Package ‘broman’

April 11, 2019

Version 0.69-5
Date 2019-04-11
Title Karl Broman's R Code
Description Miscellaneous R functions, including functions related to
graphics (mostly for base graphics), permutation tests, running
mean/median, and general utilities.
Author Karl W Broman <broman@wisc.edu>, with contributions from Aimee
Teo Broman
Maintainer Karl W Broman <broman@wisc.edu>
Depends R (>= 2.15.0)
Imports utils, graphics, grDevices, stats, assertthat, RPushbullet,
jsonlite, ggplot2, grid
Suggests testthat, devtools, roxygen2
License GPL-3
URL https://github.com/kbroman/broman
BugReports https://github.com/kbroman/broman/issues
Encoding UTF-8
ByteCompile true
LazyData true
RoxygenNote 6.1.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-04-11 16:22:39 UTC

R topics documented:

<table>
<thead>
<tr>
<th>add_commas</th>
<th>..........................................................</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>align_vectors</td>
<td>...........................................................................</td>
<td>4</td>
</tr>
<tr>
<td>arrowlocator</td>
<td>...........................................................................</td>
<td>4</td>
</tr>
</tbody>
</table>
R topics documented:

- attrnames
- brocolors
- bromanversion
- cf
- chisq
- ciplot
- colwalpha
- compare_rows
- convert2hex
- crayons
- done
- dotplot
- errors2pushbullet
- excel_fig
- exit
- fac2num
- fisher
- get_precision
- grayplot
- grayplot_na
- h
- hex2dec
- histlines
- jiggle
- kbddate
- lenuniq
- make
- manyboxplot
- maxabs
- mypairs
- myround
- normalize
- note
- numbers
- objectsizes
- openfile
- paired.perm.test
- paste.
- paste00
- perm.test
- pick_more_precise
- plot_crayons
- pushbullet_devices
- qqline2
- qr2
- quantileSE
- revgrey
- revrainbow
**Description**

Convert a number to a string, with commas every 3rd digit

**Usage**

```r
add_commas(numbers)
```

**Arguments**

- `numbers` Vector of non-negative numbers (will be rounded to integers)

**Value**

Character string with numbers written like "7,547,085".

**Examples**

```r
add_commas(c(231, 91310, 2123, 9911001020, 999723285))
```
**align_vectors**  
*Align two vectors*

**Description**
Align two vectors using their names attributes, either expanding with NAs or reducing to the common values.

**Usage**
align_vectors(x, y, expand = TRUE)

**Arguments**
- **x**: A vector
- **y**: Another vector
- **expand**: If TRUE, expand each to the same length using NAs. If FALSE, remove elements not in common.

**Value**
A list with two components, x and y

**arrowlocator**  
*Use the locator function to plot an arrow*

**Description**
Use the `graphics::locator()` function to indicate the endpoints of an arrow and then plot it.

**Usage**
arrowlocator(reverse = FALSE, horizontal = FALSE, vertical = FALSE, length = 0.1, ...)

**Arguments**
- **reverse**: If FALSE, first indicate the tail of the arrow and then the head; if TRUE, first indicate the head of the arrow and then the tail.
- **horizontal**: If TRUE, force the arrow to be horizontal. (Use the average y-axis value of the two clicks for the vertical placement.)
- **vertical**: If TRUE, force the arrow to be vertical. (Use the average x-axis value of the two clicks for the horizontal placement.)
- **length**: Length of the edges of the arrow head.
- **...**: Additional graphics parameters
Details

Use `graphics::locator()` to indicate the two endpoints of an arrow and then draw it.

Value

The locations of the endpoints of the arrow, as a two-row matrix. The first row indicates the location of the tail of the arrow; the second row indicates the location of the head of the arrow.

See Also

`graphics::arrows()`, `graphics::locator()`

Examples

```r
## Not run:
plot(0,0,type="n", xlab="", ylab="", xlim=c(0,100), ylim=c(0,100))
arrows(locator(2), lwd=2)
## End(Not run)
```

---

**attrnames**

*Get names of attributes*

**Description**

Get the names of the attributes of an object

**Usage**

`attrnames(object)`

**Arguments**

- `object` Any object

**Details**

It just does `names(attributes(object))`.

**Value**

Vector of character strings with the names of the attributes.

**Examples**

```r
x <- matrix(1:100, ncol=5)
colnames(x) <- LETTERS[1:5]
attnames(x)
```
brocolors  

Vectors of colors for figures

Description

Creates different vectors of related colors that may be useful for figures.

Usage

brocolors(set = c("general", "general2", "bg", "bgpng", "CC", "CCalt", "f2", "sex", "main", "crayons", "web"))

Arguments

set  
Character string indicating a set of colors.

Value

Vector of character strings representing the chosen set of colors, in RGB.

See Also

plot_crayons()

Examples

par(mar=c(0.6,5.1,0.6,0.6))
plot(0, 0, type="n", xlab="", ylab="", xlim=c(0, 9), ylim=c(8.5, 0), yaxs="i",
xaxt="n", yaxt="n", xaxs="i")
axis(side=2, at=1:8, c("general", "general2", "bg", "bgpng", "CC", "f2", "sex", "main"), las=1)

gen <- brocolors("general")
points(seq(along=gen), rep(1,length(gen)), pch=21, bg=gen, cex=4)
text(seq(along=gen), rep(c(0.55, 0.7), length(gen))[seq(along=gen)], names(gen))

gen2 <- brocolors("general2")
points(seq(along=gen2), rep(2,length(gen2)), pch=21, bg=gen2, cex=4)
text(seq(along=gen2), rep(c(0.55, 0.7), length(gen2))[seq(along=gen2)], names(gen2))

points(1, 3, pch=21, bg=brocolors("bg"), cex=4)
points(1, 4, pch=21, bg=brocolors("bgpng"), cex=4)

CC <- brocolors("CC")
points(seq(along=CC), rep(5,length(CC)), pch=21, bg=CC, cex=4)
text(seq(along=CC), rep(c(0.55, 0.7), length(CC))[seq(along=CC)], names(CC))

f2 <- brocolors("f2")
points(seq(along=f2), rep(6,length(f2)), pch=21, bg=f2, cex=4)
text(seq(along=f2), rep(5.7, length(f2)), names(f2))
bromanversion

```
sex <- brocolors("sex")
points(seq(along=sex), rep(7, length(sex)), pch=21, bg=sex, cex=4)
text(seq(along=sex), rep(6.7, length(sex)), names(sex))
points(1, 8, pch=21, bg=brocolors("main"), cex=4)
```

---

**bromanversion**  
*Installed version of R/broman*

---

**Description**

Print the version number of the currently installed version of R/broman.

**Usage**

```
bromanversion()
```

**Value**

A character string with the version number of the currently installed version of R/broman.

**Examples**

```
bromanversion()
```

---

**cf**  
*Compare objects, including missing data pattern*

---

**Description**

Check whether two objects are the same, including their patterns of NAs.

**Usage**

```
 cf(a, b)
```

**Arguments**

- **a**  
  Some object.
- **b**  
  Another object

**Details**

It’s not very complicated: `((is.na(a) & is.na(b)) | (!is.na(a) & !is.na(b) & a == b))`
Value

Boolean object with TRUE indicating an element is the same.

Examples

```r
x <- c(5, 8, 9, NA, 3, NA)
y <- c(5, 2, 9, 4, NA, NA)
cf(x, y)

x <- matrix(rnorm(1000), ncol=20)
x[sample(seq(1:100), 100)] <- NA
all(cf(x, x))

y <- x
y[4, 8] <- NA
sum(!cf(x, y))
y[6, 2] <- 18
sum(!cf(x, y))
y[6, 5] <- 32
sum(!cf(x, y))

x <- as.data.frame(x)
y <- as.data.frame(y)
sum(!cf(x, y))

x <- as.list(x)
y <- as.list(y)
sapply(cf(x, y), function(a) sum(!a))
```

With `n.sim = 1000`

```r
chisq(tab, n.sim = 1000)
```

Arguments

- `tab` A matrix of counts.
- `n.sim` Number of samples of permuted tables to consider.
ciplot

Details

This is like the function stats::chisq.test(), but calculates an approximate P-value rather than referring to asymptotics. This will be better for large, sparse tables.

Value

A single number: the P-value testing independence of rows and columns in the table.

See Also

stats::chisq.test(), stats::fisher.test(), fisher()

Examples

TeaTasting <- matrix(c(3,1,1,3),nrow=2)
chisq(TeaTasting,1000)

ciplot(est, se = NULL, lo = NULL, hi = NULL, SEMult = 2,
       labels = NULL, rotate = FALSE, ...)

