

Package ‘gtools’

January 27, 2015

Title Various R programming tools
Description Various R programming tools
Version 3.4.1
Date 2014-05-27
Author Gregory R. Warnes, Ben Bolker, and Thomas Lumley
Maintainer Gregory R. Warnes <greg@warnes.net>
License LGPL-2.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-05-28 09:52:04

R topics documented:

| | |
|-----------------------------|----|
| ASCIIfy | 2 |
| ask | 3 |
| binsearch | 4 |
| checkRVersion | 6 |
| combinations | 7 |
| defmacro | 8 |
| ELISA | 11 |
| foldchange | 12 |
| getDependencies | 13 |
| gtools-defunct | 14 |
| gtools-deprecated | 15 |
| invalid | 15 |
| keywords | 16 |
| lastAdd | 17 |
| logit | 18 |
| mixedsort | 20 |
| odd | 21 |
| permute | 22 |
| quantcut | 23 |
| rdirichlet | 24 |

| | |
|-------------------------|----|
| running | 25 |
| scat | 27 |
| setTCPNoDelay | 28 |
| smartbind | 29 |
| stars.pval | 30 |

| | |
|--------------|-----------|
| Index | 32 |
|--------------|-----------|

| | |
|---------|------------------------------------|
| ASCIIfy | <i>Convert Characters to ASCII</i> |
|---------|------------------------------------|

Description

Convert character vector to ASCII, replacing non-ASCII characters with single-byte (`'\x00'`) or two-byte (`'\u0000'`) codes.

Usage

```
ASCIIfy(x, bytes = 2, fallback = "?")
```

Arguments

| | |
|-----------------------|---|
| <code>x</code> | a character vector, possibly containing non-ASCII characters. |
| <code>bytes</code> | either 1 or 2, for single-byte (<code>'\x00'</code>) or two-byte (<code>'\u0000'</code>) codes. |
| <code>fallback</code> | an output character to use, when input characters cannot be converted. |

Value

A character vector like `x`, except non-ASCII characters have been replaced with `'\x00'` or `'\u0000'` codes.

Note

To render single backslashes, use these or similar techniques:

```
write(ASCIIfy(x), "file.txt")
cat(paste(ASCIIfy(x), collapse="\n"), "\n", sep="")
```

The resulting strings are plain ASCII and can be used in R functions and datasets to improve package portability.

Author(s)

Arni Magnusson <arnima@hafro.is>

See Also

[showNonASCII](#) identifies non-ASCII characters in a character vector.

Examples

```
cities <- c("São Paulo", "Reykjavík")
print(cities)
ASCIIify(cities, 1)
ASCIIify(cities, 2)

athens <- "Ἰθάκη"
print(athens)
ASCIIify(athens)
```

ask

Display a prompt and collect the user's response

Description

Display a prompt and collect the user's response

Usage

```
ask(msg = "Press <RETURN> to continue: ")
```

Arguments

msg Character vector providing the message to be displayed

Details

The prompt message will be displayed, and then `readLines` is used to collect a single input value (possibly empty), which is then returned.

Value

A character scalar containing the input provided by the user.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[readLines](#), [scan](#)

Examples

```
# use default prompt
ask()

silly <- function()
{
  age <- ask("How old aroe you? ")
  age <- as.numeric(age)
  cat("In 10 years you will be", age+10, "years old!\n")
}
```

binsearch

*Binary Search***Description**

Search within a specified range to locate an integer parameter which results in the the specified monotonic function obtaining a given value.

Usage

```
binsearch(fun, range, ..., target = 0, lower = ceiling(min(range)),
          upper = floor(max(range)), maxiter = 100, showiter = FALSE)
```

Arguments

| | |
|----------|---|
| fun | Monotonic function over which the search will be performed. |
| range | 2-element vector giving the range for the search. |
| ... | Additional parameters to the function fun. |
| target | Target value for fun. Defaults to 0. |
| lower | Lower limit of search range. Defaults to min(range). |
| upper | Upper limit of search range. Defaults to max(range). |
| maxiter | Maximum number of search iterations. Defaults to 100. |
| showiter | Boolean flag indicating whether the algorithm state should be printed at each iteration. Defaults to FALSE. |

Details

This function implements an extension to the standard binary search algorithm for searching a sorted list. The algorithm has been extended to cope with cases where an exact match is not possible, to detect whether that the function may be monotonic increasing or decreasing and act appropriately, and to detect when the target value is outside the specified range.

