonsci.digits**: the number of digits to print before resorting to scientific notation (default is to print out numbers with up to 9 digits and print larger numbers scientifically).
- **na.print**: character string to substitute for missing values.
return logical. Don’t print but instead return the formatted object.

print.gap integer. Spacing between printed columns. Passed to print.default.

**Details**

The algorithm used to compute statistics is well described [here](#) (see sections Welford’s online algorithm, Weighted incremental algorithm and Higher-order statistics). Skewness and kurtosis are calculated as described in Higher-order statistics and are mathematically identical to those implemented in the moments package. Just note that qsu computes the kurtosis (like moments::kurtosis), not the excess-kurtosis (= kurtosis - 3) defined in Higher-order statistics. The Weighted incremental algorithm described can easily be generalized to higher-order statistics).

Grouped computations specified with g/by are carried out extremely efficiently as in fsum (in a single pass, without splitting the data).

If pid is used, qsu performs a panel-decomposition of each variable and computes 3 sets of statistics: Statistics computed on the ’Overall’ (raw) data, statistics computed on the ’Between’ - transformed (pid - averaged) data, and statistics computed on the ’Within’ - transformed (pid - demeaned) data.

More formally, let \( x \) (bold) be a panel vector of data for \( N \) individuals indexed by \( i \), recorded for \( T \) periods, indexed by \( t \). \( x_{it} \) then denotes a single data-point belonging to individual \( i \) in time-period \( t \) (\( t/T \) must not represent time). Then \( \bar{x}_i \) denotes the average of all values for individual \( i \) (averaged over \( t \)), and by extension \( \bar{x}_N \) is the vector (length \( N \)) of such averages for all individuals.

If no groups are supplied to g/by, the ’Between’ statistics are computed on \( x_N \), the vector of individual averages. (This means that for a non-balanced panel or in the presence of missing values, the ’Overall’ mean computed on \( x \) can be slightly different than the ’Between’ mean computed on \( x_N \).) If groups are supplied to g/by, \( x_N \) is expanded to the vector \( x_i \) (length \( N \times T \)) by replacing each value \( x_{it} \) in \( x \) with \( x_{i} \), while preserving missing values in \( x \). Grouped Between-statistics are then computed on \( x_i \), with the only difference that the number ob observations (’Between-N’) reported for each group is the number of distinct non-missing values of \( x_i \) in each group (not the total number of non-missing values of \( x_i \) in each group, which is already reported in ’Overall-N’).

’Within’ statistics are always computed on the vector \( x - \bar{x}_i + \bar{x}_N \), where \( \bar{x} \) is simply the ’Overall’ mean computed from \( x \), which is added back to preserve the level of the data. The ’Within’ mean computed on this data will always be identical to the ’Overall’ mean. In the summary output, qsu reports not ’N’, which would be identical to the ’Overall-N’, but ’T’, the average number of time-periods of data available for each individual obtained as ’T’ = ’Overall-N’ / ’Between-N’. See Examples.

Apart from ’N/T’ and the extrema, the standard-deviations (’SD’) computed on between- and within- transformed data are extremely valuable because they indicate how much of the variation in a panel-variable is between-individuals and how much of the variation is within-individuals (over time). At the extremes, variables that have common values across individuals (such as the time-variable ’t’ in a balanced panel), can readily be identified as individual-invariant because the ’Between-SD’ on this variable is 0 and the ’Within-SD’ is equal to the ’Overall-SD’. Analogous, time-invariant individual characteristics (such as the individual-id ’i’) have a 0 ’Within-SD’ and a ’Between-SD’ equal to the ’Overall-SD’.

qsu comes with it’s own print method which by default writes out up to 9 digits at 2 decimal places. Larger numbers are printed in scientific format. for numbers between 7 and 9 digits, a comma ’,’ is placed after the 6th digit to designate the millions. Missing values are printed using ’-‘.
Value

A matrix, array or list of matrices of summary statistics. All matrices and arrays have a class 'qsu' and a class 'table' attached, responding i.e. to print.qsu and aperm.table...

Note

If weights \( w \) are used together with \( \text{pid} \), transformed data is computed using weighted individual means i.e. weighted \( x_i \) and weighted \( x \ldots \) Weighted statistics are subsequently computed on this weighted-transformed data.