Arguments

est Vector of estimates
se Vector of standard errors
lo Vector of lower values for the intervals
hi Vector of upper values for the intervals
SEmult SE multiplier to create intervals
labels Labels for the groups (vector of character strings)
rotate If TRUE, have group as y-axis; default (FALSE) has group on x-axis.
... Optional graphics arguments
Details

Calls `grayplot()` with special choices of graphics parameters, as in `dotplot()`.

Provide either se or both lo and hi. In the case that se is used, the intervals will be est +/- SEmult * se.

If labels is not provided, group names are taken from the names(est). If that is also missing, we use capital letters.

You can control the CI line widths with `ci_lwd` and the color of the CI segments with `ci_col`. You can control the width of the segments at the top and bottom with `ci_endseg`.

Value

None.

See Also

`grayplot()`, `dotplot()`

Examples

```r
x <- rnorm(40, c(1,3))
g <- rep(c("A", "B"), 20)
me <- tapply(x, g, mean)
se <- tapply(x, g, function(a) sd(a)/sqrt(sum(!is.na(a))))
ciplot(me, se) # default is +/- 2 SE

# with SEmult

# rotate = FALSE

lo <- me - 2*se
hi <- me + 2*se

# rotate = TRUE

ciplot(me, se, rotate=TRUE)

lo <- me - 2*se
hi <- me + 2*se

# rotate = TRUE

ciplot(me, lo=lo, hi=hi)
```

---

**colwalpha**

Convert a color to use alpha transparency

**Description**

Convert a color to RGB and then to RGB with alpha transparency

**Usage**

`colwalpha(color, alpha = 1)`

**Arguments**

- **color**
  A character string for a color

- **alpha**
  Transparency value (between 0 and 1)
**compare_rows**

**Value**

A character string representing a color

**Examples**

```r
colwalpha(c("blue", "red"), 0.5)
```

**Description**

For all pairs of rows in a matrix, calculate the proportion of mismatches or the RMS difference.

**Usage**

```r
compare_rows(mat, method = c("prop_mismatches", "rms_difference"))
```

**Arguments**

- `mat` Numeric matrix. Should be integers in the case `method="prop_mismatches"`.
- `method` Indicates whether to use proportion mismatches or the RMS difference. Missing values are omitted.

**Value**

A square matrix of dimension `nrow(mat)` with NAs on the diagonal and the calculated statistic in the body.

**Examples**

```r
n <- 10
p <- 200
x <- matrix(sample(1:4, n*p, replace=TRUE), ncol=p)
d <- compare_rows(x)
```
**convert2hex**  
*Convert decimal to hex*

**Description**
Convert a number to hexadecimal notation.

**Usage**

```r
convert2hex(d)
```

**Arguments**
- `d` A number.

**Details**
Nothing important to say here.

**Value**
A character string; the input in hex.

**See Also**
- `hex2dec()`

**Examples**

```r
cr<convert2hex(333)
cr<dec2hex(333) == "14D"
cr<dec2hex(0:30)
```

---

**crayons**  
*Crayon colors*

**Description**
Vector of colors corresponding to Crayola crayons

**Usage**

```r
crayons(color_names = NULL, ...)
```
Arguments

- `color_names`: Optional vector of color names; can be partial matches.
- `...`: Additional optional color names

Value

- Vector of named RGB colors

References


See Also

- `plot_crayons()`, `brocolors()`

---

**done**

Send a short message via RPushbullet.

Description

Send a short message via RPushbullet, to be used to indicate that some R job is complete.

Usage

```r
done(message = "R is done", recipients = NULL)
```

Arguments

- `message`: A character string with a message. (passed to `RPushbullet::pbPost()`.)
- `recipients`: A character or numeric vector indicating the devices this post should go to. If NULL, the default device is looked up from an optional setting, and if none has been set the push is sent to all devices. (passed to `RPushbullet::pbPost()`.)

See Also

- `errors2pushbullet()`, `note()`, `pushbullet_devices()`

Examples

```r
## Not run: done("Your R job is complete.")
```
Description

Like the `grayplot()` function, but with one axis assumed to be categorical.

Usage

dotplot(group, y, jiggle = NULL, rotate = FALSE, ...)  

Arguments

- `group` Categorical coordinates for the plot  
- `y` Coordinates of points in the plot  
- `jiggle` Vector of amounts to jiggle the points horizontally, or a character string ("fixed" or "random") indicating the jigging method; see `jiggle()`.
- `rotate` If TRUE, have group as y-axis; default (FALSE) has group on x-axis.
- `...` Optional graphics arguments

Details

Calls `grayplot()` with special choices of graphics parameters for the case of categorical `x`.

If `group` is a factor, the order of the groups is as in the levels. Otherwise, we take `sort(unique(group))`. So if you want to control the order of the levels, make `group` a factor with the levels in the desired order, for example `group <- factor(group, levels=unique(group))`.

Value

None.

See Also

`grayplot()`

Examples

```r
x <- rnorm(40, c(1,3))
g <- rep(c("A", "B"), 20)
dotplot(g, x)
dotplot(g, x, "random")
dotplot(g, x, runif(length(g), -0.25, 0.25))
```
errors2pushbullet  Send further errors to pushbullet

Description

Set options to use RPushbullet to use pushbullet to push notifications of any error messages.

Usage

```
errors2pushbullet(recipients = NULL)
```

Arguments

- **recipients** A character or numeric vector indicating the devices this post should go to. If NULL, the default device is looked up from an optional setting, and if none has been set the push is sent to all devices. (passed to `RPushbullet::pbPost()`.)

See Also

- `stop_sending_errors()`, `note()`, `done()`, `pushbullet_devices()`

Examples

```
## Not run: errors2pushbullet()
```

---

excel_fig  Excel-style figure displaying contents of a matrix

Description

Turn a matrix of data into an SVG of how it might look in Excel

Usage

```
excel_fig(mat, file = NULL, cellwidth = 80, cellheight = 26,
  textsize = 16, fig_width = NULL, fig_height = NULL,
  border = "#CECECE", headcol = "#E9E9E9", headborder = "#969696",
  headtextcol = "#626262", textcol = "black", row_names = FALSE,
  col_names = TRUE, hilitcells = NULL, hilitcolor = "#F0DCDB",
  lwd = 1, direct2svg = FALSE, mar = rep(0.1, 4))
```
Arguments

mat A matrix
file Optional file name (must have extension .svg, .png, .jpg, or .pdf)
cellwidth Width of each cell, in pixels
cellheight Height of each cell, in pixels
textsize Size for text (if file is provided or direct2svg=TRUE)
fig.width Width of figure, in pixels (if NULL, taken from cellwidth); ignored when direct2svg=FALSE
fig.height Height of figure, in pixels (if NULL, taken from cellheight); ignored when direct2svg=FALSE
border Color of border of cells for the body of the matrix
headcol Background color of cells on the top and left border
headborder Color of border of cells on the top and left border
headtextcol Color of text in cells on the top and left border
textcol Color of text in cells in body of the matrix
row_names If TRUE, and row names are present, include them as a first column
col_names If TRUE, and column names are present, include them as a first row
hilitcells Optional character vector of cells to highlight, like "A1" or "D4"
hilitcolor Color to highlight cells, a vector of length 1 or the same length as hilitcells
lwd Line width for rectangles
direct2svg If TRUE, rather than R graphics, just print an SVG directly with base::cat().
mar Plot margins, passed to graphics::par().

Examples

df <- data.frame(id=c(101, 102, 103),
sex=c("M","F","M"),
weight=c(22.3, 15.8, 19.7),
stringsAsFactors=FALSE)
excel_fig(df, col_names=TRUE)

exit R without saving

Description
exit R without saving workspace.

Usage
exit()
fac2num

Details
This just calls q("no")

Value
None.

fac2num | Convert a factor to numeric

Description
Convert a factor with numeric levels to a non-factor

Usage
fac2num(x)

Arguments
x | A vector containing a factor with numeric levels

Value
The input factor made a numeric vector

Examples
x <- factor(c(3, 4, 9, 4, 9), levels=c(3,4,9))
fac2num(x)

fisher | Fisher's exact test for a two-way table

Description
Performs a sampling version of Fisher's exact test for a two-way contingency table.

Usage
fisher(tab, n.sim = 1000)

Arguments
tab | A matrix of counts.
n.sim | Number of samples of permuted tables to consider.
Details

This is like the function \texttt{stats::fisher.test()}, but calculates an approximate P-value rather than performing a complete enumeration. This will be better for large, sparse tables.