The algorithm initializes two variable lo and high to the extremes values of range. It then generates a new value center halfway between lo and hi. If the value of fun at center exceeds target, it

becomes the new value for lo, otherwise it becomes the new value for hi. This process is iterated until lo and hi are adjacent. If the function at one or the other equals the target, this value is returned, otherwise lo, hi, and the function value at both are returned.

Note that when the specified target value falls between integers, the *two* closest values are returned. If the specified target falls outside of the specified range, the closest endpoint of the range will be returned, and an warning message will be generated. If the maximum number of iterations was reached, the endpoints of the current subset of the range under consideration will be returned.

Value

A list containing:

| | |
|---------|---|
| call | How the function was called. |
| numiter | The number of iterations performed |
| flag | One of the strings, "Found", "Between Elements", "Maximum number of iterations reached", "Reached lower boundary", or "Reached upper boundary." |
| where | One or two values indicating where the search terminated. |
| value | Value of the function fun at the values of where. |

Note

This function often returns two values for where and value. Be sure to check the flag parameter to see what these values mean.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[optim](#), [optimize](#), [uniroot](#)

Examples

```
### Toy examples

# search for x=10
binsearch( function(x) x-10, range=c(0,20) )

# search for x=10.1
binsearch( function(x) x-10.1, range=c(0,20) )

### Classical toy example

# binary search for the index of 'M' among the sorted letters
fun <- function(X) ifelse(LETTERS[X] > 'M', 1,
                          ifelse(LETTERS[X] < 'M', -1, 0) )

binsearch( fun, range=1:26 )
```

```
# returns $where=13
LETTERS[13]

### Substantive example, from genetics
## Not run:
library(genetics)
# Determine the necessary sample size to detect all alleles with
# frequency 0.07 or greater with probability 0.95.
power.fun <- function(N) 1 - gregorius(N=N, freq=0.07)$missprob

binsearch( power.fun, range=c(0,100), target=0.95 )

# equivalent to
gregorius( freq=0.07, missprob=0.05)

## End(Not run)
```

| | |
|---------------|---|
| checkRVersion | <i>Check if a newer version of R is available</i> |
|---------------|---|

Description

Check if a newer version of R is available

Usage

```
checkRVersion(quiet = FALSE)
```

Arguments

`quiet` Logical indicating whether printed output should be suppressed.

Details

This function accesses the R web site to discover the latest released version of R. It then compares this version to the running version. If the running version is the same as the latest version, it prints the message, "The latest version of R is installed:" followed by the version number, and returns NULL. If the running version is older than the current version, it displays the message, "A newer version of R is now available:" followed by the corresponding version number, and returns the version number.

If `quiet=TRUE`, no printing is performed.

Value

Either the version number of the latest version of R, if the running version is less than the latest version, or NULL.

Note

This function utilizes the internet to access the R project web site. If internet access is unavailable, the function will fail.

Author(s)

Gregory R. Warnes <gregory.warnes@rochester.edu>

See Also

[R.Version](#)

Examples

```
checkRVersion()

ver <- checkRVersion()
print(ver)
```

| | |
|--------------|---|
| combinations | <i>Enumerate the Combinations or Permutations of the Elements of a Vector</i> |
|--------------|---|

Description

combinations enumerates the possible combinations of a specified size from the elements of a vector. permutations enumerates the possible permutations.

Usage

```
combinations(n, r, v=1:n, set=TRUE, repeats.allowed=FALSE)
permutations(n, r, v=1:n, set=TRUE, repeats.allowed=FALSE)
```

Arguments

| | |
|-----------------|---|
| n | Size of the source vector |
| r | Size of the target vectors |
| v | Source vector. Defaults to 1:n |
| set | Logical flag indicating whether duplicates should be removed from the source vector v. Defaults to TRUE. |
| repeats.allowed | Logical flag indicating whether the constructed vectors may include duplicated values. Defaults to FALSE. |

Details

Caution: The number of combinations and permutations increases rapidly with n and $r!$.

To use values of n above about 45, you will need to increase R's recursion limit. See the `expression` argument to the `options` command for details on how to do this.

Value

Returns a matrix where each row contains a vector of length r .

