

Package ‘bojan’

October 20, 2008

Version 0.2-6

Date 2008-10-20

Title Michal Bojanowski’s R package

Author Michal Bojanowski <mbojan@ifispan.waw.pl>

Maintainer Michal Bojanowski <mbojan@ifispan.waw.pl>

Depends R (>= 2.2.1), methods

Suggests foreign, sna, RNetCDF

Description A collection of functions written by me for various tasks. Includes also some datasets picked up “along the way” that may prove usefull in the future. See ?bojan for more detailed information.

License GPL version 2 or newer

R topics documented:

EcheniqueFryer	2
Inflacja	3
White	4
anova.princomp	5
bojan-deprecated	6
bojan-package	7
carryover	10
ccties	11
coef.princomp	12
dePolish	13
dispplot	14
dynetSimple	15
etas	16
freeman	18
graphCoords	19
iCurveFit	20
is.symmetric	21
isGraphical	22
lss	24
ncinfo-deprecated	25

nullModel	27
panel.model	28
placement	29
plotGraph	30
rescale	31
see-deprecated	32
show.keywords-deprecated	33
sourceall-deprecated	34
speedcomp-deprecated	34
ssi	35
subst-deprecated	37
toWord	38
wcalc	40

Index	41
--------------	-----------

EcheniqueFryer	<i>Example data from Echenique and Fryer (2006)</i>
----------------	---

Description

Example data from Echenique & Fryer (2006) representing a city with black and white neighbourhoods.

Usage

```
data(EcheniqueFryer)
```

Format

The format is a list with two components:

```
network a 30x30 matrix containing the network data
type a numeric vector of length 30 designating the race
```

Details

This data is taken from Echenique & Fryer (2006, figure III). The data represent a fictional city composed of 30 neighborhoods that are either black or white.

The data is a list of two components. The `network` component contains a network adjacency matrix that represents the grid structure of neighborhood proximities. For a given pair of two neighborhoods `i` and `j` if the entry `network[i, j]` is equal to 0 then they are not adjacent. Non-zero value mean that these two neighborhoods are adjacent. The non-zero entries are rescaled so that for every neighborhood they sum to 1.

The second component `type` defines a type of the neighborhood, 1 is “black”, 2 is “white”.

Source

Echenique, Federico and Roland G. Fryer, Jr. (2006) A Measure of Segregation Based On Social Interactions

Examples

```

data(EcheniqueFryer)

# ordinary graph
require(sna)
gplot( EcheniqueFryer$network, gmode="graph",
       vertex.col=EcheniqueFryer$type )

# Rough reproduction E&F's figure III
gplot( EcheniqueFryer$network, mode="hall", jitter=FALSE, gmode="graph",
       vertex.col=EcheniqueFryer$type, displaylabels=TRUE,
       sub="Source: Echenique & Fryer (2006)",
       main="Neighborhood racial segregation\n in a fictional city" )

```

Inflacja

Monthly inflation rates in Poland

Description

Data on monthly inflation rates in Poland since January 1989 up to March 2005.

Usage

```
data(Inflacja)
```

Format

A data frame with 195 observations on the following 6 variables.

year Year

month Month

infl11 a time-series, inflation as compared to the same month of the previous year

infl12 a time-series, inflation as compared to the previous month

infl13 a time-series, inflation as compared to December of the previous year

infl14 a time-series, average inflation of the previous 12 months

Source

(Polish) Central Statistical Office, <http://www.stat.gov.pl>

Examples

```

data(Inflacja)

### transform inflX variables into time-series objects
Inflacja$infl1ts <- ts(Inflacja$infl1, start=c(1989,1), end=c(2005,3),
                      freq=12)
Inflacja$infl2ts <- ts(Inflacja$infl2, start=c(1989,1), end=c(2005,3),
                      freq=12)
Inflacja$infl3ts <- ts(Inflacja$infl3, start=c(1989,1), end=c(2005,3),
                      freq=12)
Inflacja$infl4ts <- ts(Inflacja$infl4, start=c(1989,1), end=c(2005,3),

```

```

      freq=12)

### plot some...
plot( Inflacja$infl11ts, main="Inflation rates in Poland",
      ylab="Inflation")
lines( Inflacja$infl14ts, lty=2 )
legend( 1995, 1200,
        legend=c("compared to year ago","average of last 12 months"),
        lty=1:2 )

```

 White

White's data on Effective Kinship Networks

Description

This data is taken from Freeman (1978) who uses data from White (1975) to illustrate the segregation measure.

Usage

```
data(White)
```

Format

The format is a list with two components:

```

network 10x10 binary adjacency matrix
gender  numeric vector of length 10 designating gender

```

Details

Based on Freeman (1978):

White dealt with the problem of segregation among social positions rather than among individual persons. He specified a set of standard kinship positions that he called the “effective kinship network”.

Traditional analysis (e.g. Murdock, 1971) have argued that societies sometimes proscribe interaction among some kinship positions as an extension of incest taboos. Thus, given this reasoning, kinship positions should be segregated according to the gender of their occupants. White’s data provide possibility to test of this hypothesis.

White collected data on the rules governing various kinds of interaction among occupants of his ten standard kinship positions for a sample of 219 societies. For every pair of positions White specified whether or not interaction between their occupants was ever restricted in any society in the sample.

Source

Freeman, Linton C. (1978) "Segregation in Social Networks" *Sociological Methods and Research* 6(4):411–429

References

- Freeman, Linton C. (1978) "Segregation in Social Networks" *Sociological Methods and Research* 6(4):411–429
- Murdock, G. P. (1971) "Cross-Sex Patterns of Kin Behavior" *Ethnology* 1: 359–368
- White, D. R. (1975) "Communicative Avoidance in Social Networks". University of California, Irvine. (mimeo)

Examples

```
data(White)
str(White)

if( require(sna) )
{
  pos <- gplot( White$network, vertex.col=White$gender,
              gmode="graph", displaylabels=TRUE, label=dimnames(White$network)[[1]],
              label.cex=.7, boxed.labels=FALSE,
              main="White's (1975) data on kinship networks",
              sub="Red=Males, Black=Females")
}
```

 anova.princomp

Component variation in PCA analysis

Description

Calculate percent of explained variance for Principal Component Analysis objects.

Usage

```
## S3 method for class 'princomp':
anova(object, ...)
```

Arguments

object	object of class princomp
...	other arguments currently not supported

Details

The function calculates percentages of variance explained for each component resulting from Principal Component Analysis. The variance is equal to:

$$100 \frac{\sigma_i^2}{\sum_i \sigma_i^2}$$

where σ is equal to component's standard deviation.

Value

A vector of length equal to a number of components existing in `object`. Each element contains the percent of variance explained by the corresponding component.

See Also

[princomp](#), [anova](#)

Examples

```
pcmodel <- princomp( USArrests, cor=TRUE )
pcmodel
anova(pcmodel) # variance prc.
```

bojan-deprecated *Deprecated Functions in Bojan package*

Description

These functions are provided for compatibility with older versions of package **bojan** and may be defunct as soon as the next release.

Usage

```
ncinfo(object)
see(o, ...)
## S4 method for signature 'ANY':
see(o, ...)
## S4 method for signature 'data.frame':
see(o, ...)
show.keywords(file = NULL)
sourceall(dir = getwd(), pattern = ".*\\.R", ...)
speedcomp(...)
subst(object, a, b)
## S3 method for class 'list':
subst(object, a, b=NA)
```

Arguments

object	object of class "RNetCDF", or a name of the file
o	object to be inspected
file	path to file with keywords, defaults to 'RHOME/doc/KEYWORDS'
dir	character, the directory to look for the files. Defaults to the working directory
pattern	character, the regular expression that defines the pattern of the files to be sourced. Defaults to all files with ".R" extension.
a	value to be substituted
b	value to substitute a's with, defaults to NA
...	arguments passed to other methods

Details

The original help page for these functions is often available at `help("oldName-deprecated")` (note the quotes). Functions in packages other than the base package are listed in `help("pkg-deprecated")`.