Value

A single number: the P-value testing independence of rows and columns in the table.

See Also

\texttt{stats::chisq.test()}, \texttt{stats::fisher.test()}, \texttt{chisq}

Examples

\begin{example}
TeaTasting <- matrix(c(3,1,1,3),nrow=2)
fisher(TeaTasting,1000)
\end{example}

---

\begin{function}
get_precision

\textit{Determine the precision of a number}

\textbf{Description}

Determine the precision of a number, as the number of digits past the decimal point.

\textbf{Usage}

\texttt{get\_precision(x, ...)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} A numeric vector
  \item \texttt{...} \hspace{1cm} Ignore this
\end{itemize}

\textbf{Details}

If the number is expressed in scientific notation, we take the number of digits

\textbf{Value}

A vector of integers, with the number of digits (to the last non-zero digit) past the decimal point.
grayplot

Scatterplot with a gray background

Description

Like the plot function, but using a gray background just for the plot region.

Usage

```r
grayplot(x, y = NULL, ..., type = "p", hlines = NULL,
          hlines.col = "white", hlines.lty = 1, hlines.lwd = 1,
          vlines = NULL, vlines.col = "white", vlines.lty = 1,
          vlines.lwd = 1, xat = NULL, yat = NULL, bgcolor = "gray90",
          pch = 21, bg = "lightblue", col = "black", v_over_h = FALSE)
```

Arguments

- **x**: Coordinates of points in the plot
- **y**: Coordinates of points in the plot (optional)
- **...**: Optional graphics arguments
- **type**: Plot type (points, lines, etc.)
- **hlines**: Locations of horizontal grid lines; use hlines=NA to prevent horizontal grid lines
- **hlines.col**: Colors of horizontal grid lines
- **hlines.lty**: Line type of horizontal grid lines
- **hlines.lwd**: Line width of horizontal grid lines
- **vlines**: Locations of vertical grid lines; use vlines=NA to prevent vertical grid lines
- **vlines.col**: Colors of vertical grid lines
- **vlines.lty**: Line type of vertical grid lines
- **vlines.lwd**: Line width of vertical grid lines
- **xat**: Locations for x-axis labels; xat=NA indicates no labels
- **yat**: Locations for y-axis labels; yat=NA indicates no labels
- **bgcolor**: Background color
- **pch**: point type
- **bg**: Background color in points
- **col**: Color of outer circle in points
- **v_over_h**: If TRUE, place vertical grid lines on top of the horizontal ones.

Details

Calls `graphics::plot()` with type="n", then `graphics::rect()` to get the background, and then `graphics::points()`. Additional arguments you can include: mgp.x and mgp.y (like mgp, for controlling parameters of axis labels, but separate for x- and y-axis).
Value

None.

See Also

`graphics::plot()`, `graphics::par()`, `graphics::rect()`, `graphics::points()`

Examples

```r
x <- rnorm(100)
y <- x+rnorm(100, 0, 0.7)
grayplot(x, y, col=“slateblue”, pch=16)
at <- seq(-3, 3)
grayplot(x, y, col=“violetred”, pch=16, hlines=at, vlines=at)
grayplot(x, col=“Orchid”, pch=16, bgcolor=“gray90”,
          hlines=seq(-4, 4, by=0.5), hlines.lwd=c(3,1),
          vlines=seq(0, 100, by=5), vlines.lwd=c(3,1,1,1))
```

Description

Scatterplot with a gray background and with points with missing values shown in separate panels near the margins.

Usage

```r
grayplot_na(x, y = NULL, type = "p", bgcolor = "gray90",
            v_over_h = FALSE, pch = 21, bg = "lightblue", col = "black",
            force = c("none", "x", "y", "both"), ...)
```

Arguments

- `x`: Coordinates of points in the plot
- `y`: Coordinates of points in the plot (optional)
- `type`: Plot type (points, lines, etc.)
- `bgcolor`: Background color
- `v_over_h`: If TRUE, place vertical grid lines on top of the horizontal ones.
- `pch`: Point type
- `bg`: Background color in points
- `col`: Color of outer circle in points
- `force`: Indicates whether to force the NA box (on the x-axis, y-axis, or both) even when there are no missing values.
- `...`: Optional graphics arguments
Details

Calls `graphics::plot()` with `type="n"`, then `graphics::rect()` to get the background, and then `graphics::points()`.

There are a bunch of hidden graphical arguments you can include: `na.width` controls the proportional width devoted to the NA boxes, and `na.gap` the proportion for the gap between the NA boxes and the main plot region. `mgp.x` and `mgp.y` (like `mgp`, for controlling parameters of axis labels, but separate for x- and y-axis). Also `hlines` to indicate locations of of horizontal gridlines, and `hlines.col`, `hlines.lwd`, and `hlines.lty` to set their color, width, and type. `hlines=NA` suppresses the grid lines. Similarly `vlines`, `vlines.col`, `vlines.lwd`, and `vlines.lty`. `xat` and `yat` are for specifying the locations of x- and y-axis labels, respectively. `xat=NA` and `yat=NA` indicate no labels.

Value

None.

See Also

`grayplot()`, `dotplot()`

Examples

```r
n <- 100
x <- rnorm(n)
y <- x+rnorm(n, 0, 0.7)
x[sample(n, 10)] <- NA

grayplot_na(x, y)

grayplot_na(x, y, force="y")

y[sample(n, 10)] <- NA
grayplot_na(x, y)
```

Description

View the html version of a help file while running R via ESS within emacs.

Usage

`h(...)`
Arguments

... Help topics.

Details

This just calls the function `utils::help()` using the argument `htmlhelp=TRUE`.

Value

No return value.

See Also

`utils::help()`, `utils::help.start()`

Examples

h(read.cross)

---

**hex2dec**

*Convert from hex to decimal*

Description

Convert a number from hexadecimal to decimal notation.

Usage

`hex2dec(h)`

Arguments

h  # Character string with hexadecimal representation of a number

Details

Nothing important to say here.

Value

The input converted from hexadecimal to decimal notation.

Author(s)

Karl W Broman, <broman@wisc.edu>
**histlines**  

*Utility to create line-based histogram*

**Description**

Utility function to plot histogram with `graphics::lines()`.

**Usage**

```r
histlines(x, y = NULL, breaks, use = c("counts", "density"))
```

**Arguments**

- `x`: Either vector of breaks or the data itself.
- `y`: Optional vector of density/counts, with length = `length(x)-1`.
- `breaks`: Breaks for histogram, if `y` is not provided.
- `use`: Whether to use counts or density, if `y` is not provided.

**Details**

If `x` and `y` are both provided, `x` is interpreted to be the breaks for a histogram, and `y` is a vector of counts or density values for each interval. These are then revised so that they may be plotted with `graphics::lines()`. If `y` is NULL, `x` is taken to be the data. In this case `graphics::hist()` is called with `breaks=breaks`, and either the counts or density are used as `y`.

**Value**

A data.frame with two columns: `x` and `y`.

**See Also**

`graphics::hist()`, `graphics::lines()`
Examples

```r
x <- rnorm(1000, mean=20, sd=5)
# basic use
out <- hist(x, breaks=60, plot=FALSE)
plot(histlines(out$breaks, out$counts),
     type="l", lwd=2, xlab="x", ylab="counts", las=1)
# alternative use
plot(histlines(x, breaks=60, use="density"),
     type="l", lwd=2, xlab="x", ylab="Density", las=1)
# comparing two distributions
z <- rnorm(1000, mean=25, sd=5)
br <- seq(min(c(x,z)), max(c(x,z)), len=50)
xlines <- histlines(x, breaks=br, use="density")
zlines <- histlines(z, breaks=br, use="density")
ymx <- max(c(xlines$y, zlines$y))*1.05
plot(xlines, ylim=c(0, ymx), yaxs="i", xaxs="i",
     type="l", lwd=2, xlab="x", ylab="Density", las=1,
     col="blue")
lines(zlines, lwd=2, col="red")
```

---

**jiggle**

**Jiggle points horizontally**

Description

Spread points out horizontally so that, in dot plot of quantitative response in multiple categories, the separate points can be seen.

Usage

```r
jiggle(group, y, method = c("fixed", "random"), hnum = 35, vnum = 40)
```

Arguments

- **group**: Categorical variable defining group; can be a factor, character, or numeric vector
- **y**: Vector of quantitative responses
- **method**: What method to use for horizontal jiggling.
- **hnum**: Number of horizontal bins for the jiggling.
- **vnum**: Number of vertical bins for the jiggling.