See Also

descr, \texttt{pwcor}, Fast Statistical Functions, Collapse Overview

Examples

```r
## World Development Panel Data
# Simple Summaries --------------------------
qsu(wlddev)  # Simple summary
qsu(wlddev, vlabels = TRUE)  # Display variable labels
qsu(wlddev, higher = TRUE)  # Add skewness and kurtosis

# Grouped Summaries ------------------------
qsu(wlddev, ~ region, vlabels = TRUE)  # Statistics by World Bank Region
qsu(wlddev, PCGDP + LIFEEX ~ income)  # Summarize GDP per Capita and Life Expectancy by
stats <- qsu(wlddev, ~ region + income, cols = 9:10, higher = TRUE)  # Same variables, by both region and income
aperm(stats)  # A different perspective on the same stats

# Panel-Data Summaries ---------------------
qsu(wlddev, pid = ~ iso3c, vlabels = TRUE)  # Adding between and within countries statistics
# -> They show amongst other things that year and decade are individual-invariant,
# that we have GINI-data on only 161 countries, with only 8.42 observations per country on average,
# and that GDP, LIFEEX and GINI vary more between-countries, but ODA received varies more within
# countries over time.

# Using plm:
pwldev <- plm::pdata.frame(wlddev, index = c("iso3c","year"))
qsu(pwldev)  # Summary for pdata.frame -> qsu(wlddev, pid = ~ iso3c)
qsu(pwldev$PCGDP)  # Default summary for Panel-Series (class pseries)
qsu(G(pwldev$PCGDP))  # Summarizing GDP growth, see also ?G

# Grouped Panel-Data Summaries --------------
qsu(wlddev, ~ region, ~ iso3c, cols = 9:12)  # Panel-Statistics by region
psr <- qsu(pwldev, ~ region, cols = 9:12)  # Same on plm pdata.frame
psr  # -> Gives a 4D array
print.qsu(psr[,"N/T",,])  # Checking out the number of observations:
# In North america we only have 3 countries, for the GINI we only have 3.91 observations on average
# for 45 Sub-Saharan-African countries, etc...
print.qsu(psr[,"SD",,])  # Considering only standard deviations
```
In all regions variations in inequality (GINI) between countries are greater than variations in inequality within countries. The opposite is true for Life-Expectancy in all regions apart from Europe, etc...

```r
psrl <- qsu(wlddev, ~ region, ~ iso3c,  # Same, but output as nested list
cols = 9:12, array = FALSE)  
psrl  # We can use unlist2d to create a tidy data.frame
head(unlist2d(psrl, c("Variable","Trans"),
               row.names = "Region"))
```

Weighted Summaries -----------------------

```r
n <- nrow(wlddev)  
weights <- abs(rnorm(n))  # Generate random weights
qsu(wlddev, w = weights, higher = TRUE)  # Computed weighted mean, SD, skewness and kurtosis
weightsNA <- weights  # Weights may contain missing values... inserting 1000
weightsNA[sample.int(n, 1000)] <- NA
qsu(wlddev, w = weightsNA, higher = TRUE)  # But now these values are removed from all variables
```

Grouped and panel-summaries can also be weighted in the same manor

---

### rapply2d

Recursively Apply a Function to a List of Data Objects

#### Description

rapply2d is a recursive version of lapply with two key differences to rapply: (1) Data frames are considered as final objects, not as (sub-)lists, and (2) the result is not simplified.

#### Usage

```r
rapply2d(l, FUN, ...)
```

#### Arguments

- `l` a list.
- `FUN` a function that can be applied to all elements in l.
- `...` additional elements passed to FUN.

#### Value

A list of the same structure as `l`, where `FUN` was applied to all elements.

#### See Also

unlist2d, List Processing, Collapse Overview

#### Examples

```r
l <- list(mtcars, list(mtcars, as.matrix(mtcars)))
rapply2d(l, fmean)
unlist2d(rapply2d(l, fmean))```
**Transform Data by (Groupwise) Replacing or Sweeping out Statistics**

**Description**

TRA is an S3 generic that efficiently transforms data by either (column-wise) replacing data values with supplied statistics or sweeping the statistics out of the data. TRA supports grouped operations and data frame's, and is thus a generalization of sweep.

**Usage**