`ncinfo`, `see`, `show.keywords`, `sourceall`, and `speedcomp` were moved to package **mbtools**. `subst` is superseded by [replace](#) in package **base**.

See Also

[Deprecated](#), [base-defunct](#)

bojan-package	<i>Michal Bojanowski's collection of function hopefully useful also to others</i>
---------------	---

Description

This package contains some functions, datasets and tools that I created and collected “along the way”.

Details

Among other things the package contains:

- Functions for data analysis, especially for social networks
- Some illustrative datasets
- Other statistical data-management and utility functions which proved useful at some point and may be useful in the future.

This package is, and probably will be, and ever-growing collection of functions and data. Some of the routines are to be considered *experimental*. The list of improvements is only a little bit shorter than the list of new features I want to add. Also, the documentation is very scarce in various places. Type `help(package="bojan")` for the list of available functions.

If you find this package useful in your work please cite it. Type `citation(package="bojan")` for the information how to do that.

Enjoy!

Changes**Version 0.2-6** (2008-10-20)

- Some functions are moved to package **mbtools** and so are deprecated in package **bojan**. These include: [ncinfo](#), [see](#), [show.keywords](#), [sourceall](#), [speedcomp](#).
- Because of the above the package no longer suggests **RNetCDF**.
- Added function `nullModel` for calculating null single level regression models (pure random intercept model) using closed-form formulas. Based on Snijders & Bosker.
- Simplified the CITATION file.

Version 0.2-5 (2008-08-27)

- Updated version of [see](#). Method for functions is now a default method: signature ANY. Also, uses [file.show](#) for displaying the dumped object instead of [edit](#).

Version 0.2-4 (2008-04-02)

- Corrected code of `graphicalIneq` which is a part of [isGraphical](#) checks.

Version 0.2-3 (2008-03-12)

- Exported [tcb](#), which I forgot in the previous release...

- Examples of `toWord` are now executed. They were expanded substantively as well.

Version 0.2-2 (2008-03-12)

- Corrected the `toWord` method for data frames.
- Added default method for `toWord` which tries to coerce the argument to matrix.
- Created an alias `tcb` (To ClipBoard) for `toWord`.

Version 0.2-1 (2008-03-05)

- Corrected code for `isGraphical` so now the default is appropriately picked to be "tv".
- Fixed NAMESPACE which now correctly exports `speedcomp` class.
- Some documentation fixes.

Version 0.2-0 (2008-03-02)

- Added function `ncinfo` for getting summary information about NetCDF files (using package `RNetCDF`).
- Added function `isGraphical` for testing for degree sequences.
- Changed the URL in help pages to <http://bojan.3e.pl/weblog>
- Some documentation fixes.

Version 0.1-0 (2008-02-29)

- Added function `panel.model` for drawing model-fitted lines on panels of `coplot` and `pairs`.

Version 0.0-15 (2007-12-03)

- Forgot to export the `see` function

Version 0.0-14 (2007-12-02)

- Added function `see` for inspecting objects, especially functions.

Version 0.0-13 (2007-10-29)

- Added function `rescale` for simple linear rescaling of numeric vectors to new interval.
- Added function `dynetSimple` for simple on-screen visualization of network dynamics with a constantset of vertices.

Version 0.0-12 (2007-10-13)

- Added function `iCurveFit` for interactive curve fitting to supplied set of points.
- Added function `lss`, an extended version of `ls` that prints some more information about the objects in the specified environment.
- Added package-specific menu in Windows GUI version of R. Through a menu "bojan" it is possible to access some functions usefull especially in package development. Functions accessible now is `show.keywords` and `lss`.
- Update help files for `speedcomp` and some other.

Version 0.0-11 (2007-10-05)

- Argument `alg` to function `graphCoords` now accepts either `NULL` or character name of the placement algorithm which is looked up with `get`

- `plotGraph` is now S4 generic with methods for graph coordinates supplied as matrix, character name of the placement function or nothing.

Version 0.0-10 (2007-05-14)

- Corrected code of `freeman` the function now properly calculates the index if the argument `bDist` is not `NULL`.
- New version of `dePolish` that uses Polish letters loaded from separate file (got rid of warnings issued during package building/checking). Now it is also possible to process Polish letters in different encodings(Thanks to Brian Ripley for suggestions!).
- New function `speedcomp` for comparing speeds of several R expressions.

Version 0.0-9 (2007-04-26)

- Added function `ccties` for cross-classification of ties in the network depending on the pre-defined types of nodes.
- Added function `is.symmetric` to test whether a matrix is symmetric
- Added function `freeman` to calculate Freeman's segregation index

Version 0.0-8 (2007-04-26)

- Deleted the `.onLoad()` call as it was not really necessary and generated errors during compilation under R 2.5.0
- Corrected error in the search pattern of `sourceall()`

Version 0.0-7 (2007-01-26)

- Function `graphCoords` loads `sna` package quietly.
- With function `plotGraph` it is now possible to fill the vertices with colors (argument `v.bg` as in `points`).
- Corrected the package citation information.

Version 0.0-6 (2007-01-23)

- Added two functions for graph plotting `plotGraph` and `graphCoords`.
- Correction in DESCRIPTION file and NAMESPACE. Deleted some unnecessary dependencies. Package no longer automatically loads the `sna` package etc.

Version 0.0-5 (2006-11-19)

- Added function `toWord` for facilitating the work with MS Office programs.

Version 0.0-4 (2006-10-31)

- Added function `carryover` for recursive filling-in the gaps in vectors with last "meaningful" values.
- Added function `wcalc` for calculating column widths of fixed-width files
- Some minor corrections to DESCRIPTION and in documentation.

Version 0.0-3 (2006-10-26)

- Function `show.keywords` that lists the keywords database. Useful for writing Rd files.
- Added data `EcheniqueFryer` from Echenique and Fryer (2006)

- Added functions `ssi` and `ssib` implementing Spectral Segregation Index from Echenique and Fryer (2006)

Version 0.0-2 (2006-08-25)

- Added White's dataset.
- Some minor improvements to DESCRIPTION.

Version 0.0-1 (2006-02-19)

First release!

Author(s)

Michał Bojanowski <mbojan@ifispan.waw.pl> <http://bojan.3e.pl>

carryover

Carrying over values from previous elements

Description

This function recursively carries over values from previous to subsequent elements of a vector

Usage

```
carryover(x, empty = NA )
```

Arguments

<code>x</code>	numeric or character, vector to be processed
<code>empty</code>	numeric or character scalar, value to be replaced

Details

The function recursively imputes to all values of `x` that are equal to `empty` the values of the preceding elements of `x`.

If the first value of `x` is equal to `empty`, then an error is returned. For now you should take care of it by manually imputing the appropriate value.

Value

A vector `x` with all elements equal to `empty` replaced with last non-`empty` values.

See Also

[approx](#) for other ways of filling-in the gaps in vectors

Examples

```
x <- c(1, 2, 2, NA, NA, 1, 4)

# replacing NA's with last non-NA observation
carryover(x)
```

ccties	<i>Count the number of cross-class ties in the network</i>
--------	--

Description

Cross-classify network ties according to the type of tie “sender” and “receiver”.

Usage

```
ccties(g, b, th = 0, directed = FALSE, tab = TRUE)
```

Arguments

g	square matrix representing the network
b	vector of types
th	numeric scalar, all entries of g strictly greater than th are treated as representing a tie
directed	logical, should the network be interpreted as directed
tab	logical, type of result to be returned, see Details

Details

The function assumes that g is a squared matrix. The corresponding type vector b identifies class memberships of the vertices. It is assumed that it has the same number of elements as there are row/columns in g.

If matrix g is both symmetric and declared as undirected (directed=FALSE) than computations are performed on the lower triangle of g. In all other cases the full matrix is considered. This should increase the performance when dealing with undirected networks.