Details

The "random" method is similar to `base::jitter()` but with amount of jiggling proportional to the number of nearby points. The "fixed" method is similar to the `beeswarm` package.
Value

Numeric vector with amounts to jiggle the points horizontally

See Also

`base::jitter()`, `dotplot()`

---

**kbdate**

*My little date facility*

Description

Sys.Date as a string, in a few different formats

Usage

```r
kbdate(format = c("dateonly", "standard"), date = Sys.time())
```

Arguments

- `format` The format for the output
- `date` The date/time to convert

Value

A character string representation of the date/time

See Also

`base::Sys.time()`, `base::date()`

Examples

```r
kbdate()
kbd(date("standard"))
```
Description

Get the number of unique values in a vector

Usage

lenuniq(vec, na.rm = TRUE)

Arguments

vec A vector
na.rm If TRUE, remove any missing values

Details

It just does length(unique(vec)) or, if na.rm=TRUE (the default) length(unique(vec[!is.na(vec)]))

Value

Number of unique values.

Examples

x <- c(1, 2, 1, 3, 1, 2, 2, 3, NA, NA, 1)
lenuniq(x)
lenuniq(x, na.rm=FALSE)

Description

Run make within a package directory

Usage

make(pkg = ".", makefile = "Makefile", target = "", quiet = FALSE)
**manyboxplot**  

**Arguments**  

- **pkg**  
  Path to directory containing the GNU Make file, or an R package description, which can be a path or a package name. (See `devtools::as.package()` for more information.)  

- **makefile**  
  File name of makefile.  

- **target**  
  Optional character string specifying the target.  

- **quiet**  
  If TRUE suppresses output from this function.  

**Value**  

Exit value from `base::system()` with intern=FALSE  

**See Also**  

`devtools::load_all()`  

**Examples**  

```r  
## Not run: make() # run make within working directory  
make("/path/to/mypackage") # run make within /path/to/mypackage  
## End(Not run)  
```

---

**manyboxplot**  

**Boxplot-like figure for many groups**  

**Description**  

Boxplot-like figure for many groups, with lines connecting selected quantiles.  

**Usage**  

```r  
manyboxplot(x, probs = c(0.05, 0.1, 0.25), dotcol = "blue",  
  linecol = c("black", "red", "green", "orange"), ...)  
```

**Arguments**  

- **x**  
  Matrix of data, with columns indicating the groups.  

- **probs**  
  Numeric vector of probabilities with values in [0,1). Quantiles will be symmetric, and the median will always be included.  

- **dotcol**  
  Color for median  

- **linecol**  
  Line colors, same length as `probs`  

- **...**  
  Additional graphics parameters
Details

Calculates quantiles of the columns of \( x \) and then plots dots or lines at median plus lines at a series of quantiles, using `grayplot()` for the actual plot.

Value

None.

See Also

`grayplot()`

Examples

```r
mu <- c(rnorm(50, 0, 0.3), rnorm(50, 2, 0.3))  # vector of means
x <- t(matrix(rnorm(1000*100, mu), ncol=1000))
manyboxplot(x, c(0.05, 0.25), ylim=range(x),
            dotcol=c("blue","green")[(1:100 > 50) + 1],
            hlines=seq(-4, 6, by=2),
            vlines=c(1, seq(20, 100, by=20)))
```

---

Describes `maxabs`

The maximum of absolute value

Usage

`maxabs(x, na.rm = FALSE)`

Arguments

- `x` a numeric vector or array
- `na.rm` a logical indicating whether missing values should be removed.

Value

The maximum of the absolute value of the input

Examples

```r
x <- c(5, -2, 8, -20, 2.3)
maxabs(x)
```
mypairs  

My scatterplot matrix

Description
A matrix of scatterplots is produced; it’s similar to `graphics::pairs()`, but with only the upper triangle is made.

Usage
```r
mypairs(x, ...)
```

Arguments
- **x**: A numeric matrix or data frame.
- **...**: Passed to the `graphics::plot()` function.

Details
This is like the function `graphics::pairs()`, but only the upper triangle is produced.

Value
None.

See Also
- `graphics::pairs()`

Examples
```r
v <- rbind(c(1,0.5,0.2),c(0.5,1,0.9),c(0.2,0.9,1))
x <- rmvn(500, rep(5,3), v)
mypairs(x, col=sample(c("blue","red"), 500, repl=TRUE))
```

myround  

Round a number, preserving extra 0’s

Description
Round a number, preserving extra 0’s.

Usage
```r
myround(x, digits = 1)
```
**normalize**

**Arguments**

- `x` Number to round.
- `digits` Number of digits past the decimal point to keep.

**Details**

Uses `base::sprintf()` to round a number, keeping extra 0’s.

**Value**

A vector of character strings.

**See Also**

`base::round()`, `base::sprintf()`

**Examples**

```r
myround(51.01, 3)
myround(0.199, 2)
```

---

**normalize**  
Quantile normalization

**Description**

Quantile normalizes two vectors or a matrix.

**Usage**

```r
normalize(x, y = NULL)
```

**Arguments**

- `x` Numeric vector or matrix
- `y` Optional second numeric vector

**Details**

We sort the columns, take averages across rows, and then plug the averages back into the respective positions. The marginal distributions in the columns are thus forced to be the same. Missing values, which can result in differing numbers of observed values per column, are dealt with by linear interpolation.
Value

If two vectors, \(x\) and \(y\), are provided, the output is a matrix with two columns, with the quantile normalized versions of \(x\) and \(y\). If \(y\) is missing, \(x\) should be a matrix, in which case the output is a matrix of the same dimensions with the columns quantile normalized with respect to each other.

Examples

```r
z <- rmvn(10000, mu=c(0.5,10), V = rbind(c(1,0.5,0.5),c(0.5,1,0.5),c(0.5,0.5,1)))
z[sample(prod(dim(z)), 1500)] <- NA
pairs(z)
br <- seq(min(z, na.rm=TRUE), max(z, na.rm=TRUE), length=200)
par(mfrow=c(3,1))
for(i in 1:3)
  hist(z[,i], xlab="z", main=i, breaks=br)
zn <- normalize(z)
br <- seq(min(zn, na.rm=TRUE), max(zn, na.rm=TRUE), length=200)
for(i in 1:3)
  hist(zn[,i], xlab="normalized z", main=i, breaks=br)
pairs(zn)
```

Description

Even simpler interface for `RPushbullet::pbPost()` to send a bit of text to pushbullet.

Usage

```
note(title, recipients = NULL, body = NULL)
```

Arguments

- **title**: The title of the note (could be the whole thing).
- **recipients**: A character or numeric vector indicating the devices this post should go to. If `NULL`, the default device is looked up from an optional setting, and if none has been set the push is sent to all devices. (passed to `RPushbullet::pbPost()`)
- **body**: The body of the note (by default, empty)

See Also

`errors2pushbullet(), done(), pushbullet_devices()`

Examples

```r
## Not run: note("Hello.")
```
numbers

 Numbers spelled out in English

Description
The numbers 1-20 spelled out in English, for use in reports.

Format
A vector of character strings

Details
- numbers - lower case
- Numbers - Capitalized

Examples
numbers[5]
Numbers[5]

objectsizes

 Calculate sizes of all objects in workspace

Description
Calculate the sizes of all of the objects in one's workspace.

Usage
objectsizes(obj = NULL, sortbysize = TRUE)

Arguments
- obj Vector of object names. If missing, we pull out all object names.
- sortbysize If TRUE, sort the objects from smallest to largest.

Details
Calls utils::object.size() repeated to get the size of a list of objects.

Value
A data frame with the only column being the size of each object in megabytes (Mb). The row names are the names of the objects.
openfile

See Also

utils::object.size(), base::objects()

Examples

print(output <- objectsizes())
## Not run: sum(output)

openfile(file)

Arguments

file File name (character string)

Details

I’d thought that to open a file you’d use open in MacOS and start in Windows, but system("start myfile.pdf") doesn’t work in Windows, and rather system("open myfile.pdf") does, so here we’re just using open, except on Linux where at least on my system, you can use "start".

Value

None.