```r
TRA(x, STATS, FUN = "-", ...)  
## Default S3 method:
TRA(x, STATS, FUN = "-", g = NULL, ...)  
## S3 method for class 'matrix'
TRA(x, STATS, FUN = "-", g = NULL, ...)  
## S3 method for class 'data.frame'
TRA(x, STATS, FUN = "-", g = NULL, ...)  
## S3 method for class 'grouped_df'
TRA(x, STATS, FUN = "-", keep.group_vars = TRUE, ...)
```

**Arguments**

- **x**: a atomic vector, matrix, data frame or grouped tibble (dplyr::grouped_df).
- **STATS**: a matching set of summary statistics computed on x. If `g = NULL` (no groups), all methods support an atomic vector of statistics of length `NCOL(x)`. The matrix and data.frame methods also support a 1-row matrix or 1-row data.frame/list, respectively. If groups are supplied to `g`, STATS needs to be of the same type as `x` and of appropriate dimensions (such that `NCOL(x) == NCOL(STATS)` and `NROW(STATS)` matches the number of groups supplied to `g` i.e. the number of levels if `g` is a factor, with the first row of `STATS` corresponding to the first level of `g` etc...).
- **FUN**: an integer or character string indicating the operation to perform. There are 10 supported operations:

<table>
<thead>
<tr>
<th>Int.</th>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;replace_fill&quot;</td>
<td>replace and overwrite missing values</td>
</tr>
<tr>
<td>2</td>
<td>&quot;replace&quot;</td>
<td>replace but preserve missing values</td>
</tr>
<tr>
<td>3</td>
<td>&quot;-&quot;</td>
<td>subtract (i.e. center)</td>
</tr>
<tr>
<td>4</td>
<td>&quot;-+&quot;</td>
<td>subtract group-statistics but add group-frequency weighted average of group statistics (i.e. center)</td>
</tr>
<tr>
<td>5</td>
<td>&quot;/&quot;</td>
<td>divide (i.e. scale, but also changes mean. fscale can scale and keep mean)</td>
</tr>
<tr>
<td>6</td>
<td>&quot;%&quot;</td>
<td>compute percentages (i.e. divide and multiply by 100)</td>
</tr>
</tbody>
</table>
### Details

Without groups (`g = NULL`), TRA is nothing more than a column based version of base:`sweep, albeit 4-times more efficient on matrices and many times more efficient on data frames. TRA always preserves all attributes of `x`.

With groups passed to `g`, TRA expects (and checks for) a set of statistics such that `NROW(STATS)` equals the number of groups. If this condition is satisfied, TRA will (without further checks) assume that the first row of `STATS` is the set of statistics computed on the first group of `g`, the second row on the second group etc. and do groupwise replacing or sweeping out accordingly.

For example Let `x = c(1.2, 4.6, 2.5, 9.1, 8.7, 3.3)`, `g` is an integer vector in 3 groups `g = c(1, 3, 3, 2, 1, 2)` and `STATS = fmean(x, g) = c(4.95, 6.20, 3.55)`. Then `out = TRA(x, fmean(x, g), "-", g)` does the equivalent to the following for-loop: ```R for(i in 1:6) out[i] = x[i] - fmean(x, g)[g[i]]```.

Correct computation requires that `g` as used in `fmean` and `g` passed to TRA are exactly the same vector. Using `g = c(1, 3, 3, 2, 1, 2)` for `fmean` and `g = c(3, 1, 1, 2, 3, 2)` for TRA will not give the right result. The safest way of programming with TRA is thus to repeatedly employ the same factor or GRP object for all grouped computations. Atomic vectors passed to `g` will be converted to ordered factors (see `qF`) and lists will be converted to ordered GRP objects. This is also done by all Fast Statistical Functions and by default by `BY`, thus together with these functions, TRA can also safely be used with atomic- or list-groups. Problems may arise if other functions internally convert atomic vectors or lists to groups in a non-sorted way. `Note: as.factor` conversions are ok as this also involves sorting.

If `x` is a grouped tibble (grouped_df), TRA matches the columns of `x` and `STATS` and also checks for grouping columns names(attr(x,"groups")) in `x` and `STATS`. TRA, grouped_df will then only transform those columns in `x` for which matching counterparts were found in `STATS`, exempting grouping columns, and returns `x` again (with columns in the name order). If keep.group_vars = FALSE, the grouping columns are dropped after computation, however the "groups" attribute is not dropped (it can be removed using dplyr::ungroup()).
Value

x with columns replaced or swept out using STATS, grouped by g.

Note

I have tried to make TRA as redundant as possible by adding a TRA-argument to all Fast Statistical Functions (ensuring that the exact same grouping vector is used for aggregation and transformation), and by creating the B (between-transform) and W (within-transform) operators as well as fscale/STD as safer and faster solutions for frequent scaling, centering and averaging tasks. TRA is really a programmers function for cases when both aggregate statistics and transformed data need to be retained, or to work with more complex statistics (i.e. together with dapply or BY).

See Also
dapply, BY, Fast Statistical Functions, Data Transformations, Collapse Overview

Examples