Network edges are identified by entries in g that are strictly greater than th, i.e. vertices i and j are linked in g if and only if $g[i, j] > th$.

If directed is FALSE than the function checks whether g is symmetric. If it is the case it is assumed to represent an undirected network. If the check fails an error is issued. Undirected networks are handled a bit more efficiently by considering a lower triangle matrix only.

The tab argument determines the type of result to be returned, see Value section.

Value

If TRUE a cross-tabulation of edges depending on the types of row and column vertices is returned. If tab is FALSE then the number of cross-type edges is returned.

See Also

[freeman](#) that relies on this function, [is.symmetric](#) for checking if the matrix is symmetric.

Examples

```
## let's use White's data, undirected network
data(White)
White

## using the 'tab' argument
# number of ties *between* men and women
ccties( White$network, White$gender, directed=FALSE, tab=FALSE)
# cross-classification of ties 1=women, 2=men
ccties( White$network, White$gender, directed=FALSE, tab=TRUE)

## the table can be used to draw meta-networks
# package 'sna' required
if(require(sna, quietly=TRUE))
{
  # create a graph for 80 nodes
  g <- rgws(1, 9, 2, 1, .3)
  # sample type vector of 4 types
  b <- sample( 1:4, 81, replace=TRUE )
  # plot the network
  plotGraph( g, v.col=b )
  # cross-classify the ties
  tab <- ccties( g, b, directed=FALSE, tab=TRUE)
  tab
  # build the meta-graph
  mg <- matrix( 0, ncol=4, nrow=4 )
  ro <- as.numeric(dimnames(tab)[[1]])
  co <- as.numeric(dimnames(tab)[[2]])
  mg[ro,col] <- tab
  # plot the meta network, edge width corresponds to the number of ties
  gplot( tab, edge.lwd=tab, vertex.col=sort(unique(b)), main="Meta network",
        displaylabels=TRUE, gmode="graph" )
}
```

coef.princomp

Component coefficients based on PCA

Description

Calculate the coefficients for components estimated in Principal Component Analysis with `princomp` function.

Usage

```
## S3 method for class 'princomp':
coef(object, ...)
```

Arguments

object	object of class <code>princomp</code>
...	other arguments currently not implemented

Details

Function calculates the coefficients with formula:

$$\lambda \% * \%diag(\sigma)$$

where λ is a matrix of “factor loadings” as resulting from applying function `loadings.princomp` and σ is a vector of diagonal matrix containing component’s standard deviations.

Value

Matrix containing component coefficients.

References

John Fox’s message on Rhelp from Thu Apr 10 10:55:18 2003

See Also

`princomp`, `coef`

Examples

```
pcmodel <- princomp( USArrests, cor=TRUE )
pcmodel
coef(pcmodel) # component coefficients
```

dePolish

Rip Polish letters of the diacritics

Description

The function substitutes all Polish letters in the provided character vector with ASCII counterparts.

Usage

```
dePolish(x, encoding=NULL)
```

Arguments

<code>x</code>	character vector, do be processed
<code>encoding</code>	character, encoding used in <code>x</code>

Details

The function substitutes (with `sub`) case-sensitively all Polish letters with simple ASCII counterparts.

The `encoding` argument defines the way in which Polish letters are encoded in `x`. If `NULL` and the platform is Windows the function uses CP1250 encoding. If not on Windows the function uses `iconv` to translate the letters to default encoding on current locale (via argument `from=""`).

Value

A character vector after substitutions

See Also

[sub](#), [iconv](#)

Examples

```
## Not run:
# some Polish family name
x <- "G\u015b\u0142\u0105\u015bka"
dePolish(x)
rm(x) # cleanup
## End(Not run)
```

dispplot

Location-dispersion plot

Description

Function for plotting location and dispersion parameters of one variable in subgroups defined by second variable.

Usage

```
dispplot(x, g, ...)
## S4 method for signature 'numeric':
dispplot(x, g, loc=mean, disp=sd, main=NULL, sub=NULL,
xlab=NULL, ylab=NULL)
```

Arguments

<code>x</code>	numeric vector representing “dependent” variable
<code>g</code>	vector treated as a grouping factor
<code>loc</code>	function to calculate location values
<code>disp</code>	function to calculate dispersion values
<code>main</code> , <code>sub</code> , <code>xlab</code> , <code>ylab</code>	arguments for labels as in plot
<code>...</code>	other arguments passed to other methods

Value

The plot is plotted, nothing is returned

See Also

[boxplot](#) for similar functionality, [mean](#), [sd](#)

Examples

```

data(iris)
attach(iris)

# mean and standard deviation by species
displot(Sepal.Length, as.numeric(Species))

# interquartile range
iqrange <- function(x, na.rm=TRUE)
  quantile(x, .75, na.rm=na.rm) - quantile(x, .25, na.rm=na.rm)

# medians and interquartile ranges
displot( Sepal.Width, as.numeric(Species), median, iqrange )

```

dynetSimple

*Simple visualization of dynamic network***Description**

Simple visualization of a dynamics of a network that has constant set of vertices.

Usage

```

dynetSimple(g, coords = NULL, layout = "fruchtermanreingold",
  layout.par = list(), ...)

```

Arguments

<code>g</code>	three-dimensional numeric array of network matrices, first dimension corresponds to network id
<code>coords</code>	NULL, numeric network position in the stack <code>g</code> or a numeric matrix with two columns and rows containig coordinates. See Details
<code>layout</code>	character, name of the vertex placement algorithm function
<code>layout.par</code>	list, optional list of arguments to placement function
<code>...</code>	other arguments passed to gplot

Details

This function is limited for networks with the same set of vertices. The plotting mechanism uses the recording functionality of the [windows](#) device. This is also a limitation as all devices share the `.SavedPlots` object where the plots are stored.

The main argument `g` is assumed to be a three-dimensional array containing a set of square adjacency matrices representing the networks. The first dimension enumerates the networks.

The `coords` argument may specify the coordinates of the vertices by a matrix with two columns (for `x` and `y`) and number of rows equal to the number of vertices of the network. Alternatively, the placement algorithm specified with `layout` is used to calculate them. If `coords` is NULL (default) the placement is calculated based on cumulated network, i.e. network obtained by summing up all graphs in the stack (`apply(g, c(2,3), sum)`). If it is a numeric scalar it is assumed to be an index of the graph in `g` to be used for getting the coordinates.

The `layout` is a name of the vertex placement algorithm found among those supplied in **sna** package. See `gplot.layout` for a list. Optional arguments to `layout` may be passed as a list through `layout.par`.

The function uses `gplot` function from package **sna** to do the plotting. Other arguments may be passed through `...`

Value

The vertex coordinates are returned invisibly. As a side effect a `windows` device is opened with the `record` option as `TRUE`. The plots are created and can be navigated with the device keys. Also, an object `.SavedPlots` storing the history is created in the global environment.

Note

As all `windows` devices share `.SavedPlots` object to store the history. This should be manually deleted after the device is closed. This directive could not be a part of the function as the object cannot be deleted if the device is still on.

See Also

`windows` and `recordPlot` for more information on the device and the recording functionality. `gplot` and `gplot.layout` for functions in **sna** package which this function uses.

Examples

```
# 'sna' package needed
if(require(sna, quietly=TRUE))
{
  # generate a stack of random graphs
  g <- rgraph(10, 9, tprob=.3)
  # plot using cumulated placement
  dynetSimple(g)
}
```

etas

Calculate Eta coefficients

Description

This generic function calculates Eta coefficients which are also known as “Correlation ratios” or (the squared value) as the “Proportion of explained variance”.

Usage

```
etas(object, ...)
## S4 method for signature 'numeric':
etas(object, fac)
## S4 method for signature 'anova':
etas(object)
## S4 method for signature 'lm':
etas(object)
```

Arguments

<code>object</code>	the R object
<code>fac</code>	vector for conditioning variable
<code>...</code>	arguments passed to other methods

Details

The Eta coefficient (more specifically its squared value) has a interpretation in terms of the proportion of explained variance.