Examples

## Not run: openfile("myplot.pdf")
paired.perm.test

**Paired permutation t-test**

**Description**
Calculates a p-value for a paired t-test via permutations.

**Usage**

```r
paired.perm.test(d, n.perm = NULL, pval = TRUE)
```

**Arguments**

- `d`: A numeric vector (of differences).
- `n.perm`: Number of permutations to perform. If NULL, all possible permutations are considered, and an exact p-value is calculated.
- `pval`: If TRUE, return just the p-value. If FALSE, return the actual permutation results (with the observed statistic as an attribute, "tobs").

**Details**

This calls the function `stats::t.test()` to calculate a t-statistic comparing the mean of `d` to 0. Permutations are performed to give an exact or approximate conditional p-value.

**Value**

If `pval=TRUE`, the output is a single number: the P-value testing for the symmetry about 0 of the distribution of the population from which `d` was drawn. If `pval=FALSE`, the output is a vector of the t statistics from the permutations. An attributed "tobs" contains the t statistic with the observed data.

**See Also**

`stats::t.test()`, `perm.test()`

**Examples**

```r
x <- c(43.3, 57.1, 35.0, 50.0, 38.2, 31.2)
y <- c(51.9, 95.1, 90.0, 49.7, 101.5, 74.1)
paired.perm.test(x-y)
```
Description
Calls `base::paste()` with `sep="."`.  

Usage
`paste(...)`

Arguments
... Passed to `paste`.

Details
There’s not much to this function. It just is `base::paste()` with `sep="."`, ’cause I’m lazy.

Value
A character string or vector of character strings.

See Also
`base::paste(), base::paste0(), paste0(), paste(), paste0(), paste0()`

Examples
```r
x <- 3
y <- 4
paste(x, y)
```

Description

Call `base::paste()` with `sep="."` or `sep=""` and `collapse=""` or `collapse="."`.

Usage
`paste00(...)`

```r
```
Arguments

... Passed to paste.

Details

There’s not much to these functions. `paste00(...)` is like `paste(..., sep="", collapse="")`
`paste01(...)` is like `paste(..., sep=".", collapse="")`
`paste02(...)` is like `paste(..., sep="", collapse=".")`
`paste03(...)` is like `paste(..., sep=".", collapse="")`

Value

A character string or vector of character strings.

See Also

`base::paste()`, `base::paste00()`, `paste()`

Examples

```r
x <- c(3, 4)
y <- c(5, 6)
paste00(x, y)
paste01(x, y)
paste02(x, y)
paste03(x, y)
```

Description

Calculates a p-value for a t-test via permutations.

Usage

```r
perm.test(x, y, n.perm = NULL, var.equal = TRUE, pval = TRUE)
```

Arguments

- `x` A numeric vector.
- `y` A second numeric vector.
- `n.perm` Number of permutations to perform. If NULL, all possible permutations are considered, and an exact p-value is calculated.
- `var.equal` A logical variable indicating whether to treat the two population variances as being equal.
- `pval` If TRUE, return just the p-value. If FALSE, return the actual permutation results (with the observed statistic as an attribute, "tobs").
Details
This calls the function `stats::t.test()` to calculate a t-statistic comparing the vectors x and y. Permutations are performed to give an exact or approximate conditional p-value.

Value
If `pval=TRUE`, the output is a single number: the P-value testing for a difference in the distributions of the populations from which x and y were drawn. If `pval=FALSE`, the output is a vector of the t statistics from the permutations. An attribute “`tobs`” contains the t statistic with the observed data.

See Also
`stats::t.test()`, `paired.perm.test()`

Examples
```r
x <- c(43.3, 57.1, 35.0, 50.0, 38.2, 61.2)
y <- c(51.9, 95.1, 90.0, 49.7, 101.5, 74.1)
perm.test(x,y)
```

Description
Align two vectors of numbers by their names and then pick a single value from each, favoring the more precise one. If the two values differ by more than round-off error, treat the value as missing.

Usage
```r
pick_more_precise(x, y, tol = 0.000001)
```

Arguments
- `x`: A numeric vector
- `y`: A second numeric vector
- `tol`: Tolerance for differences between the values

Details
Okay, this is a bit weird. But suppose you have two columns of numbers that have been subjected to different quirky rounding patterns. We align the vectors using their names and then for each element we pick between the two choices, favoring the more-precise one. If one is missing, choose the non-missing value. If the two differ by more than the round-off error, treat it as missing.
Value

A vector of combined values

Description

Creates a plot of the crayon colors in `brocolors()`

Usage

```r
plot_crayons(method=order, cex = 0.6, 
mar = rep(0.1, 4), bg = "white", fg = "black", border = FALSE)
```

Arguments

- `method`: method to order colors ("hsv" or "cluster")
- `cex`: character expansion for the text
- `mar`: margin parameters; vector of length 4 (see `graphics::par()`)
- `bg`: Background color
- `fg`: Foreground color (for text and box outlines)
- `border`: If TRUE, plot a border around each rectangle

Value

None

References


See Also

`brocolors()`

Examples

```r
plot_crayons()
```
pushbullet_devices

Grab info on Pushbullet devices.

Description

Get names and identifiers of Pushbullet devices.

Usage

pushbullet_devices()

Value

data frame with nickname, model, and iden for active devices.

See Also

ersors2pushbullet(), done(), note()

qqline2

qqline for qqplot

Description

Adds a line to a quantile-quantile plot for two datasets, from stats::qqplot. (The available
stats::qqline function works mainly for stats::qnorm, with one sample being theoretical
quantiles.)

Usage

qqline2(x, y, probs = c(0.25, 0.75), qtype = 7, ...)

Arguments

x The first sample
y The second sample.
probs numeric vector of length two, representing probabilities. Corresponding quantile pairs define the line drawn.
qtype the type of quantile computation used in stats::quantile().
... graphical parameters.

Value

Intercept and slope of the line.
See Also

\texttt{stats::qqline()}, \texttt{stats::qqplot()}

Examples

\begin{verbatim}
x <- rchisq(500, 3)
y <- rgamma(730, 3, 1/2)
qqplot(x, y)
qqline2(x, y)
\end{verbatim}

---

\texttt{qr2} \quad \textit{The QR decomposition of a matrix}

Description

Computes the QR decomposition of a matrix.

Usage

\begin{verbatim}
qr2(x, tol = 0.0000001)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} \quad A matrix whose QR decomposition is to be computed.
\item \texttt{tol} \quad The tolerance for detecting linear dependencies in the columns of \texttt{x}.
\end{itemize}

Details

Calls the function \texttt{base::qr()} and returns less compact but more understandable output.

Value

A list of two matrices: Q and R.

See Also

\begin{verbatim}
base::qr()
\end{verbatim}

Examples

\begin{verbatim}
hilbert <- function(n) { i <- 1:n; 1/outer(i-1,i,"*") }
h5 <- hilbert(5);
qr2(h5)
\end{verbatim}
Sample quantiles and their standard errors

**Description**

Calculate sample quantiles and their estimated standard errors.

**Usage**

```r
quantileSE(x, p = 0.95, bw = NULL, na.rm = TRUE, names = TRUE)
```

**Arguments**

- `x` Numeric vector whose sample quantiles are wanted.
- `p` Numeric vector with values in the interval [0,1]
- `bw` Bandwidth to use in the density estimation.
- `na.rm` Logical; if true, and NA and NaN's are removed from x before the quantiles are computed.
- `names` Logical; if true, the column names of the result is set to the values in p.

**Details**

The sample quantiles are calculated with the function `stats::quantile()`. Standard errors are obtained by the asymptotic approximation described in Cox and Hinkley (1974). Density values are estimated using a kernel density estimate with the function `stats::density()`.

**Value**

A matrix of size 2 x length(p). The first row contains the estimated quantiles; the second row contains the corresponding estimated standard errors.

**See Also**

`stats::quantile()`, `stats::density()`

**Examples**

```r
quantileSE(rchisq(1000,4), c(0.9,0.95))
```
revgray

Create vector of colors from white to black

Description

Calls `grDevices::gray()` then `base::rev()`

Usage

`revgray(n = 256, ...)`

Arguments

- `n` Number of colors.
- `...` Passed to `grDevices::gray()`.

Details

There’s not much to this. It’s just `gray((n:0)/n)`

Value

Vector of colors, from white to black

See Also

`grDevices::gray()`

Examples

```r
x <- matrix(rnorm(100), ncol=10)
image(x, col=revgray())
```

revrainbow

Create vector of colors from blue to red

Description

Calls `grDevices::rainbow()` then `base::rev()`

Usage

`revrainbow(n = 256, ...)`
Arguments

- `n` Number of colors.
  - Passed to `grDevices::rainbow()`.

Details

There's not much to this. It's just `rev(rainbow(start=0, end=2/3, ...))`.