```r
v <- iris$Sepal.Length # A numeric vector
f <- iris$Species # A factor
dat <- num_vars(iris) # Numeric columns
m <- qM(dat) # Matrix of numeric data

head(TRA(v, fmean(v))) # Simple centering [same as fmean(v, TRA = "-")) or W(v)]
head(TRA(m, fmean(m))) # [same as sweep(m, 2, fmean(m)), fmean(m, TRA = "-")) or W(m)]
head(TRA(dat, fmean(dat)))) # [same as fmean(dat, TRA = "-")) or W(dat)]
head(TRA(v, fmean(v), "replace")) # Simple replacing [same as fmean(v, TRA = "replace") or B(v)]
head(TRA(m, fmean(m), "replace")) # [same as sweep(m, 2, fmean(m)), fmean(m, TRA = 1L) or B(m)]
head(TRA(dat, fmean(dat), "replace")) # [same as fmean(dat, TRA = "replace") or B(dat)]
head(TRA(m, fsd(m), "/")) # Simple scaling... [same as fsd(m, TRA = "/")]

# Note: All grouped examples also apply for v and dat...  
head(TRA(m, fmean(m, f), "+", f)) # Centering [same as fmean(m, f, TRA = "-")) or W(m, f)]  
head(TRA(m, fmean(m, f), "replace", f)) # Replacing [same fmean(m, f, TRA = "replace") or B(m, f)]
head(TRA(m, fsd(m, f), "/", f)) # Scaling [same as fsd(m, f, TRA = "/")]

head(TRA(m, fmean(m, f), "+", f)) # Centering on the overall mean ...
  # [same as fmean(m, f, TRA = "+") or 
  #    W(m, f, mean = "overall.mean")]

head(TRA(m, fmean(m, f), "-", f), "+") # Also the same thing done manually !!
  fmean(m, "+")

# grouped tibble method
library(dplyr)
iris %>%
group_by(Species) %>% TRA(., fmean(.))
iris %>%
group_by(Species) %>% fmean(TRA = "-") # Same thing
iris %>%
group_by(Species) %>% TRA(., fmean(.)[c(2,4)]) # Only transforming 2 columns
iris %>%
group_by(Species) %>% TRA(., fmean(.)[c(2,4)], keep.group_vars = FALSE) # Dropping species column
Description

unlist2d efficiently unlists lists of regular R objects (objects built up from atomic elements) and creates a data.frame representation of the list. It is a faithful 2-dimensional generalization of base::unlist, and can also be understood as a recursive generalization of do.call(rbind,l), for lists of vectors, data.frames, arrays or heterogeneous objects.

Usage

unlist2d(l, idcols = ".id", row.names = FALSE, recursive = TRUE, id.factor = FALSE, DT = FALSE)

Arguments

l a unlistable list, see is.unlistable.

idcols a character stub or a vector of names for id-columns automatically added - one for each level of nesting in l. By default the stub is ".id", so columns will be of the form ".id.1", ".id.2", etc... if idcols = TRUE, the stub is also set to ".id". If idcols = FALSE, id-columns are omitted. The content of the id columns are the list names, or (if missing) integers for the list elements. Missing elements in asymmetric nested structures are filled up with NA. See examples.

row.names TRUE extracts row names from all the objects in l (where available) and adds them to the output in a column named "row.names". Alternatively, a column name i.e. row.names = "file" can be supplied.

recursive if FALSE, only process the lowest (deepest) level of l.

id.factor if TRUE and idcols != FALSE, create id columns as ordered factors instead of character or integer vectors. This is useful if id's are used for further analysis e.g. as inputs to ggplot2.

DT if TRUE, return a data.table, not a data.frame.