Author(s)

Original versions by Bill Venables <Bill.Venables@cmis.csiro.au>. Extended to handle `repeats.allowed` by Gregory R. Warnes <greg@warnes.net>.

References

Venables, Bill. "Programmers Note", R-News, Vol 1/1, Jan. 2001. <http://cran.r-project.org/doc/Rnews>

See Also

[choose](#), [options](#)

Examples

```
combinations(3,2,letters[1:3])
combinations(3,2,letters[1:3],repeats=TRUE)

permutations(3,2,letters[1:3])
permutations(3,2,letters[1:3],repeats=TRUE)

# To use large 'n', you need to change the default recursion limit
options(expressions=1e5)
cmat <- combinations(300,2)
dim(cmat) # 44850 by 2
```

defmacro

Define a macro

Description

`defmacro` define a macro that uses R expression replacement

`strmacro` define a macro that uses string replacement

Usage

```
defmacro(..., expr)
strmacro(..., expr, stexpr)
```


Arguments

| | |
|---------|--|
| ... | macro argument list |
| expr | R expression defining the macro body |
| strexpr | character string defining the macro body |

Details

defmacro and strmacro create a macro from the expression given in expr, with formal arguments given by the other elements of the argument list.

A macro is similar to a function definition except for handling of formal arguments. In a function, formal arguments are simply variables that contains the result of evaluating the expressions provided to the function call. In contrast, macros actually modify the macro body by *replacing* each formal argument by the expression (defmacro) or string (strmacro) provided to the macro call.

For defmacro, the special argument name DOTS will be replaced by ... in the formal argument list of the macro so that ... in the body of the expression can be used to obtain any additional arguments passed to the macro. For strmacro you can mimic this behavior providing a DOTS="" argument. This is illustrated by the last example below.

Macros are often useful for creating new functions during code execution.

Value

A macro function.

Note

Note that because [the defmacro code] works on the parsed expression, not on a text string, defmacro avoids some of the problems of traditional string substitution macros such as strmacro and the C preprocessor macros. For example, in

```
mul <- defmacro(a, b, expr={a*b})
```

a C programmer might expect mul(i, j + k) to expand (incorrectly) to i*j + k. In fact it expands correctly, to the equivalent of i*(j + k).

For a discussion of the differences between functions and macros, please Thomas Lumley's R-News article (reference below).

Author(s)

Thomas Lumley wrote defmacro. Gregory R. Warnes <greg@warnes.net> enhanced it and created strmacro.

References

The original defmacro code was directly taken from:

Lumley T. "Programmer's Niche: Macros in R", R News, 2001, Vol 1, No. 3, pp 11–13, <http://CRAN.R-project.org/doc/Rnews/>

See Also

[function substitute](#), [eval](#), [parse](#), [source](#), [parse](#),

Examples

```
####
# macro for replacing a specified missing value indicator with NA
# within a dataframe
###
setNA <- defmacro(df, var, values,
                  expr={
                    df$var[df$var %in% values] <- NA
                  })

# create example data using 999 as a missing value indicator
d <- data.frame(
  Grp=c("Trt", "Ctl", "Ctl", "Trt", "Ctl", "Ctl", "Trt", "Ctl", "Trt", "Ctl"),
  V1=c(1, 2, 3, 4, 5, 6, 999, 8, 9, 10),
  V2=c(1, 1, 1, 1, 1, 2, 999, 2, 999, 999)
)

d

# Try it out
setNA(d, V1, 999)
setNA(d, V2, 999)
d

###
# Expression macro
###
plot.d <- defmacro( df, var, DOTS, col="red", title="", expr=
  plot( df$var ~ df$Grp, type="b", col=col, main=title, ... )
)

plot.d( d, V1)
plot.d( d, V1, col="blue" )
plot.d( d, V1, lwd=4) # use optional 'DOTS' argument

###
# String macro (note the quoted text in the calls below)
#
# This style of macro can be useful when you are reading
# function arguments from a text file
###
plot.s <- strmacro( DF, VAR, COL="'red'", TITLE="'", DOTS="", expr=
  plot( DF$VAR ~ DF$Grp, type="b", col=COL, main=TITLE, DOTS)
)

plot.s( "d", "V1")
plot.s( DF="d", VAR="V1", COL="'blue'" )
plot.s( "d", "V1", DOTS='lwd=4') # use optional 'DOTS' argument
```

```
#####  
# Create a macro that defines new functions  
#####  
plot.sf <- defmacro(type='b', col='black',  
                    title=deparse(substitute(x)), DOTS, expr=  
  function(x,y) plot( x,y, type=type, col=col, main=title, ...)  
)  
  
plot.red <- plot.sf(col='red',title='Red is more Fun!')  
plot.blue <- plot.sf(col='blue',title="Blue is Best!", lty=2)  
  
plot.red(1:100,rnorm(100))  
plot.blue(1:100,rnorm(100))
```