In the decision theory the interpretation is related to the identification problem that involves two variables: y and x . The task is two identify the values of y .

The value of the η^2 is the proportion by which the error of predicting values of y is reduced by using the information contained in x .

Details of methods:

object is "numeric" For numeric vectors the function requires additional argument: a vector of the same length as the first.

The result is a value of the η^2 assuming that we want to predict the values of `object` with the values of `fac` using the so called "Type I regression of means".

For two variables y and x the η is given by the formula:

$$\eta^2 = (D^2(y) - E[D^2(y|x)]) / D^2(y)$$

object is "anova" or "lm" For objects of class `anova` the function calculates the Eta's and Partial Eta Squares for all effects in the given model. In this setting the eta squares for the given effect are equal to:

$$\frac{SS_{effect}}{SS_{total}}$$

where SS are appropriate Sums of Squares.

The "Partial Eta Squares" for the given effect are equal to:

$$\frac{SS_{effect}}{SS_{effect} + SS_{resid}}$$

Examples

```
x1 <- rnorm(50)
x2 <- rnorm(50)
y <- 5 + 2*x1 + rnorm(50,0,2) + 3*x2 + rnorm(50,0,.5)

# method for numeric
etas( y, rep(1:2, each=25) )

# method for 'lm' which calls 'anova'
m <- lm( y ~ x1 + x2 )
etas(m)
```

freeman

*Generalized Freeman's segregation index***Description**

Calculate generalized Freeman's segregation index for undirected networks with arbitrary number of groups.

Usage

```
freeman(g, b, bDist = NULL, verbose = FALSE)
```

Arguments

<code>g</code>	square binary matrix representing the network
<code>b</code>	vector of types of vertices
<code>bDist</code>	numeric, optional true distribution of types, see Details
<code>verbose</code>	logical, should some more output be provided

Details

Freeman's segregation index (Freeman, 1978) is designed to capture the extent to which the defined types of vertices tend to have more edges with vertices from the same type than with other types (segregation). Formally, the index compares the number of edges in the specified graph that connect vertices of different type to the number of "inter-type" edges that would be expected under pure random graph. High values indicate that there one would expect much more inter-type edges if they are to be created without considering the type of vertices, suggesting high segregation. Low values indicate that lower segregation. Especially, the index is 0 for the full graph (all possible edges are present).

The index has some discontinuity as there are graph and type configurations that are characterized by the higher number of inter-type edges that is expected under a random graph. The index truncates these situations and takes a value of 0.

The original Freeman's formulation was for only two types of vertices. Here it is extended to the arbitrary number of types. The modification only affects the way in which the expected number of inter-type edges under pure random graph is calculated.

About the function itself:

The matrix `g` is assumed to be binary, square, symmetric matrix representic an undirected graph. The vector `b`, either numeric or character, represent the types of vertices in graph `g`. Therefore its length must be equal to the number of rows/columns in `g`. It is assumed that rows/columns of the graph are in the same order as type membership in `b`.

The function internally calculates the frequency of types of vertices in the supplied vector `b`. However, it is possible to override this by specifying "true" type distribution with the `bDist` argument. It is assumed to be a table (as returned by `table`) or a numeric vector with frequencies of types of vertices. This may be especially usefull when dealing with large graphs with larger number of isolates.

If `verbose` is TRUE some intermediate results are printed (but not formally returned).

Value

The value of the Freeman's index.

References

Freeman, Linton C. (1978) Segregation in Social Networks, *Sociological Methods & Research* 6(4):411–429

See Also

Other segregation indices: [ssi](#) for Spectral Segregation Index. Additionally: [ccties](#) for cross-classification of edges by types of vertices on which this function is based.

Examples

```
## White's data from Freeman's article
data(White)
# plot the graph
plotGraph(White$network, v.col=White$gender)
# segregation level
freeman( White$network, White$gender )
# using 'verbose' argument
freeman( White$network, White$gender, verbose=TRUE )

# TODO example of using bDist
```

graphCoords

Calculate the coordinates of graph vertices

Description

Wrapper for algorithms calculating coordinates of graph vertices for plotting graphs.

Usage

```
graphCoords(m, alg = NULL, ...)
```

Arguments

<code>m</code>	square graph adjacency matrix
<code>alg</code>	character, name of function taking adjacency matrix as an argument, see Details
<code>...</code>	other arguments passed to <code>alg</code>

Details

Argument `m` is a square adjacency matrix representing a network.

If `alg` is `NULL` then the Fruchterman-Reingold placement algorithm from package `sna` is used, see [gplot.layout.fruchtermanreingold](#).

If package `sna` is not available then random placement through [placementRandom](#) is applied. Otherwise it accepts a string which is a name of a function looked-up in the global environment. This function is assumed to take an adjacency matrix as a first argument and return a two-column matrix of vertex coordinates.

Through `...` other arguments may be passed to function named with `alg`.

Value

A two-column matrix containing coordinates of vertices produced with the placement algorithm supplied.

See Also

[placementRandom](#) for a simple random placement algorithm supplied with this package. See [gplot.layout](#) for algorithms supplied in **sna** package.

Also [plotGraph](#) that relies on [graphCoords](#) and [gplot](#) in package **sna** for high-level function for drawing graphs.

Examples

```
# some random graph
x <- sample( 0:1, 25, replace=TRUE )
m <- matrix(x, ncol=5 )

if(require(sna, quietly=TRUE))
{
  # F-R placement
  graphCoords(m, layout.par=list())
}

# built-in random placement
graphCoords(m, alg="placementRandom")
```

iCurveFit

Interactive Curve Fitting

Description

Interactively enter a set of points and then fit a curve of specified form to get a coefficients.

Usage

```
iCurveFit(formula = y ~ x, n = 5, model = "lm",
  prn = 100, xlim = c(0, 10), ylim = c(0, 10), ...)
```

Arguments

formula	formula for the model to be fitted to the entered data
n	numeric, number of data points to be entered
model	character, name of the model-fitting function to be used for the entered data
prn	numeric, number of points used to evaluate the fitted model, important only for plotting the fitted curve
xlim	numeric of length 2, limits for the horizontal axis
ylim	numeric of length 2, limits for the vertical axis
...	other arguments passed to the model-fitting function

Details

Frequently there is a need to quickly come up with a particular form of a function. This function enables to quickly fit a curve of certain kind to a, usually small, number of data points.

The function starts with creating an empty plotting region ready for input of n data points. Every data input (through `locator`) results in adding a point to the plot with a subsequent number. After the data are inputted the function uses model-fitting function specified with `model` to fit the curve specified with `formula` with possibly additional arguments passed through `...`

The `formula` argument has a usual form as in almost all model-fitting functions in R. It can contain only variables y and x . Furthermore, direct transformations of y are not supported. All transformations of x should be enclosed in `I()` function.

The function was designed to be used with `lm` or `glm` as model-fitting functions. However, in principle, it should be possible to use any other function that provides similar interface, i.e. `formula` and data arguments, and the corresponding `predict` method.

Once the model is fitted the function finds its predicted values for a sequence of values defined by `xlim` of length `prn`. Then the curve itself is added to the plot. Finally the model object is returned.

Value

A model object created for the supplied data with the model-fitting function specified with `model`.

See Also

See [lowess](#) and [spline](#) for other curve fitting functions, [lm](#) and [glm](#) for help with the model-fitting functions supported. Also see [predict](#) and [I](#)

Examples

```
## Not run:
# fitting via 'lm'
iCurveFit(formula= y ~ x ) # linear
iCurveFit(formula= y ~ I(x^2) + x ) # parabolic
iCurveFit(formula= y ~ I(x^3) + I(x^2) + x, n=7) # polynomial of degree 3
iCurveFit(formula= y ~ I(sin(x)), n=7) # sinusoidal
iCurveFit(formula= y ~ I(log(x)), n=7) # log
iCurveFit(formula= y ~ I(1/x), n=7) # hyperbolic
## End(Not run)
```

is.symmetric

Test for a symmetric matrix

Description

Test whether the supplied matrix is symmetric around diagonal

Usage

```
is.symmetric(m)
```

Arguments

`m` a matrix to be tested

Details

If the matrix `m` is not a square matrix the function returns `FALSE` and issues a warning.