Details

The data.frame representation created by unlist2d is built as follows:

- Recurse down to the lowest level of the list-tree, data.frames are exempted and treated as a final elements.
- Check out the objects, if they are vectors, matrices or arrays convert them to data.frame (in the case of atomic vectors each element becomes a column).
- Row-bind these data.frame’s using data.table’s rbindlist function. Columns are matched by name. If the number of columns differ, fill empty spaces with NA’s. Create an id-column on the left, filled with the object names or indices (if unnamed). If row.names = TRUE, store row.names of the objects (if available) in a separate column.
• Move up to the next higher level of the list-tree and repeat: Convert atomic objects to data.frame and row-bind while matching all columns and filling unmatched ones with NA’s. Create another id-column for each level of nesting passed through. If the list-tree is asymmetric, fill empty spaces in lower-level id columns with NA’s.

The result of this iterative procedure is a single data.frame containing on the left side id-columns for each level of nesting (from higher to lower level), followed by a column containing all the row.names of the objects if row.names = TRUE, followed by the object columns, matched at each level of recursion. Optimal results are of course obtained with symmetric lists of arrays, matrices or data.frames, which unlist2d nicely converts to a beautiful data.frame ready for plotting or further analysis.

Value

A data.frame or (if DT = TRUE) data.table.

See Also

rapply2d, List Processing, Collapse Overview

Examples

```r
## basic examples:
l <- list(mtcars, list(mtcars, mtcars))
unlist2d(l)
unlist2d(rapply2d(l, fmean))
l = list(a = qM(mtcars[1:8]),
       b = list(c = mtcars[4:11], d = list(e = mtcars[2:10], f = mtcars)))
unlist2d(l, row.names = TRUE)
unlist2d(rapply2d(l, fmean))
unlist2d(rapply2d(l, fmean), recursive = FALSE)

## Groningen Growth and Development Center 10-Sector Database
head(GGDC10S) # See ?GGDC10S
namlab(GGDC10S, class = TRUE)

# Panel-Summarize this data by Variable (Employment and Value Added)
l <- qsu(GGDC10S, by = ~ Variable, # Output as list (instead of 4D array)
         pid = ~ Variable + Country,
         cols = 6:16, array = FALSE)
str(l) # A list of 2-levels with matrices of statistics
head(unlist2d(l)) # Default output, missing the variables (row-names)
head(unlist2d(l, row.names = TRUE)) # Here we go, but this is still not very nice
head(unlist2d(l, idcols = c("Sector","Trans"), # Now this is looking pretty good
           row.names = "Variable"))

dat <- unlist2d(l, c("Sector","Trans"), # Id-columns can also be generated as ordered factors
                "Variable", id.factor = TRUE)
str(dat)

# Split this sectoral data, first by Variable (Employment and Value Added), then by Country
sdat <- rapply2d(split(GGDC10S[,1:16], GGDC10S$Variable), function(x) split(x[,6:16], GGDC10S$Variable))
```

# Compute pairwise correlations between sectors and recombine:
dat <- unlist2d(rapply2d(sdat, pwcor),
               idcols = c("Variable","Country"),
               row.names = "Sector")
head(dat)
plot(hclust(as.dist(1-pwcor(dat[-(1:3)]))))  # Using corrs. as distance metric to cluster sectors

# Together with other functions like psmat, unlist2d can also effectively help reshape data:
head(unlist2d(psmat(subset(GGDC10S, Variable == "VA"), ~Country, ~Year, cols = 6:16, array = FALSE),
          idcols = "Sector", row.names = "Country"))

---

**wlddev**   

**World Bank Development Data**

*Description*

This dataset contains 4 indicators from the World Bank's World Development Indicators (WDI) database: (1) GDP per capita, (2) Life expectancy at birth, (3) GINI index and (4) Net ODA received. The panel-data is balanced and covers 216 present and historic countries from 1960-2018 (World Bank aggregates and regional entities are excluded). Apart from the indicators the data contains a number of identifiers (character country name, factor ISO3 country code, World Bank region and income level, numeric year and decade) and 2 generated variables: A logical variable indicating whether the country is an OECD member, and a fictional variable stating the date the data was recorded. These variables were added so that all common data-types are represented in this dataset, making it an ideal test-dataset for certain collapse functions.

*Usage*

```r
data("wlddev")
```

*Format*

A data frame with 12744 observations on the following 12 variables. All variables are labelled e.g. have a 'label' attribute.