ELISA

Data from an ELISA assay

Description

Observed signals and (for some observations) nominal concentrations for samples that were aliquoted to multiple assay plates, which were read multiple times on multiple days.

Usage

```
data(ELISA)
```

Format

a data frame with the following columns:

- PlateDayfactor. Specifies one of four physically distinct 96 well plates
- Readfactor. The signal was read 3 times for each plate.
- Descriptioncharacter. Indicates contents of sample.
- Concentrationnumeric. Nominal concentration of standards (NA for all other samples).
- Signalnumeric. Assay signal. Specifically, optical density (a colorimetric assay).

Source

Anonymized data.

| | |
|------------|--|
| foldchange | <i>Compute fold-change or convert between log-ratio and fold-change.</i> |
|------------|--|

Description

foldchange computes the fold change for two sets of values. logratio2foldchange converts values from log-ratios to fold changes. foldchange2logratio does the reverse.

Usage

```
foldchange(num,denom)
logratio2foldchange(logratio, base=2)
foldchange2logratio(foldchange, base=2)
```

Arguments

| | |
|------------|-------------------------------------|
| num,denom | vector/matrix of numeric values |
| logratio | vector/matrix of log-ratio values |
| foldchange | vector/matrix of fold-change values |
| base | Exponential base for the log-ratio. |

Details

Fold changes are commonly used in the biological sciences as a mechanism for comparing the relative size of two measurements. They are computed as: $\frac{num}{denom}$ if $num > denom$, and as $\frac{-denom}{num}$ otherwise.

Fold-changes have the advantage of ease of interpretation and symmetry about $num = denom$, but suffer from a discontinuity between -1 and 1, which can cause significant problems when performing data analysis. Consequently statisticians prefer to use log-ratios.

Value

A vector or matrix of the same dimensions as the input containing the converted values.

Author(s)

Gregory R. Warnes <greg@warnes.net>

Examples

```
a <- 1:21
b <- 21:1

f <- foldchange(a,b)

cbind(a,b,f)
```

| | |
|-----------------|---------------------------------|
| getDependencies | <i>Get package dependencies</i> |
|-----------------|---------------------------------|

Description

Get package dependencies

Usage

```
getDependencies(pkgs,
                dependencies = c("Depends", "Imports", "LinkingTo"),
                installed=TRUE,
                available=TRUE,
                base=FALSE,
                recommended=FALSE)
```

Arguments

| | |
|--------------|---|
| pkgs | character vector of package names |
| dependencies | character vector of dependency types to include. Choices are "Depends", "Imports", "LinkingTo", "Suggests", and "Enhances". Defaults to c("Depends", "Imports", "LinkingTo"). |
| installed | Logical indicating whether to pull dependency information from installed packages. Defaults to TRUE. |
| available | Logical indicating whether to pull dependency information from available packages. Defaults to TRUE. |
| base | Logical indicating whether to include dependencies on base packages that are included in the R installation. Defaults to FALSE. |
| recommended | Logical indicating whether to include dependencies on recommended packages that are included in the R installation. Defaults to FALSE. |

Details

This function recursively constructs the list of dependencies for the packages given by `pkgs`. By default, the dependency information is extracted from both installed and available packages. As a consequence, it works both for local and CRAN packages.

Value

A character vector of package names.

Note

If `available=TRUE` R will attempt to access the currently selected CRAN repository, prompting for one if necessary.

Author(s)

Gregory R. Warnes emailgreg@warnes.net based on the non exported `utils::getDependencies` and `utils:::clean_up_dependencies2`.