The testing boils down to comparing lower and upper triangles of the matrix `m` using the function `identical`.

Value

Logical, whether the matrix is symmetric or not.

See Also

See `identical`, `matrix`, `upper.tri`. Also `symmetrize` in package `sna`.

Examples

```
# create some matrix
m <- matrix( c(0, 1, 1, 0), ncol=2 )
m
is.symmetric(m) # TRUE

m1 <- m
m1[2,1] <- 0
m1
is.symmetric(m1) # FALSE

# supplying a non-square matrix
m2 <- matrix( 1:6, ncol=2 )
is.symmetric(m2) # FALSE with warning

# clean-up
rm(m, m1, m2)
```

isGraphical

Test if a degree sequence isGraphical

Description

Given a vector of integer values check whether it is “graphical”, i.e. whether there exists a graph (undirected) with a corresponding sequence of degrees.

Usage

```
isGraphical(x, method = c("tv", "eg"))
```

Arguments

`x` integer vector, degree sequence to be tested
`method` character, type of algorithm to be used, defaults to "tv", see Details

Details

A sequence of non-negative integers a_1, a_2, \dots, a_p . is *graphical* if there exists a graph with the corresponding degree sequence.

This function takes such a sequence as an argument `x` and performs the check.

Two algorithms are implemented. The first one due too Erdos and Gallai (1960), the second, a bit more efficient and the default, is due too Tripathi and Vijay (2003). The method can be selected with the `method` argument.

Value

Logical, whether the sequence is graphical or not.

References

P. Erdos, T. Gallai (1960) Graphs with prescribed degree of vertices (Hungarian), Mat. Lapok 11 264–274.

A. Tripathi and Sujith Vijay (2003) A note on a theorem of Erdos and Gallai, Discrete Mathematics 265 417–420

Examples

```
### Test both methods on some simple graphs
testit <- function(d)
{
  rval <- c( eg = isGraphical(d, "eg"),
            tv = isGraphical(d, "tv") )
  if(!all(rval))
  {
    print(rval)
    stop("failed")
  }
  rval
}
# graphs of size 3
testit( c(0,0,0) )
testit( c(0,1,1) )
testit( c(1,2,1) )
testit( c(2,2,2) )

### Compare speeds for random graphs of sizes from 20 to 200
### Package 'sna' required
if(require(sna))
{
  n <- 20
  s <- seq( 20, 500, by=20 )
  r <- matrix(NA, nrow=length(s), ncol=2)
  for( i in seq(along=s) )
  {
    g <- rgraph(s[i])
    d <- degree(g) / 2
    library(bojan)
    o <- speedcomp( e1=expression(isGraphical(d, "eg")),
                   e2=expression(isGraphical(d, "tv")) )
  }
}
```

```

        r[i,] <- o@timings[, "elapsed"]
    }
    plot( range(s), range(r), type="n", main="Speed comparison of E-G and T-V",
          xlab="Network size", ylab="Time [10ms]")
    lines(s, r[,1], pch="EG", type="b", cex=.6, col="red")
    lines(s, r[,2], pch="TV", type="b", cex=.6)
    # E-G a little-bit slower
    summary( r[1,] - r[2,] )
}

```

lss

List object names along with classes and sizes

Description

List, and optionally plot, all the objects in the specified environment along with their class attribute and size in bytes.

Usage

```
lss(graph = FALSE, env = .GlobalEnv, ...)
```

Arguments

<code>graph</code>	logical, should a dotchart of sizes be produced
<code>env</code>	environment to scan for objects, defaults to Global Environment
<code>...</code>	other arguments passed to <code>ls</code>

Details

The function scans the environment supplied by `env` with the `ls` function for objects. Then the classes of these objects are retrieved with `object.size`.

If `graph` is `TRUE` then additionally a dotchart with showing object sizes is produced.

Value

A data frame with two columns “class” and “size” containing the class and size (in bytes) of the objects found in the environment `env`.

See Also

`ls` for listing the objects, `object.size` for calculating the memory size of an object, `dotchart` for making dotcharts

Examples

```
# make some environment
e <- new.env()
# create some objects in it
assign("x", rnorm(10), envir=e)
assign("d", data.frame(a=1:10, b=10:1), envir=e)
assign("l", letters, envir=e)

# table of contents
lss(env=e)

# with a chart
lss(TRUE, env=e)

# clean-up
rm(e)
```

ncinfo-deprecated *Summary information about NetCDF file*

Description

This generic function returns a summary information about a NetCDF file like the list of dimensions, defined variables etc.

Usage

```
ncinfo(object)
```

Arguments

`object` object of class "RNetCDF", or a name of the file

Details

This function relies on package **RNetCDF**.

Currently there are two methods supported.

The `object` can be a file handle (object of class "RNetCDF") or a file name. See examples.

Value

A list of components `dimensions` and `variables`:

`dimensions` a matrix with columns "id", "name", "length", and "unlim"; and rows for all the dimensions defined in the file.

`variables` a matrix with columns "id", "name", "type", "ndims", "dimids", and "natts" and rows for all variables defined in the file.

Note

This function will be moved to package **mbtools**.

See Also

Accessing NetCDF files with **RNetCDF**

Examples

```
## This is based on examples from the help page on 'var.put.nc' from
## package RNetCDF
if(require(RNetCDF))
{
  # temporary file
  tfile <- tempfile()

  nc <- create.nc(tfile)

  # define dimensions
  dim.def.nc(nc, "station", 5)
  dim.def.nc(nc, "time", unlim=TRUE)
  dim.def.nc(nc, "max_string_length", 32)

  # Create three variables, one as coordinate variable
  var.def.nc(nc, "time", "NC_INT", "time")
  var.def.nc(nc, "temperature", "NC_DOUBLE", c(0,1))
  var.def.nc(nc, "name", "NC_CHAR", c("max_string_length", "station"))

  # Put some missing_value attribute for temperature
  att.put.nc(nc, "temperature", "missing_value", "NC_DOUBLE", -99999.9)

  # Define variable values
  mytime <- c(1:2)
  mytemperature <- c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, NA, NA, 9.9)
  myname <- c("alfa", "bravo", "charlie", "delta", "echo")
  dim(mytemperature) <- c(5,2)

  # Put the data with indicated start/count
  var.put.nc(nc, "time", mytime, 1, length(mytime))
  var.put.nc(nc, "temperature", mytemperature, c(1,1), c(5,2))
  var.put.nc(nc, "name", myname, c(1,1), c(32,5))

  sync.nc(nc)

  # Put the data with default start/count
  var.put.nc(nc, "time", mytime)
  var.put.nc(nc, "temperature", mytemperature)
  var.put.nc(nc, "name", myname)

  # close the file
  close.nc(nc)
  rm(nc)

  ### Now use the 'ncinfo' function to get the information
  # from netcdf handle object
  nc <- open.nc(tfile)
  ncinfo(nc)
  close.nc(nc)
  rm(nc)
}
```

```

# directly from file
ncinfo(tfile)

### Clean-up
unlink(tfile)
rm(tfile)
}

```

nullModel

Fit two-level null random intercept model

Description

Fit a null two-level random intercept model using closed-form formulas provided by Snijders and Bosker (1999)

Usage

```
nullModel(y, g, data = NULL)
```

Arguments

y	numeric, dependent variable
g	numeric, grouping factor
data	data frame, optional dataset where the variables should be looked for

Details

The function fits a random effect model of the form:

$$Y_{ij} = \mu + U_j + R_{ij}$$

where Y_{ij} is the value of the dependent variable (y) for i -th object within j -th group, μ is the population mean, U_j is the effect specific to group j , and R_{ij} is the within-group error.