- **country** chr  Country Name
- **iso3c** fct  Country Code
- **date** date  Date Recorded (fictional)
- **year** num  Year
- **decade** num  Decade
- **region** fct  World Bank Region
- **income** fct  World Bank Income Level
- **OECD** log  Is OECD Member Country?
PCGDP  
num GDP per capita (constant 2010 US$)

LIFEEX  
num Life expectancy at birth, total (years)

GINI  
um GINI index (World Bank estimate)

ODA  
um Net ODA received (constant 2015 US$)

Source


See Also

GGDC10S, Collapse Overview

Examples

data(wlddev)

# Panel-summarizing the 4 series
qsu(wlddev, pid = ~iso3c, cols = 9:12, vlabels = TRUE)

# By Region
qsu(wlddev, by = ~region, cols = 9:12, vlabels = TRUE)

# Panel-summary by region
qsu(wlddev, by = ~region, pid = ~iso3c, cols = 9:12, vlabels = TRUE)

# Pairwise correlations: Overall
print(pwcor(get_vars(wlddev, 9:12), N = TRUE, P = TRUE), show = "lower.tri")

# Pairwise correlations: Between Countries
print(pwcor(fmean(get_vars(wlddev, 9:12), wlddev$iso3c), N = TRUE, P = TRUE), show = "lower.tri")

# Pairwise correlations: Within Countries
print(pwcor(fwithin(get_vars(wlddev, 9:12), wlddev$iso3c), N = TRUE, P = TRUE), show = "lower.tri")
Index

*Topic **array**
  psmat, 91

*Topic **attribute**
  AA2-small-helpers, 28

*Topic **datasets**
  GGDC10S, 85
  wlddev, 105

*Topic **documentation**
  A0-collapse-documentation, 8
  A1-fast-statistical-functions, 9
  A2-fast-grouping, 11
  A3-select-replace-vars, 14
  A4-quick-conversion, 17
  A6-data-transformations, 22
  A7-time-series-panel-series, 23
  A8-list-processing, 24
  A9-summary-statistics, 26
  AA1-recode-replace, 26
  AA2-small-helpers, 28
  qsu, 95

*Topic **htest**
  fFtest, 50

*Topic **list**
  A8-list-processing, 24
  extract-list, 37
  is.regular-is.unlistable, 87
  ldepth, 88
  rapply2d, 99
  unlist2d, 103

*Topic **manip**
  A0-collapse-documentation, 8
  A1-fast-statistical-functions, 9
  A2-fast-grouping, 11
  A3-select-replace-vars, 14
  A4-quick-conversion, 17
  A5-advanced-aggregation, 18
  A6-data-transformations, 22
  A7-time-series-panel-series, 23
  A8-list-processing, 24
  A9-summary-statistics, 26
  AA1-recode-replace, 26
  BY, 30
  collapse-package, 3
  dapply, 33
  extract-list, 37
  fbetween-fwithin-B-W, 39
  fdiff-fgrowth-D-G, 43
  ffirst-flast, 48
  fHDbetween-fHDwithin-HDB-HDW, 52
  flag-L-F, 56
  fmax-fmin, 60
  fmean, 63
  fmedian, 65
  fmode, 67
  fnNdistinct, 69
  fnNobs, 71
  fprod, 73
  fscale-STD, 75
  fsum, 79
  fvar-fsd, 82
  psacf, 89
  psmat, 91
  rapply2d, 99
  TRA, 100
  unlist2d, 103

*Topic **misc**
  AA2-small-helpers, 28

*Topic **multivariate**
  fHDbetween-fHDwithin-HDB-HDW, 52
  pwcor, pwcov, pwNobs, 94

*Topic **package**
  collapse-package, 3

*Topic **ts**
  A7-time-series-panel-series, 23
  A9-summary-statistics, 26
  fdiff-fgrowth-D-G, 43
  flag-L-F, 56
  psacf, 89

107
psmat, 91

*Topic univar
  A1-fast-statistical-functions, 9
descr, 35
ffirst-flate, 48
fmax-fmin, 60
fmean, 63
fmedian, 65
fmode, 67
fNdistinct, 69
fNobs, 71
fprod, 73
fscale=STD, 75
fsum, 79
fvar-fsd, 82
qsu, 95