See Also

[installed.packages](#), [available.packages](#)

Examples

```
## A locally installed package
getDependencies("MASS", installed=TRUE, available=FALSE)

## Not run:
## A package on CRAN
getDependencies("gregmisc", installed=FALSE, available=TRUE)

## End(Not run)

## Show base and recommended dependencies
getDependencies("MASS", available=FALSE, base=TRUE, recommended=TRUE)

## Not run:
## Download the set of packages necessary to support a local package
deps <- getDependencies("MyLocalPackage", available=FALSE)
download.packages(deps, destdir=". /R_Packages")

## End(Not run)
```

gtools-defunct

Defunct Functions in package gtools

Description

The functions or variables listed here are no longer part of package gtools.

Details

- `assert` is a defunct synonym for [stopifnot](#).
- `addLast` has been replaced by `lastAdd`, which has the same purpose but applied using different syntax.
- `capture` and `capture.output` have been removed in favor of `capture.output` from the `utils` package.

See Also

[Defunct](#), [stopifnot](#), [lastAdd](#), [capture.output](#)

gtools-deprecated *Deprecated Functions in the gtools package*

Description

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

Details

gtools currently contains no deprecated functions.

See Also

[Deprecated](#)

invalid *Test if a value is missing, empty, or contains only NA or NULL values*

Description

Test if a value is missing, empty, or contains only NA or NULL values.

Usage

```
invalid(x)
```

Arguments

x value to be tested

Value

Logical value.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[missing](#), [is.na](#), [is.null](#)

Examples

```
invalid(NA)
invalid()
invalid(c(NA,NA,NULL,NA))

invalid(list(a=1,b=NULL))

# example use in a function
myplot <- function(x,y) {
  if(invalid(y)) {
    y <- x
    x <- 1:length(y)
  }
  plot(x,y)
}
myplot(1:10)
myplot(1:10,NA)
```

keywords

List valid keywords for R man pages

Description

List valid keywords for R man pages

Usage

```
keywords(topic)
```

Arguments

topic object or man page topic

Details

If topic is provided, return a list of the keywords associated with topic. Otherwise, display the list of valid R keywords from the R doc/KEYWORDS file.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[help](#)

Examples

```
## Show all valid R keywords
keywords()

## Show keywords associated with the 'merge' function
keywords(merge)
keywords("merge")
```

| | |
|---------|--|
| lastAdd | <i>Non-destructively construct a .Last function to be executed when R exits.</i> |
|---------|--|

Description

Non-destructively construct a .Last function to be executed when R exits.

Usage

```
lastAdd(fun)
```

Arguments

| | |
|-----|------------------------|
| fun | Function to be called. |
|-----|------------------------|

Details

lastAdd constructs a new function which can be used to replace the existing definition of .Last, which will be executed when R terminates normally.

If a .Last function already exists in the global environment, the original definition is stored in a private environment, and the new function is defined to call the function fun and then to call the previous (stored) definition of .Last.

If no .Last function exists in the global environment, lastAdd simply returns the function fun.

Value

A new function to be used for .Last.

Note

This function replaces the (now defunct) addLast function.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[.Last](#)

Examples

```
## Print a couple of cute messages when R exits.
helloWorld <- function() cat("\nHello World!\n")
byeWorld <- function() cat("\nGoodbye World!\n")

.Last <- lastAdd(byeWorld)
.Last <- lastAdd(helloWorld)

## Not run:
q("no")

## Should yield:
##
## Save workspace image? [y/n/c]: n
##
## Hello World!
##
## Goodbye World!
##
## Process R finished at Tue Nov 22 10:28:55 2005

## End(Not run)

## Unix-flavour example: send Rplots.ps to printer on exit.
myLast <- function()
{
  cat("Now sending PostScript graphics to the printer:\n")
  system("lpr Rplots.ps")
  cat("bye bye...\n")
}
.Last <- lastAdd(myLast)

## Not run:
quit("yes")

## Should yield:
##
## Now sending PostScript graphics to the printer:
## lpr: job 1341 queued
## bye bye...
##
## Process R finished at Tue Nov 22 10:28:55 2005

## End(Not run)
```

Description

Compute generalized logit and generalized inverse logit functions.