The function estimates between-group and pooled within-group variances as well as the intra-class correlation coefficient. The closed-form expressions are provided by Snijders and Bosker (1999, p. 18–21).

Value

The function returns an object of S4 class `nullModel` which contains the following slots:

```

this-is-escaped-codenormal-bracket32bracket-normal
  character, name of the dependent variable
this-is-escaped-codenormal-bracket35bracket-normal
  character, name of the grouping factor
this-is-escaped-codenormal-bracket38bracket-normal
  numeric, number of groups
this-is-escaped-codenormal-bracket41bracket-normal
  numeric, total number of observations
this-is-escaped-codenormal-bracket44bracket-normal
  numeric, pooled within-group variance

```

```

this-is-escaped-codenormal-bracket47bracket-normal
    numeric, between-group variance
this-is-escaped-codenormal-bracket50bracket-normal
    numeric, estimate of the population between-group variance
this-is-escaped-codenormal-bracket53bracket-normal
    numeric, intra-class correlation coefficient

```

References

Snijders, T.A.B., Bosker, R. (1999) "Multilevel Analysis. An Introduction to Basic and Advanced Multilevel Modelling". London: Sage

See Also

[lmer](#) in package **lme4** for general fitting of mixed-effects models.

Examples

```

if(require(lme4))
{
  # fit null model with lmer
  library(lme4)
  system.time(mm <- lmer( Reaction ~ 1 | Subject, sleepstudy ))
  # fit with closed-form formulas
  system.time(m <- nullModel( sleepstudy$Reaction, sleepstudy$Subject ))
  mm
  m
}

```

panel.model

Panel plot with model-based curve

Description

A “panel” function for use with [coplot](#) and [pairs](#) functions. The function plots a cloud of points together with a line based on fitted values from the specified model.

Usage

```

panel.model(x, y, mfun = "lm", mformula = formula(y ~ x),
  col = par("col"), pch = par("pch"), cex = 1,
  col.model = "red", bg = NA, ...)

```

Arguments

<code>x</code> , <code>y</code>	numeric vectors of the same length
<code>mfun</code>	character, name of the model-fitting function, defaults to "lm"
<code>mformula</code>	formula with variables <code>x</code> and <code>y</code> only, specification of the model to be fitted in each panel
<code>col</code> , <code>pch</code> , <code>cex</code> , <code>bg</code>	graphical parameters passed to <code>points</code>
<code>col.model</code>	color specification for the line
<code>...</code>	other arguments passed to <code>lines</code>

See Also

See [panel.smooth](#) for another example of panel function, also [coplot](#) and [pairs](#) for the context in which this function can be used.

Examples

```
# create some data
d <- data.frame( y = rnorm(100),
  x = rnorm(100),
  f = factor(sample(1:6, 100, replace=TRUE)) )
# fit straight line (these are defaults)
coplot( y ~ x | f, data=d, col="gray", col.model="black",
  panel=panel.model)
# other fittings through 'lm'
# parabola
coplot( y ~ x | f, data=d, col="gray", col.model="black",
  panel=panel.model, mformula=formula( y ~ x + I(x^2)) )
# exponential
coplot( y ~ x | f, data=d, col="gray", col.model="black",
  panel=panel.model, mformula=formula( y ~ x + exp(x) ) )
# sinus
coplot( y ~ x | f, data=d, col="gray", col.model="black",
  panel=panel.model, mformula=formula( y ~ x + sin(x)))
```

placement

Some graph vertex placement algorithms

Description

A collection of graph placement algorithms

Usage

```
placementRandom(m, xlim = c(-1, 1), ylim = c(-1, 1))
```

Arguments

m	square numeric graph adjacency matrix
xlim	limits for the x-axis
ylim	limits for the y-axis

Details

placementRandom is a simple random placement algorithm that allocates the points ranomly within the limits specified by xlim and ylim.

Value

A matrix with two columns containing the vertex coordinates.

plotGraph

Basic plotting of undirected graphs

Description

This function is a simple routine for plotting undirected graphs.

Usage

```
plotGraph(m, coords, ...)
## S4 method for signature 'matrix, matrix':
plotGraph(m, coords, thres=0, pch=19, v.col="red", v.cex=1,
v.bg=NULL, e.lwd=1, e.col="black", e.lty=1, v.lab=NULL)
## S4 method for signature 'matrix, character':
plotGraph(m, coords, thres=0, pch=19, v.col="red", v.cex=1,
v.bg=NULL, e.lwd=1, e.col="black", e.lty=1, v.lab=NULL, ... )
## S4 method for signature 'matrix, missing':
plotGraph(m, coords, thres=0, pch=19, v.col="red", v.cex=1,
v.bg=NULL, e.lwd=1, e.col="black", e.lty=1, v.lab=NULL, ... )
```

Arguments

<code>m</code>	numeric graph adjacency matrix
<code>coords</code>	two-column matrix containing vertex coordinates
<code>thres</code>	values strictly greater than <code>thres</code> are treated as 1
<code>pch</code>	symbol used for vertex plotting, as in points
<code>v.col</code>	color used for plotting vertices
<code>v.bg</code>	color used for filling vertex symbols
<code>v.cex</code>	expansion factor for vertices
<code>e.lwd</code>	line width of graph edges
<code>e.col</code>	line color of graph edges
<code>e.lty</code>	line type of graph edges
<code>v.lab</code>	character, vertex labels
<code>...</code>	other arguments passed to the vertex placement function

Details

The adjacency matrix `m` is treated as undirected, therefore if either `m[i, j]` or `m[j, i]` is strictly greater than `thres` it is assumed that the network link exists.

The placement of the vertices is resolved based on `coords` and `alg` arguments. If `coords` is not `NULL` it is assumed to be a matrix with two columns and one row for every vertex in `m`. Columns correspond to *x* and *y* coordinates of the vertices. If `coords` is `NULL` then `alg` is taken as a name of the function of the vertex placement algorithm.

Value

A plot of a graph is created and the vertex coordinates are returned invisibly.

See Also

Function `gplot` in package `sna` for similar functionality, `graphCoords` for a wrapper for vertex placements algorithms.

Examples

```
x <- sample( 0:1, 100, replace=TRUE, prob=c(.7, .3))
g <- matrix(x, ncol=10)
plotGraph(g)
```

rescale

*Rescale linearly to new minimum and maximum***Description**

Linearly rescale values of a numeric vector so that the result have new prescribed minimum and maximum.

Usage

```
rescale(x, nmin = 0, nmax = 1, na.rm = FALSE, coef=FALSE)
```

Arguments

<code>x</code>	numeric vector to rescale
<code>nmin</code>	numeric scalar, new minimum, defaults to 0
<code>nmax</code>	numeric scalar, new maximum, defaults to 1
<code>na.rm</code>	logical, should the missing values be removed, defaults to FALSE
<code>coef</code>	logical, should a list with coefficients of the transformation be returned instead

Details

The function linearly rescales the values of `x` so that the minimum and maximum of the result are equal to the values supplied with `nmin` and `nmax`.

The formula for the transformation is $a + bx$ where the coefficients of the transformation are given by:

$$a = nmin - \frac{\min(x)(nmax - nmin)}{\max(x) - \min(x)}$$

and

$$b = \frac{nmax - nmin}{\max(x) - \min(x)}$$

If `x` contains missing data (NAs) and `na.rm` is not TRUE, the function will return an error.

If `coef` is TRUE then instead of transformed values of `x` the coefficients a and b are returned in a list.

Value

If `coef` is FALSE, which is the default, a vector of transformed values of `x` is returned. If it is TRUE then the list with components `a` and `b` is returned containing the coefficients of the transformation.