*Topic utilities
  AA2-small-helpers, 28

_COLLAPSE_ALL
  (A0-collapse-documentation), 8
_COLLAPSE_DATA
  (A0-collapse-documentation), 8
_COLLAPSE GENERIC
  (A0-collapse-documentation), 8
_COLLAPSE_TOPICS
  (A0-collapse-documentation), 8
_FAST_FUN
  (A1-fast-statistical-functions), 9
_FAST_STAT_FUN
  (A1-fast-statistical-functions), 9
OPERATOR_FUN
  (A6-data-transformations), 22
_.psmat (psmat), 91
%!in% (AA2-small-helpers), 28

A0-collapse-documentation, 8
A1-fast-statistical-functions, 9
A2-fast-grouping, 11
A3-select-replace-vars, 14
A4-quick-conversion, 17
A5-advanced-aggregation, 18
A6-data-transformations, 22
A7-time-series-panel-series, 23
A8-list-processing, 24
A9-summary-statistics, 26
AA1-recode-replace, 26
AA2-small-helpers, 28
add_stub (AA2-small-helpers), 28
add_vars (A3-select-replace-vars), 14
add_vars<- (A3-select-replace-vars), 14
Advanced Data Aggregation, 8
all_identical (AA2-small-helpers), 28
all_obj_equal (AA2-small-helpers), 28
aperm.psmat (psmat), 91
as.character, 13
as.character_factor
  (AA2-small-helpers), 28
as.data.frame.descr (descr), 35
as.factor.GRP
  (A2-fast-grouping), 11
as.numeric_factor (AA2-small-helpers), 28
atomic_elem, 8, 25
atomic_elem (extract-list), 37
atomic_elem<- (extract-list), 37
av (A3-select-replace-vars), 14
av<- (A3-select-replace-vars), 14
B, 102
B (fbetween-fwithin-B-W), 39
BY, 8, 11, 17, 20, 22, 23, 30, 34, 101, 102
cat_vars (A3-select-replace-vars), 14
cat_vars<- (A3-select-replace-vars), 14
char_vars (A3-select-replace-vars), 14
char_vars<- (A3-select-replace-vars), 14
ckmatch (AA2-small-helpers), 28
collap, 11, 22, 23, 32, 34
collap (A5-advanced-aggregation), 18
collap (A5-advanced-aggregation), 18
collapse, 8, 18
collapse (collapse-package), 3
Collapse Documentation & Overview, 4
Collapse Overview, 4, 11, 14, 16, 18, 20, 23,
  24, 26, 28, 29, 32, 34, 36, 39, 42, 47,
  50, 51, 56, 59, 62, 64, 66, 69, 71, 72,
  75, 78, 81, 84, 86–88, 91, 93, 95, 98,
  99, 102, 104, 106
collapse-documentation
  (A0-collapse-documentation), 8
collapse-package, 3, 9
collapv (A5-advanced-aggregation), 18
cor, 94
cor.test, 94
cov, 94
D, 24
Data Transformations, 8, 17, 22, 23, 31, 32, 33, 102
D (fdiff-fgrowth-D-G), 43
dapply, 8, 17, 22, 23, 31, 32, 33, 102
Date, 35
Date_vars (A3-select-replace-vars), 14
descr. 8, 26, 35, 98
documentation, 4
extract-list, 37
f (flag-L-F), 56
fact_vars (A3-select-replace-vars), 14
Fast Grouping, 8
Fast Statistical Function, 20
Fast Statistical Functions, 8, 13, 18, 20, 22, 23, 26, 31, 32, 34, 36, 39, 43, 50, 56, 62, 64, 66, 69, 71, 72, 75, 78, 81, 84, 98, 101, 102
fbetween (fbetween-fwithin-B-W), 39
fbetween-fwithin-B-W, 39
fbetween/B, 8, 9, 22, 23
fbetween/B and fwithin/W, 56
fdiff, 24
fdiff (fdiff-fgrowth-D-G), 43
fdiff-fgrowth-D-G, 43
fdiff/B, 8, 9, 23, 24, 59
ffirst, 8, 9
ffirst (ffirst-flast), 48
ffirst-flast, 48
ffttest, 8, 23, 50, 56
fgrowth, 24
fkgrowth (fdiff-fgrowth-D-G), 43
fkgrowth/G, 8, 9, 23, 24, 59
fHDbetween
   (fHDbetween-fHDbetween-HDB-HDW), 52
fHDbetween-fHDbetween-HDB-HDW, 52
fHDbetween/fHDbetween-HDB-HDW, 52
fHDbetween/HDB, 8, 9, 22, 23
fHDbetween/HDB and fHDbetween/HDW, 42, 51
fHDbetween/HDW, 23, 51
fHDbetween
   (fHDbetween-fHDbetween-HDB-HDW), 52
fHDbetween/HDW, 8, 9, 22, 23
finteraction (AA2-small-helpers), 28
has_elem, 8, 24, 25, 87, 88
has_elem (extract-list), 37
HDB (fHDbetween-fHDwithin-HDB-HDW), 52
HDW (fHDbetween-fHDwithin-HDB-HDW), 52