Usage

```
logit(x, min = 0, max = 1)  
inv.logit(x, min = 0, max = 1)
```

Arguments

| | |
|-----|-----------------------------|
| x | value(s) to be transformed |
| min | Lower end of logit interval |
| max | Upper end of logit interval |

Details

The generalized logit function takes values on [min, max] and transforms them to span [-Inf,Inf] it is defined as:

$$y = \log\left(\frac{p}{1-p}\right)$$

where

$$p = \frac{(x - \text{min})}{(\text{max} - \text{min})}$$

The generalized inverse logit function provides the inverse transformation:

$$x = p'(\text{max} - \text{min}) + \text{min}$$

where

$$p' = \frac{\exp(y)}{(1 + \exp(y))}$$

Value

Transformed value(s).

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[logit](#)

Examples

```
x <- seq(0,10, by=0.25)
xt <- logit(x, min=0, max=10)
cbind(x,xt)

y <- inv.logit(xt, min=0, max=10)
cbind(x,xt,y)
```

mixedsort

Order or Sort strings with embedded numbers so that the numbers are in the correct order

Description

These functions sort or order character strings containing numbers so that the numbers are numerically sorted rather than sorted by character value. I.e. "Asprin 50mg" will come before "Asprin 100mg". In addition, case of character strings is ignored so that "a", will come before "B" and "C".

Usage

```
mixedsort(x)
```

Arguments

x Character vector to be sorted

Details

I often have character vectors (e.g. factor labels) that contain both text and numeric data, such as compound and dose. This function is useful for sorting these character vectors into a logical order.

It does so by splitting each character vector into a sequence of character and numeric sections, and then sorting along these sections, with numbers being sorted by numeric value (e.g. "50" comes before "100"), followed by characters strings sorted by character value (e.g. "A" comes before "B").

Empty strings are always sorted to the front of the list, and NA values to the end.

Value

mixedorder returns a vector giving the sort order of the input elements. mixedsort returns the sorted vector.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also[sort, order](#)**Examples**

```
# compound & dose labels
Treatment <- c("Control", "Asprin 10mg/day", "Asprin 50mg/day",
              "Asprin 100mg/day", "Acetomycin 100mg/day",
              "Acetomycin 1000mg/day")

# ordinary sort puts the dosages in the wrong order
sort(Treatment)

# but mixedsort does the 'right' thing
mixedsort(Treatment)

# Here is a more complex example
x <- rev(c("AA 0.50 ml", "AA 1.5 ml", "AA 500 ml", "AA 1500 ml",
          "EXP 1", "AA 1e3 ml", "A A A", "1 2 3 A", "NA", NA, "1e2",
          "", "-", "1A", "1 A", "100", "100A", "Inf"))

mixedorder(x)

mixedsort(x)
# notice that plain numbers, including 'Inf' show up before strings.
```

odd

Detect odd/even integers

Description

detect odd/even integers

Usage

```
odd(x)
even(x)
```

Arguments

x vector of integers

Value

Vector of TRUE/FALSE values.

Author(s)

Gregory R. Warnes <greg@warnes.net>

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

```
odd(4)
even(4)
```

```
odd(1:10)
even(1:10)
```

permute

Randomly Permute the Elements of a Vector

Description

Randomly Permute the elements of a vector

Usage

```
permute(x)
```

Arguments

x Vector of items to be permuted

Details

This is simply a wrapper function for [sample](#).

Value

Vector with the original items reordered.

Author(s)

Gregory R. Warnes <greg@warnes.net>

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

```
x <- 1:10
permute(x)
```

| | |
|----------|--|
| quantcut | <i>Create a Factor Variable Using the Quantiles of a Continuous Variable</i> |
|----------|--|

Description

Create a factor variable using the quantiles of a continuous variable.

Usage

```
quantcut(x, q=seq(0,1,by=0.25), na.rm=TRUE, ...)
```

Arguments

| | |
|-------|---|
| x | Continuous variable. |
| q | Vector of quantiles used for creating groups. Defaults to <code>seq(0, 1, by=0.25)</code> . See quantile for details. |
| na.rm | Boolean indicating whether missing values should be removed when computing quantiles. Defaults to TRUE. |
| ... | Optional arguments passed to cut . |

Details

This function uses [quantile](#) to obtain the specified quantiles of `x`, then calls [cut](#) to create a factor variable using the intervals specified by these quantiles.

It properly handles cases where more than one quantile obtains the same value, as in the second example below. Note that in this case, there will be fewer generated factor levels than the specified number of quantile intervals.