Value

Vector of colors, from blue to red.

See Also

`base::rev(), grDevices::rainbow()`.

Examples

```r
x <- matrix(rnorm(100), ncol=10)
image(x, col=revrainbow())
```

---

## `rmvn`

**Simulate multivariate normal**

Description

Simulate from a multivariate normal distribution.

Usage

```r
rmvn(n, mu = 0, V = matrix(1))
```

Arguments

- `n` Number of simulation replicates.
- `mu` Mean vector.
- `V` Variance-covariance matrix.

Details

Uses the Cholesky decomposition of the matrix `V`, obtained by `base::chol()`.

Value

A matrix of size `n x length(mu)`. Each row corresponds to a separate replicate.
See Also

stats::rnorm()

Examples

```r
x <- rmvm(100, c(1,2), matrix(c(1,1,1,4), ncol=2))
```

---

**runningmean**  
*Running mean, sum, or median*

**Description**

Calculates a running mean, sum or median with a specified window.

**Usage**

```r
runningmean(pos, value, at = NULL, window = 1000, what = c("mean", "sum", "median", "sd"))
```

**Arguments**

- `pos` Positions for the values.
- `value` Values for which the running mean/sum/median/sd is to be applied.
- `at` Positions at which running mean (or sum or median or sd) is calculated. If NULL, `pos` is used.
- `window` Window width.
- `what` Statistic to use.

**Value**

A vector with the same length as the input at (or pos, if at is NULL), containing the running statistic.

**Author(s)**

Karl W Broman <broman@wisc.edu>

**See Also**

runningratio()
**runningratio**

**Examples**

```r
x <- 1:10000
y <- rnorm(length(x))
plot(x, y, xaxs="i", yaxs="i")
lines(x, runningmean(x, y, window=100, what="mean"),
     col="blue", lwd=2)
lines(x, runningmean(x, y, window=100, what="median"),
     col="red", lwd=2)
lines(x, runningmean(x, y, window=100, what="sd"),
     col="green", lwd=2)
```

**runningratio**  

**Running ratio**

**Description**

Calculates a running ratio; a ratio sum(top)/sum(bottom) in a sliding window.

**Usage**

```
runningratio(pos, numerator, denominator, at = NULL, window = 1000)
```

**Arguments**

- `pos` Positions for the values.
- `numerator` Values for numerator in ratio.
- `denominator` Values for denominator in ratio.
- `at` Positions at which running ratio is calculated. If NULL, `pos` is used.
- `window` Window width.

**Value**

A vector with the same length as the input `at` (or `pos`, if `at` is NULL), containing the running ratio.

**Author(s)**

Karl W Broman <brroman@wisc.edu>

**See Also**

`runningmean()`
Examples

```r
x <- 1:1000
y <- runif(1000, 1, 5)
z <- runif(1000, 1, 5)
plot(x, runningratio(x, y, z, window=5), type="l", lwd=2)
lines(x, runningratio(x, y, z, window=50), lwd=2, col="blue")
lines(x, runningratio(x, y, z, window=100), lwd=2, col="red")
```

---

**setRNGparallel**

*Set up random number generation for parallel calculations*

**Description**

Set random number generation to L’Ecuyer-CMRG, for use in parallel calculations.

**Usage**

```r
setRNGparallel()
```

```r
unsetRNGparallel()
```

**Details**

I can never remember the command `RNGkind("L'Ecuyer-CMRG");` this is a shortcut. `unsetRNG4parallel` sets the random number generator back to the default type.

**Examples**

```r
RNGkind()
setRNGparallel()
RNGkind()
unsetRNGparallel()
RNGkind()
```

---

**simp**

*Numerical integration*

**Description**

Perform numerical integration by Simpson’s rule or the trapezoidal rule.

**Usage**

```r
simp(f, a, b, tol = 0.00000001, max.step = 1000, ...)
```
spell_out

Arguments

- **f**: The integrand; must be a vectorized function.
- **a**: Lower limit of integration.
- **b**: Upper limit of integration.
- **tol**: Tolerance for choosing the number of grid points.
- **max.step**: Log base 2 of the total number of grid points.
- **...**: Other arguments passed to the integrand, `f`.

Details

Iterately doubles the number of grid points for the numerical integral, stopping when the integral decreases by less than `tol`.

Value

The integral of `f` from `a` to `b`.

See Also

`stats::integrate()`

Examples

```r
f <- function(x) x**x*(1-x)*sin(x**x)
I1 <- trap(f,0,2)
I2 <- simp(f,0,2)
```

---

spell_out

*Spell out an integer*

Description

Spell out an integer as a word, for use in reports/papers.

Usage

```r
spell_out(number, capitalize = FALSE, max_value = 9)
```

Arguments

- **number**: A number that is to be spelled out (can be a vector).
- **capitalize**: If TRUE, capitalize the first letter.
- **max_value**: Maximum value to use (generally 9); if larger than this, use numerals.
Value

Character string (or vector of character strings) with numbers spelled out, or as numerals if large.

Examples

spell_out(9)
spell_out(9, cap=TRUE)
spell_out(9, max_value=5)

stop_sending_errors  Stop sending errors to pushbullet

Description

Clear the error option, so that error notifications are no longer sent to pushbullet.

Usage

stop_sending_errors()

See Also

errors2pushbullet(), pushbullet_devices()

Examples

## Not run: stop_sending_errors()

strwidth2lines  Calculate width of a character string in number of lines

Description

Convert stringwidth units to number of (margin) lines

Usage

strwidth2lines(s, ...)

Arguments

s  A character or expression vector whose length is to be calculated

...  additional information used by strwidth, such as cex
Value

Maximum string width in units of margin lines

Author(s)

Aimee Teo Broman

Examples

```r
p <- par(TRUE)
string <- sapply(sample(1:20,15,replace=TRUE),
    function(a) paste(LETTERS[1:a], collapse=""))
nlines <- strwidth2lines(string)
mar <- par("mar")
par(mar=c(mar[1],nlines+0.1,mar[3:4]))
    plot(1:length(string),1:length(string),yaxt="n", ylab=""
    axis(side=2, at=seq_along(string), lab=string, las=1)
par(p)
nlines <- strwidth2lines(string,cex=1.5)
par(mar=c(mar[1:3],nlines+0.1))
    plot(1:length(string),1:length(string),ylab="")
mgp <- par("mgp")
    axis(side = 4, at=seq_along(string),
        labels = string ,las=1, hadj=1,
        mgp=c(mgp[1],nlines,mgp[3]),cex.axis=1.5)
par(p)
```

---

**strwidth2xlim**

*Calculate horizontal limit in user coordinates for adding labels*

Description

Calculates the x-axis limits when adding (long) labels to a plot

Usage

```
strwidth2xlim(x, xstring, pos = 4, offset = 0.5, ...)
```

Arguments

- `x` numeric vector of horizontal coordinates
- `xstring` character vector, specifying text to be written
- `pos` position specifier for text; values of 1, 2, 3, and 4, respectively, indicate positions below, to the left of, above, and to the right of the coordinates
- `offset` offset of the label from the coordinate in fractions of a character width
- `...` additional text parameters from `par`, such as `cex`
Details
See text for details on pos and offset.

Value
Minimum and maximum x-axis limits for adding horizontal text

Author(s)
Aimee Teo Broman

See Also
graphics::text()

Examples
x <- runif(15, -1, 1)*10
xlabs <- sapply(sample(1:20, 15, replace=TRUE),
                function(a) paste(LETTERS[1:a], collapse=""))
## Labels to the left ##
xlims <- strwidth2xlim(x, xlabs, pos=2)
plot(x, 1:length(x), xlim=xlims)
text(x, 1:length(x), xlabs, pos=2)
## Labels to the right ##
xlims <- strwidth2xlim(x, xlabs, pos=4, cex=0.7)
plot(x, 1:length(x), xlim=xlims)
text(x, 1:length(x), xlabs, pos=4, cex=0.7)

---

switchv

Vectorized version of switch

Description
Vectorized version of base::switch(): just loops over input and calls base::switch().

Usage
switchv(EXPR, ...)