irreg_elem, 8, 25
irreg_elem (extract-list), 37
is.categorical, 19
is.categorical (AA2-small-helpers), 28
is.Date (AA2-small-helpers), 28
is.GRP (A2-fast-grouping), 11
is.regular, 8, 24, 25, 37, 38
is.regular (is.regular-is.unlistable), 87
is.regular-is.unlistable, 87
is.unlistable, 8, 24, 25, 37, 88, 103
is.unlistable (is.regular-is.unlistable), 87

L, 23
L (flag-L-F), 56
ldepth, 8, 24, 25, 87, 88
List Processing, 8, 39, 87, 88, 99, 104
list_elem, 8, 25
list_elem (extract-list), 37
list_elem<-(extract-list), 37
logi_vars (A3-select-replace-vars), 14
logi_vars<-(A3-select-replace-vars), 14

mctl, 34
mctl (A4-quick-conversion), 17
mrtl, 34
mrtl (A4-quick-conversion), 17

na_insert (AA2-small-helpers), 28
na_rm (AA2-small-helpers), 28
namlab (AA2-small-helpers), 28
num_vars (A3-select-replace-vars), 14
num_vars<-(A3-select-replace-vars), 14
nv (A3-select-replace-vars), 14
nv<-(A3-select-replace-vars), 14

plot.GRP (A2-fast-grouping), 11
plot.psmat (psmat), 91
print.descr (descr), 35
print.GRP (A2-fast-grouping), 11
print.pwcov (pwcov, pwNobs), 94
print.pwcov (pwcov, pwNobs), 94
print.qsu (qsu), 95

psacf, 8, 24, 89
psccf, 8, 24
psccf (psacf), 89
psmat, 8, 24, 91
pspacf, 8, 24
pspacf (psacf), 89
pwcor, 8, 26, 36, 98
pwcor (pwcor, pwcov, pwNobs), 94
pwcor, pwcov, pwNobs, 94
pwcov, 8, 26
pwcov (pwcov, pwcov, pwNobs), 94
pwNobs, 8, 26
pwNobs (pwcov, pwcov, pwNobs), 94
qDF, 19, 35
qDF (A4-quick-conversion), 17
qDT (A4-quick-conversion), 17
qF, 13, 14, 31, 101
qF (A4-quick-conversion), 17
qG, 13, 14
qG (A4-quick-conversion), 17
qM (A4-quick-conversion), 17
qsu, 8, 26, 35, 36, 95, 95
qsu.default, 35, 36
quantile, 31, 35
Quick Data Conversion, 8
Quick Select and Replace Variables, 8

rapply2d, 8, 25, 99, 104
Recode, 28, 29
Recode (AA1-recode-replace), 26
Recode and Replace Values, 8
reg_elem, 8, 25
reg_elem (extract-list), 37
replace_non_finite
 (AA1-recode-replace), 26
replace_outliers (AA1-recode-replace), 26
rm_stub (AA2-small-helpers), 28
seq_col (AA2-small-helpers), 28
seq_row (AA2-small-helpers), 28
setColnames (AA2-small-helpers), 28
setDimnames (AA2-small-helpers), 28
setRownames (AA2-small-helpers), 28
Small (Helper) Functions, 8, 28
STD (fscale-STD), 75
sub, 29
Summary Statistics, 8
sweep, 100

table, 35
Time-Series and Panel-Series, 8, 9, 11, 23, 47, 59, 91, 93
TRA, 8, 10, 11, 22, 23, 42, 48–50, 56, 60–75, 78–80, 82–84, 100

unlist2d, 8, 25, 99, 103

vclasses (AA2-small-helpers), 28
vlabels, 96
vlabels (AA2-small-helpers), 28
vlabels<- (AA2-small-helpers), 28
vtypes (AA2-small-helpers), 28

W, 102
W (fbetween-fwithin-B-W), 39
wlddev, 8, 86, 105