Value

Factor variable with one level for each quantile interval given by `q`.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[cut](#), [quantile](#)

Examples

```
## create example data

x <- rnorm(1000)

## cut into quartiles
quartiles <- quantcut( x )
table(quartiles)

## cut into deciles
deciles <- quantcut( x, seq(0,1,by=0.1) )
table(deciles)

## show handling of 'tied' quantiles.
x <- round(x) # discretize to create ties
stem(x)      # display the ties
deciles <- quantcut( x, seq(0,1,by=0.1) )

table(deciles) # note that there are only 5 groups (not 10)
                # due to duplicates
```

rdirichlet

Functions for the Dirichlet Distribution

Description

Functions to compute the density of or generate random deviates from the Dirichlet distribution.

Usage

```
rdirichlet(n, alpha)
ddirichlet(x, alpha)
```

Arguments

| | |
|-------|--|
| x | A vector containing a single random deviate or matrix containing one random deviate per row. |
| n | Number of random vectors to generate. |
| alpha | Vector or (for ddirichlet) matrix containing shape parameters. |

Details

The Dirichlet distribution is the multidimensional generalization of the beta distribution. It is the canonical Bayesian distribution for the parameter estimates of a multinomial distribution.

Value

`ddirichlet` returns a vector containing the Dirichlet density for the corresponding rows of `x`.
`rdirichlet` returns a matrix with `n` rows, each containing a single Dirichlet random deviate.

Author(s)

Code original posted by Ben Bolker to R-News on Fri Dec 15 2000. See <http://www.r-project.org/nocvs/mail/r-help/2000/3865.html>. Ben attributed the code to Ian Wilson <i.wilson@maths.abdn.ac.uk>. Subsequent modifications by Gregory R. Warnes <greg@warnes.net>.

See Also

[dbeta](#), [rbeta](#)

Examples

```
x <- rdirichlet(20, c(1,1,1) )
ddirichlet(x, c(1,1,1) )
```

 running

Apply a Function Over Adjacent Subsets of a Vector

Description

Applies a function over subsets of the vector(s) formed by taking a fixed number of previous points.

Usage

```
running(X, Y=NULL, fun=mean, width=min(length(X), 20),
        allow.fewer=FALSE, pad=FALSE, align=c("right", "center", "left"),
        simplify=TRUE, by, ...)
```

Arguments

| | |
|--------------------------|---|
| <code>X</code> | data vector |
| <code>Y</code> | data vector (optional) |
| <code>fun</code> | Function to apply. Default is <code>mean</code> |
| <code>width</code> | Integer giving the number of vector elements to include in the subsets. Defaults to the lesser of the length of the data and 20 elements. |
| <code>allow.fewer</code> | Boolean indicating whether the function should be computed for subsets with fewer than <code>width</code> points |

| | |
|----------|---|
| pad | Boolean indicating whether the returned results should be 'padded' with NAs corresponding to sets with less than width elements. This only applies when allow.fewer is FALSE. |
| align | One of "right", "center", or "left". This controls the relative location of 'short' subsets with less than width elements: "right" allows short subsets only at the beginning of the sequence so that all of the complete subsets are at the end of the sequence (i.e. 'right aligned'), "left" allows short subsets only at the end of the data so that the complete subsets are 'left aligned', and "center" allows short subsets at both ends of the data so that complete subsets are 'centered'. |
| simplify | Boolean. If FALSE the returned object will be a list containing one element per evaluation. If TRUE, the returned object will be coerced into a vector (if the computation returns a scalar) or a matrix (if the computation returns multiple values). Defaults to FALSE. |
| by | Integer separation between groups. If by=width will give non-overlapping windows. Default is missing, in which case groups will start at each value in the X/Y range. |
| ... | parameters to be passed to fun |

Details

running applies the specified function to a sequential windows on X and (optionally) Y. If Y is specified the function must be bivariate.

Value

List (if simplify==TRUE), vector, or matrix containing the results of applying the function fun to the subsets of X (running) or X and Y.

Note that this function will create a vector or matrix even for objects which are not simplified by sapply.

Author(s)

Gregory R. Warnes <greg@warnes.net>, with contributions by Nitin Jain <nitin.jain@pfizer.com>.