Arguments

EXPR An expression evaluating to a vector of numbers of strings
...
List of alternatives

Value
Vector of returned values.
theme_karl

Examples

```r
switchv(c("horse", "fish", "cat", "bug"),
  horse="fast",
  cat="cute",
  "what?")
```

Description

Karl's ggplot2 theme: black border and no ticks

Usage

```r
theme_karl(base_size = 12, base_family = ",", ...)
karl_theme(base_size = 12, base_family = "", ...)
```

Arguments

- `base_size`: Base font size
- `base_family`: Base font family
- `...`: Passed to `ggplot::theme()`

Value

An object as returned by `ggplot2::theme()`

See Also

- `ggplot2::theme()`

Examples

```r
library(ggplot2)
mtcars$cyl <- factor(mtcars$cyl)
ggplot(mtcars, aes(y=mpg, x=disp, color=cyl)) +
  geom_point() + theme_karl()
```
triarrow  

Plot an arrow within a Holmans triangle

Description

Plot an arrow within a Holmans triangle (an equilateral triangle used to depict trinomial distributions).

Usage

triarrow(x, ...)

Arguments

x  
A matrix with three rows and two columns, each column being a trinomial distribution. An arrow between the two points is plotted.

...  
Passed to graphics::arrows().

Details

Plot of an equilateral triangle, in order to depict trinomial distributions. A trinomial distribution (that is, a trio of non-negative numbers that add to 1) is equated to a point in the triangle through the distances to the three sides. This makes use of the fact that for any point in an equilateral triangle, the sum of the distances to the three sides is constant. First use triplot() to first plot the equilateral triangle.

Value

The (x,y) coordinates of the endpoints of the arrows plotted.

See Also

triplot(), tripoints(), trilines(), tritext()

Examples

triplot()
x <- cbind(c(0.9, 0.05, 0.05), c(0.8, 0.1, 0.1), c(0.1, 0.9, 0), c(0, 0.9, 0.1))
tripoints(x, lwd=2, col=c("black","blue","red","green"), pch=16)
trilines(x, lwd=2, col="orange")
y <- cbind(c(0.05, 0.05, 0.9), c(0.25, 0.25, 0.5))
triarrow(y, col="blue", lwd=2, len=0.1)
trilines  

Plot lines within a Holmans triangle

Description

Plot lines within a Holmans triangle (an equilateral triangle used to depict trinomial distributions).

Usage

trilines(x, ...)

Arguments

x A matrix with three rows, each column being a trinomial distribution. Lines between these points are plotted.

... Passed to graphics::lines().

Details

Plot of an equilateral triangle, in order to depict trinomial distributions. A trinomial distribution (that is, a trio of non-negative numbers that add to 1) is equated to a point in the triangle through the distances to the three sides. This makes use of the fact that for any point in an equilateral triangle, the sum of the distances to the three sides is constant. First use triplot() to first plot the equilateral triangle.

Value

The (x,y) coordinates of the endpoints of the lines plotted.

See Also

triplot(), tripoints(), triarrow(), tritext()

Examples

triplot()
x <- cbind(c(0.9, 0.05, 0.05), c(0.8, 0.1, 0.1), c(0.1, 0.9, 0), c(0, 0.9, 0.1))
triplots(x, lwd=2, col=c("black", "blue", "red", "green"), pch=16)
trilines(x, lwd=2, col="orange")
y <- cbind(c(0.05, 0.05, 0.9), c(0.25, 0.25, 0.5))
triarrow(y, col="blue", lwd=2, len=0.1)
triplot

Plot Holmans triangle

Description

Plot Holmans triangle (an equilateral triangle used to depict trinomial distributions).

Usage

triplot(labels = c("(1,0,0)", "(0,1,0)", "(0,0,1)"), col = "black",
lwd = 2, bgcolor = "gray90", ...)

Arguments

labels Labels for the three corners (lower-right, top, lower-left).
col Color of edges of triangle
lwd Line width for edges of triangle
bgcolor Background color for triangle
... Passed to graphics::plot().

Details

Plot of an equilateral triangle, in order to depict trinomial distributions. A trinomial distribution
(that is, a trio of non-negative numbers that add to 1) is equated to a point in the triangle through the
distances to the three sides. This makes use of the fact that for any point in an equilateral triangle,
the sum of the distances to the three sides is constant. The triplot function creates an empty
triangle for use with the related functions tripoints(), trilines(), triarrow().

Value

The (x,y) coordinates of the points plotted, if any.

See Also

tripoints(), trilines(), triarrow(), tritext()

Examples

triplot()
x <- cbind(c(0.9, 0.05, 0.05), c(0.8, 0.1, 0.1), c(0.1, 0.9, 0), c(0, 0.9, 0.1))
tripoints(x, lwd=2, col=c("black","blue","red","green"), pch=16)
trilines(x, lwd=2, col="orange")
y <- cbind(c(0.05, 0.05, 0.9), c(0.25, 0.25, 0.5))
triarrow(y, col="blue", lwd=2, len=0.1)
Description

Plot points within a Holmans triangle (an equilateral triangle used to depict trinomial distributions).

Usage

```r
tripoints(x, ...)
```

Arguments

- `x` A matrix with three rows, each column being a trinomial distribution.
- `...` Passed to `graphics::points()`.

Details

Plot of an equilateral triangle, in order to depict trinomial distributions. A trinomial distribution (that is, a trio of non-negative numbers that add to 1) is equated to a point in the triangle through the distances to the three sides. This makes use of the fact that for any point in an equilateral triangle, the sum of the distances to the three sides is constant. First use `triplot()` to first plot the equilateral triangle.

Value

The (x,y) coordinates of the points plotted.

See Also

`triplot()`, `trilines()`, `triarrow()`, `tritext()`

Examples

```r
triplot()
x <- cbind(c(0.9, 0.05, 0.05), c(0.8, 0.1, 0.1), c(0.1, 0.9, 0), c(0, 0.9, 0.1))
tripoints(x, lwd=2, col=c("black","blue","red","green"), pch=16)
trilines(x, lwd=2, col="orange")
y <- cbind(c(0.05, 0.05, 0.9), c(0.25, 0.25, 0.5))
triarrow(y, col="blue", lwd=2, len=0.1)
```
tritext  

*Plot text within a Holmans triangle*

**Description**
Plot text within a Holmans triangle (an equilateral triangle used to depict trinomial distributions).

**Usage**
tritext(x, labels, ...)

**Arguments**
- **x**  
  A matrix with three rows, each column being a trinomial distribution.
- **labels**  
  A vector of character strings, with length equal to the number of columns of x.
- **...**  
  Passed to `graphics::text()`.

**Details**
Plot of an equilateral triangle, in order to depict trinomial distributions. A trinomial distribution (that is, a trio of non-negative numbers that add to 1) is equated to a point in the triangle through the distances to the three sides. This makes use of the fact that for any point in an equilateral triangle, the sum of the distances to the three sides is constant. First use `triplot()` to first plot the equilateral triangle.

**Value**
Text is plotted at the (x,y) coordinates of the points.

**See Also**
`triplot()`, `trilines()`, `triarrow()`, `tripoints()`

**Examples**
```r
triplot()
x <- cbind(c(0.25, 0.5, 0.25), c(1/3, 1/3, 1/3))
tripoints(x, lwd=2, pch=21, bg="lightblue")
xp <- x + c(0.02, 0, -0.02)
tritext(xp, c("(1/4,1/2,1/4)", "(1/3,1/3,1/3)"), adj=c(0, 0.5))
```
vec2string

Turn a vector into a single character string

Description

Turn a vector into a single character string with the items separated by commas and an "and".

Usage

vec2string(x)

Arguments

x

A vector

Examples

vec2string(letters[1:2])
vec2string(letters[1:4])

venn

Plot to-scale Venn diagram

Description

Plot a Venn diagram (with two groups), to scale, either with circles or with squares.

Usage

venn(setA = 50, setB = 50, both = 25, method = c("circle", 
"square"), labels = c("A", "B"), col = c("blue", "red"))

Arguments

setA

Total area of set A.

setB

Total area of set B.

both

Area of intersection of sets A and B.

method

Indicates whether to plot circles or squares.

labels

Labels for the two sets. (NULL for no labels.)

col

Colors of the two sets.

Details

Plots a to-scale Venn diagram with two sets, so that the relative areas of the two sets and their intersection are exact.
winsorize

Value

None.

Examples

venn(setA=86, setB=1622, both=10)
venn(setA=86, setB=1622, both=10, method="square")

---

winsorize  Winsorize a vector

Description

For a numeric vector, move values below and above the q and 1-q quantiles to those quantiles.

Usage

winsorize(x, q = 0.006)

Arguments

x      Numeric vector
q      Lower quantile to use

Value

A vector like the input x, but with extreme values moved in to the q and 1-q quantiles.