See Also[scale](#)**Examples**

```
# simple functionality
x <- rnorm(20)
y <- rescale(x)
summary(x)
summary(y)
# coefficients of the transformation
k <- rescale(x, coef=TRUE)
k

# show how it works on the plot
plot(x, y, xlab="x", ylab="Result", main="Mechanics of 'rescale()'",
      axes=TRUE, asp=1, sub=paste("a=", k$a, ", b=", k$b, sep=""))
abline(v=c(min(x), max(x)), lty=2, col="gray")
abline(h=c(min(y), max(y)), lty=2, col="gray")
abline(a=k$a, b=k$b, col="gray")
text((max(x) + min(x))/2, (max(y) + min(y))/2,
      label="a + bx", srt=tan(k$b) / pi * 180, pos=3)
```

see-deprecated

*Inspect the content of an object***Description**

Inspect the content of an object, usually in an external editor. Methods are two implemented for functions and data frames.

Usage

```
see(o, ...)
## S4 method for signature 'ANY':
see(o, ...)
## S4 method for signature 'data.frame':
see(o, ...)
```

Arguments

- o object to be inspected
- ... arguments passed to other methods

Details

For data frames the function uses [edit](#). The result is returned invisibly.

For any other objects R code is generated via [dump](#) to a temporary file and the file is opened in an external editor. The editor is taken from `getOption("editor")`.

In case of the data frames the R console waits until the file is closed.

Value

Either NULL or the result invisibly.

Note

This function will be moved to package **mbtools**.

See Also

[edit](#)

Examples

```
## Not run:  
# for function  
see(lm)  
# for data.frame  
see( data.frame(x=1:5, y=5:1) )  
## End(Not run)
```

```
show.keywords-deprecated  
      Show the Keywords list
```

Description

This function will show the Keywords list needed for writing R Documentation files in Rd format.

Usage

```
show.keywords(file = NULL)
```

Arguments

`file` path to file with keywords, defaults to 'RHOME/doc/KEYWORDS'

Details

The function list the `file`. By default it is taken to be a 'KEYWORDS' file in 'RHOME/doc/KEYWORDS'. The value of the RHOME variable is taken by `Sys.getenv` function.

Value

Nothing, i.e. NULL.

See Also

[file.show](#), [Sys.getenv](#)

Examples

```
show.keywords()
```

sourceall-deprecated

Source all R-code files that match the pattern

Description

This function sources the files in a specified directory that follow the supplied pattern.

Usage

```
sourceall(dir = getwd(), pattern = ".*\\.R", ...)
```

Arguments

<code>dir</code>	character, the directory to look for the files. Defaults to the working directory
<code>pattern</code>	character, the regular expression that defines the pattern of the files to be sourced. Defaults to all files with ".R" extension.
<code>...</code>	other arguments passed to <code>source()</code>

Details

All files in the `dir` directory are sourced in alphabetical order via [source](#).

Note

This function will be moved to package **mbtools**.

See Also

[source](#)

speedcomp-deprecated

Comparing CPU times of several R expressions

Description

This function enables comparing CPU times of several R expressions. The times are calculated via [system.time](#), additionally time ratios are calculated.

Usage

```
speedcomp(...)
```

Arguments

`...` a collection of R expressions in the form of tag=value

Details

The function uses `system.time` to estimate CPU times needed to evaluate each of the expressions. The output provides raw timings as well as ratios of time elapsed.

Value

An object of S4 class `speedcomp` with two slots

`e` a list of R expressions processed
`timings` a matrix containing the CPU times of the expressions in `e` estimated with `system.time`. Rows correspond to expressions and columns to the fields returned by `system.time`, i.e. "user cpu", "system cpu", "elapsed", "subproc1" and "subproc2".

The included `show` method prints the results nicely and additionally calculates the ratios of times elapsed.

Note

This function will be moved to package **mbtools**.

See Also

`system.time`

Examples

```
# some testing of 'lm' fitting with different data sizes
e1 <- expression( lm( I(rnorm(10000)) ~ I(runif(10000)) ) )
e2 <- expression( lm( I(runif(20000)) ~ I(rnorm(20000)) ) )

# compare
speedcomp(e1, e2)
```

 ssi

Spectral Segregation Index for Social Networks

Description

These functions implement Spectral Segregation Index as proposed by Echenique & Fryer (2006). This index is an individual-level measure of segregation in a given network.

Usage

```
ssi(n, v)
ssib(n, v, b)
```

Arguments

`n` numeric matrix representing a network, should be *normalized*, see Details
`v` numeric vector, representing vertex types
`b` numeric scalar, for which type the segregation indices should be calculated

Details

For a full description and axiomatization see Echenique & Fryer (2006). The function `ssi` essentially applies `ssib` to all groups defined by `v`.

The network `n` should be normalized so that all rows sum-up to 1. As networks are usually represented as binary matrices such matrix may be normalized using `sweep` function, see Examples below.

The procedure essentially consists of creating a submatrix, say, B of the supplied matrix `n`. This submatrix B contains only vertices of the given type. It may be viewed as a type-homogeneous subnetwork of `n`. This subnetwork is further decomposed into connected components. Then, for every component, an eigenvalue decomposition is applied. The value of the index for the component is simply the largest eigenvalue, and the individual-level indices are obtained by distributing it according to the corresponding eigenvector.

Value

Both functions return a `data.frame` with the following columns:

<code>ind</code>	actor number, a row/column index of the <code>n</code> matrix
<code>group</code>	group id as supplied by <code>v</code> and <code>b</code>
<code>cmem</code>	a component id <i>within</i> each group
<code>cssi</code>	value of the SSI index on the component level
<code>ssi</code>	value of the SSI index on the individual level

References

Echenique, Federico and Roland G. Fryer, Jr. (2006) A Measure of Segregation Based On Social Interactions

See Also

Data from E&F [EcheniqueFryer](#).

Examples

```
# load the example data from E&F
data( EcheniqueFryer )

# SSI indices for "blacks"
# ssib( EcheniqueFryer$network, EcheniqueFryer$type, 1 )

# SSI indices for everybody
ssi(EcheniqueFryer$network, EcheniqueFryer$type)
```

subst-deprecated *Perform a value substitution within an object*

Description

This is a generic function that perform substitutions of values within an object. Methods are available for vectors and lists.

Usage

```
subst(object, a, b)
## S3 method for class 'list':
subst(object, a, b=NA)
```

Arguments

object	an object containing the values to be substituted, currently vector or list
a	value to be substituted
b	value to substitute a's with, defaults to NA
...	other arguments to methods

Details

For vectors this simply substitutes all occurrences of a's with b's. By default all a's are "recoded" into NA's.

For lists the function performs a substitution for each component separately (via `lapply`).

Value

A vector or list (depending on the class of input object) after substitutions.

See Also

[recode](#) in package `car`

Examples

```
x <- c(1,2,1,3,1,4) # Some vector
x
subst( x, 1 ) # NA instead of 1's

# Substituting strings
LETTERS[x]
subst( LETTERS[x], "A", "was A" )
```

Description

This function exports R objects into Windows clipboard in a format that makes the further work in MS Office easier. Current methods cover matrices and data frames.

Usage

```
toWord(object, ...)
## S4 method for signature 'matrix':
toWord(object, useNames=TRUE)
## S4 method for signature 'data.frame':
toWord(object, useNames=TRUE)
tcb(object, ...)
```

Arguments

object	object to processed, currently matrix, data frame or table are supported directly. Other objects are coerced to matrix.
useNames	logical, whether to use dimension names
...	other arguments to methods

Details

There are currently three methods: for matrices, tables, and data frames. Data frames and tables are converted to matrices. `tcb` is an alias for `toWord`.

For any other objects the function tries to coerce its argument to matrix.

If `useNames` is `TRUE` then the object is expanded to contain row and column names. The first cell in the first row contain dimension names separated with a backslash.

This function converts the object to a tab-separated table and puts it in the Windows clipboard. The content can then be pasted in to MS Word or MS Excel. For making tables in MS Word you will have to first create an empty table with appropriate number of rows and columns. To aid you in this after the `toWord` is called it displays an informative message about the dimensionality of the exported table. As a next step you have to select the whole table in MS Word, preferably via menus: `Table|Select|Table`. Then paste in the content of the clipboard.