Examples

x <- sample(c(1:10, rep(NA, 10), 21:30))
winsorize(x, 0.2)
**xlimlabel**

*Calulate horizontal limit in user coordinates for adding labels*

**Description**

Calculates the x-axis limits when adding (long) labels to a plot

**Usage**

`xlimlabel(x, xlabels, pos = 4, offset = 0.5, ...)`

**Arguments**

- `x` numeric vector of horizontal coordinates
- `xlabels` character vector, specifying text to be written
- `pos` position specifier for text; values of 1, 2, 3, and 4, respectively, indicate positions below, to the left of, above, and to the right of the coordinates
- `offset` offset of the label from the coordinate in fractions of a character width
- `...` Additional par arguments

**Details**

See `graphics::text()` for details on `pos` and `offset`.

**Value**

Minimum and maximum x-axis limits for adding horizontal text

**Author(s)**

Aimee Teo Broman

**See Also**

`graphics::text()`

**Examples**

```r
x <- runif(15, -1, 1)*10
xlabs <- sapply(sample(1:20, 15, replace=TRUE),
                function(a) paste(LETTERS[1:a], collapse=''))
par(mfrow=c(2,1), las=1)
## Labels to the left ##
xlims <- xlimlabel(x, xlabs, pos=2)
plot(x, 1:length(x), xlim=xlims, ylab="Index")
text(x, 1:length(x), xlabs, pos=2)
## Labels to the right ##
xlims <- xlimlabel(x, xlabs, pos=4, cex=0.7)
```
\begin{verbatim}
plot(x, 1:length(x), xlim=xlims, ylab="Index")
text(x, 1:length(x), xlabs, pos=4, cex=0.7)
\end{verbatim}

\begin{verbatim}
\%nin\%
\end{verbatim}

\section*{Value matching}

\subsection*{Description}
\%\texttt{in}\% returns logical vector indicating values that do not have a match. \%\texttt{win}\% returns a vector of the values that have a match. \%\texttt{winn}\% returns a vector of the values that do not have a match.

\subsection*{Usage}
\begin{verbatim}
x \%\texttt{in}\% table
x \%\texttt{win}\% table
x \%\texttt{winn}\% table
\end{verbatim}

\subsection*{Arguments}
\begin{verbatim}
x Vector of values to be matched.
table Vector of values to be matched against.
\end{verbatim}

\subsection*{Value}
\%\texttt{in}\% returns a logical vector of the same length of \texttt{x}, indicating which values are not in \texttt{table}.
\%\texttt{win}\% returns a sub-vector of \texttt{x} with the values that were found in \texttt{table}.
\%\texttt{winn}\% returns a sub-vector of \texttt{x} with the values that were not found in \texttt{table}.

\subsection*{See Also}
\texttt{base::match()}

\subsection*{Examples}
\begin{verbatim}
vals <- c("a", "xa", "b")
vals %\texttt{in}\% letters
vals %\texttt{win}\% letters
vals %\texttt{winn}\% letters
\end{verbatim}
Index

*Topic **IO**
  openfile, 33
*Topic **algebra**
  qr2, 40
*Topic **character**
  paste., 35
  paste00, 35
*Topic **color**
  colwalpha, 10
  revgray, 42
  revrainbow, 42
*Topic **datagen**
  rmvn, 43
*Topic **datasets**
  numbers, 32
*Topic **data**
  cf, 7
*Topic **documentation**
  h, 21
*Topic **graphics**
  ciplot, 9
  dotplot, 14
  grayplot, 19
  grayplot_na, 20
  histlines, 23
  manyboxplot, 27
*Topic **hplot**
  arrowlocator, 4
  excel_fig, 15
  mypairs, 29
  qpline2, 39
  triarrow, 52
  trillines, 53
  triplot, 54
  tripoints, 55
  tritext, 56
  venn, 57
*Topic **htest**
  chisq, 8
  fisher, 17
  paired.perm.test, 34
  perm.test, 36
*Topic **manip**
  convert2hex, 12
  hex2dec, 22
*Topic **math**
  simp, 46
*Topic **print**
  bromanversion, 7
*Topic **univar**
  quantileSE, 41
  runningmean, 44
  runningratio, 45
*Topic **utilities**
  attrnames, 5
  brocolors, 6
  crayons, 12
  done, 13
  errors2pushbullet, 15
  exit, 16
  kbdate, 25
  lenuniq, 26
  make, 26
  myround, 29
  normalize, 30
  note, 31
  objectsizes, 32
  setRNGparallel, 46
  stop_sending_errors, 48
  winsorize, 58
  %win% (%nin%), 60
  %wnin% (%nin%), 60
  %nin%, 60

  add_commas, 3
  align_vectors, 4
  arrowlocator, 4
  attrnames, 5
base::cat(), 16
base::chol(), 43
base::date(), 25
base::jitter(), 24, 25
base::match(), 60
base::objects(), 33
base::paste(), 35, 36
base::paste0(), 35, 36
base::qr(), 40
base::rev(), 42, 43
base::round(), 30
base::sprintf(), 30
base::switch(), 50
base::Sys.time(), 25
base::system(), 27
brocolors, 6
brocolors(), 13, 38
bromanversion, 7
cf, 7
chisq(), 8
chisq(), 18
ciplot, 9
colwalpha, 10
compare_rows, 11
convert2hex, 12
crayons, 12
dec2hex (convert2hex), 12
dec2hex(), 23
devtools::as.package(), 27
devtools::load_all(), 27
done, 13
done(), 15, 31, 39
dotplot, 14
dotplot(), 10, 21, 25
errors2pushbullet, 15
errors2pushbullet(), 13, 31, 39, 48
eexcel_fig, 15
exit, 16
fac2num, 17
fisher, 17
fisher(), 9
get_precision, 18
gggplot2::theme(), 51
 graphics::hist(), 23
 graphics::lines(), 23, 53
 graphics::locator(), 4, 5
 graphics::pairs(), 29
 graphics::par(), 16, 20, 38
 graphics::plot(), 19–21, 29, 54
 graphics::points(), 19–21, 55
 graphics::rect(), 19–21
 graphics::text(), 50, 56, 59
 grayplot, 19
 grayplot(), 9, 10, 14, 21, 28
 grayplot_na, 20
 grDevices::gray(), 42
 grDevices::rainbow(), 42, 43
 h, 21
 hex2dec, 22
 hex2dec(), 12
 histlines, 23
 jiggle, 24
 jiggle(), 14
 karl_theme (theme_karl), 51
 kbddate, 25
 lenuniq, 26
 make, 26
 manyboxplot, 27
 maxabs, 28
 mypairs, 29
 myround, 29
 normalize, 30
 note, 31
 note(), 13, 15, 39
 Numbers (numbers), 32
 numbers, 32
 objectsizes, 32
 openfile, 33
 paired.perm.test, 34
 paired.perm.test(), 37
 paste, 35
 paste(), 36
 paste () (paste00), 35
 paste(), 35
 paste.0 (paste00), 35
INDEX

paste0(), 35
paste0.(paste00), 35
paste0(), 35
paste00, 35
paste0(){, 35
perm.test, 36
perm.test(), 34
pick_more_precise, 37
plot_crayons, 38
plot_crayons(), 6, 13
pushbullet_devices, 39
pushbullet_devices(), 13, 15, 31, 48

qline(), 39
qr2, 40
quantileSE, 41

revgray, 42
revrainbow, 42
rmvn, 43
RPushbullet::pbPost(), 13, 15, 31
runningmean, 44
runningmean(), 45
runningratio, 45
runningratio(), 44

setRNGparallel, 46
simp, 46
spell_out, 47
stats::chisq.test(), 9, 18
stats::density(), 41
stats::fisher.test(), 9, 18
stats::integrate(), 47
stats::qline(), 39, 40
stats::qqnorm(), 39
stats::qqplot(), 39, 40
stats::quantile(), 39, 41
stats::rnorm(), 44
stats::t.test(), 34, 37
stopSendingErrors, 48
stopSendingErrors(), 15
strwidth2lines, 48
strwidth2xlim, 49
switchv, 50

theme_karl, 51
trap(simp), 46
triarrow, 52
triarrow(), 53–56

trilines, 53
trilines(), 52, 54–56
triplot, 54
triplot(), 52, 53, 55, 56
tripoints, 55
tripoints(), 52–54, 56
tritext, 56
tritext(), 52–55

unsetRNGparallel(setRNGparallel), 46
utils::help(), 22
utils::help.start(), 22
utils::object.size(), 32, 33

vec2string, 57
venn, 57

winsorize, 58

xlimlabel, 59