In MS Excel you are not required to select the appropriate area of the spread sheet. It is sufficient to put the cursor (cell selector) in the upper-left corner of the to-be-table.

Value

The converted object is put to clipboard and returned invisibly.

Note

This function will work only on MS Windows distributions of R

See Also

[writeClipboard](#) and packages `xtable` and my other package `texex` for exporting objects to TeX and HTML

Examples

```
# 'showClip' function for testing only:
# print the contents of the clipboard to the console
showClip <- function()
{
  d <- readClipboard( format=1, raw=FALSE )
  cat(d, sep="\n")
  cat("\n")
}

# some matrix
m <- matrix(1:4, ncol=2, dimnames=list(r=1:2, c=1:2))
m

### Matrix with names
toWord(m)
showClip()
# clipboard contains:
# r\c  1      2
# 1    1      3
# 2    2      4

### Matrix without names
toWord(m, useNames=FALSE)
showClip()
# 1      3
# 2      4

### Data frame
DF <- as.data.frame(m)
DF
toWord(DF)
showClip()
#      V1      V2
# 1      1      3
# 2      2      4

### Table
# one-dimensional horizontally
toWord( table(1:5) )
showClip()
# two-dimensional as matrix
toWord( table(1:5, 1:5) )
showClip()
# higher dimensions should return error
r <- try( toWord( table(1:5, 1:5, 1:5) ), silent=TRUE )
cat(r)
stopifnot(inherits(r, "try-error"))

# 'tcb' is an alias for 'toWord'
tcb(m)
```

tcb (DF)

wcalc

Calculate the column widths of a fixed-width file

Description

Calculate column widths of a plain text fixed-width file from column numbers on which each field begins.

Usage

wcalc (p)

Arguments

p numeric vector of column numbers, should start with 1 and end with the number of the last column in a file+1

Details

The first element of the p argument should be 1 as it is the first column of the first field. Subsequent numbers should correspond to the column numbers that begin other fields. The last number of p should be the end of the last field + 1 or, equivalently, the beginning of the next (non-existing) field.

Value

Numeric vector of column widths.

See Also

[readThomsonFwf](#)

Examples

```
# TODO add examples
```

Index

- *Topic **IO**
 - toWord, 38
 - *Topic **aplot**
 - panel.model, 28
 - *Topic **arith**
 - rescale, 31
 - *Topic **array**
 - ccties, 10
 - is.symmetric, 21
 - *Topic **character**
 - dePolish, 13
 - *Topic **datasets**
 - EcheniqueFryer, 1
 - Inflacja, 2
 - White, 3
 - *Topic **dplot**
 - graphCoords, 19
 - placement, 29
 - *Topic **environment**
 - lss, 23
 - *Topic **file**
 - ncinfo-deprecated, 25
 - show.keywords-deprecated, 33
 - sourceall-deprecated, 34
 - *Topic **graphs**
 - ccties, 10
 - dynetSimple, 15
 - freeman, 17
 - isGraphical, 22
 - *Topic **hplot**
 - dispplot, 14
 - dynetSimple, 15
 - plotGraph, 29
 - *Topic **iplot**
 - iCurveFit, 20
 - *Topic **manip**
 - carryover, 9
 - is.symmetric, 21
 - subst-deprecated, 37
 - wcalc, 40
 - *Topic **methods**
 - dispplot, 14
 - etas, 16
 - plotGraph, 29
 - toWord, 38
 - *Topic **misc**
 - bojan-deprecated, 5
 - *Topic **multivariate**
 - anova.princomp, 4
 - coef.princomp, 12
 - *Topic **package**
 - bojan-package, 6
 - *Topic **programming**
 - sourceall-deprecated, 34
 - speedcomp-deprecated, 34
 - *Topic **regression**
 - etas, 16
 - *Topic **smooth**
 - iCurveFit, 20
 - *Topic **univar**
 - ssi, 35
 - *Topic **utilities**
 - see-deprecated, 32
 - show.keywords-deprecated, 33
 - wcalc, 40
- anova, 5
anova.princomp, 4
approx, 10

base-defunct, 6
bojan (*bojan-package*), 6
bojan-deprecated, 5
bojan-package, 6
boxplot, 14

carryover, 9, 9
ccties, 10, 18
coef, 12
coef.princomp, 12
coplot, 7, 28

dePolish, 8, 13
Deprecated, 6
dispplot, 14
dispplot, numeric-method
(*dispplot*), 14

- dispplot-methods (*dispplot*), 14
- dump, 32
- dynetSimple, 8, 15
- EcheniqueFryer, 1, 36
- edit, 7, 32, 33
- etas, 16
- etas, anova-method (*etas*), 16
- etas, lm-method (*etas*), 16
- etas, numeric-method (*etas*), 16
- etas-methods (*etas*), 16
- file.show, 7, 33
- freeman, 8, 11, 17
- get, 8
- glm, 21
- gplot, 15, 16, 19, 30
- gplot.layout, 15, 16, 19
- gplot.layout.fruchtermanreingold, 19
- graphCoords, 8, 9, 19, 30
- I, 21
- iconv, 13
- iCurveFit, 8, 20
- identical, 22
- Inflacja, 2
- is.symmetric, 11, 21
- isGraphical, 7, 22
- lm, 21
- lmer, 27
- loadings.princomp, 12
- lowess, 21
- ls, 8
- lss, 8, 23
- matrix, 22
- mean, 14
- ncinfo, 7
- ncinfo (*bojan-deprecated*), 5
- ncinfo, ANY-method (*bojan-deprecated*), 5
- ncinfo, character-method (*bojan-deprecated*), 5
- ncinfo-deprecated, 25
- ncinfo-methods (*bojan-deprecated*), 5
- nullModel, 7, 26
- nullModel-class (*nullModel*), 26
- pairs, 7, 28
- panel.model, 7, 28
- panel.smooth, 28
- placement, 29
- placementRandom, 19
- placementRandom (*placement*), 29
- plot, 14
- plotGraph, 8, 9, 19, 29
- plotGraph, matrix, character-method (*plotGraph*), 29
- plotGraph, matrix, matrix-method (*plotGraph*), 29
- plotGraph, matrix, missing-method (*plotGraph*), 29
- plotGraph-methods (*plotGraph*), 29
- points, 30
- predict, 21
- princomp, 5, 12
- readThomsonFwf, 40
- recode, 37
- recordPlot, 16
- replace, 6
- rescale, 8, 31
- scale, 31
- sd, 14
- see, 7, 8
- see (*bojan-deprecated*), 5
- see, ANY-method (*bojan-deprecated*), 5
- see, data.frame-method (*bojan-deprecated*), 5
- see-deprecated, 32
- see-methods (*bojan-deprecated*), 5
- show, 35
- show, nullModel-method (*nullModel*), 26
- show, speedcomp-method (*bojan-deprecated*), 5
- show.keywords, 7, 8
- show.keywords (*bojan-deprecated*), 5
- show.keywords-deprecated, 33
- source, 34
- sourceall, 7
- sourceall (*bojan-deprecated*), 5
- sourceall-deprecated, 34
- speedcomp, 7, 8
- speedcomp (*bojan-deprecated*), 5
- speedcomp-class (*bojan-deprecated*), 5
- speedcomp-deprecated, 34
- spline, 21

ssi, 18, 35
ssib(ssi), 35
sub, 13
subst(bojan-deprecated), 5
subst-deprecated, 37
symmetrize, 22
Sys.getenv, 33
system.time, 34, 35

tcb, 7
tcb(toWord), 38
toWord, 7, 9, 38
toWord, ANY-method(toWord), 38
toWord, data.frame-method
 (toWord), 38
toWord, matrix-method(toWord), 38
toWord, table-method(toWord), 38
toWord-methods(toWord), 38

upper.tri, 22

wcalc, 9, 40
White, 3
windows, 15, 16
writeClipboard, 39