See Also

[wapply](#) to apply a function over an x-y window centered at each x point, [sapply](#), [lapply](#)

Examples

```
# show effect of pad
running(1:20, width=5)
running(1:20, width=5, pad=TRUE)

# show effect of align
running(1:20, width=5, align="left", pad=TRUE)
running(1:20, width=5, align="center", pad=TRUE)
running(1:20, width=5, align="right", pad=TRUE)
```

```

# show effect of simplify
running(1:20, width=5, fun=function(x) x ) # matrix
running(1:20, width=5, fun=function(x) x, simplify=FALSE) # list

# show effect of by
running(1:20, width=5)      # normal
running(1:20, width=5, by=5) # non-overlapping
running(1:20, width=5, by=2) # starting every 2nd

# Use 'pad' to ensure correct length of vector, also show the effect
# of allow.fewer.
par(mfrow=c(2,1))
plot(1:20, running(1:20, width=5, allow.fewer=FALSE, pad=TRUE), type="b")
plot(1:20, running(1:20, width=5, allow.fewer=TRUE, pad=TRUE), type="b")
par(mfrow=c(1,1))

# plot running mean and central 2 standard deviation range
# estimated by *last* 40 observations
dat <- rnorm(500, sd=1 + (1:500)/500 )
plot(dat)
sdfun <- function(x,sign=1) mean(x) + sign * sqrt(var(x))
lines(running(dat, width=51, pad=TRUE, fun=mean), col="blue")
lines(running(dat, width=51, pad=TRUE, fun=sdfun, sign=-1), col="red")
lines(running(dat, width=51, pad=TRUE, fun=sdfun, sign= 1), col="red")

# plot running correlation estimated by last 40 observations (red)
# against the true local correlation (blue)
sd.Y <- seq(0,1,length=500)

X <- rnorm(500, sd=1)
Y <- rnorm(500, sd=sd.Y)

plot(running(X,X+Y,width=20,fun=cor,pad=TRUE),col="red",type="s")

r <- 1 / sqrt(1 + sd.Y^2) # true cor of (X,X+Y)
lines(r,type="l",col="blue")

```

scat

Display debugging text

Description

If `getOption('DEBUG')==TRUE`, write text to `STDOUT` and flush so that the text is immediately displayed. Otherwise, do nothing.

Usage

```
scat(...)
```

Arguments

... Arguments passed to cat

Value

NULL (invisibly)

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[cat](#)

Examples

```
options(DEBUG=NULL) # makee sure DEBUG isn't set
scat("Not displayed")
```

```
options(DEBUG=TRUE)
scat("This will be displayed immediatly (even in R BATCH output \n")
scat("files), provided options()$DEBUG is TRUE.")
```

setTCPNoDelay

Modify the TCP_NODELAY ('de-Nagle') flag for socket objects

Description

Modify the TCP_NODELAY ('de-Nagle') flag for socket objects

Usage

```
setTCPNoDelay(socket, value=TRUE)
```

Arguments

socket A socket connection object
value Logical indicating whether to set (TRUE) or unset (FALSE) the flag

Details

By default, TCP connections wait a small fixed interval before actually sending data, in order to permit small packets to be combined. This algorithm is named after its inventor, John Nagle, and is often referred to as 'Nagling'.

While this reduces network resource utilization in these situations, it imposes a delay on all outgoing message data, which can cause problems in client/server situations.

This function allows this feature to be disabled (de-Nagling, value=TRUE) or enabled (Nagling, value=FALSE) for the specified socket.

Value

The character string "SUCCESS" will be returned invisible if the operation was succesful. On failure, an error will be generated.

Author(s)

Gregory R. Warnes <greg@warnes.net>

References

"Nagle's algorithm" at WhatIS.com http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci754347,00.html

Nagle, John. "Congestion Control in IP/TCP Internetworks", IETF Request for Comments 896, January 1984. <http://www.ietf.org/rfc/rfc0896.txt?number=896>

See Also

[make.socket](#), [socketConnection](#)

Examples

```
## Not run:
s <- make.socket(host='www.r-project.org', port=80)
setTCPNoDelay(s, value=TRUE)

## End(Not run)
```

smartbind

Efficient rbind of data frames, even if the column names don't match

Description

Efficient rbind of data frames, even if the column names don't match

Usage

```
smartbind(..., fill=NA, sep=':', verbose=FALSE)
```

Arguments

| | |
|---------|--|
| ... | Data frames to combine |
| fill | Value to use when 'filling' missing columns. Defaults to NA. |
| sep | Character string used to separate column names when pasting them together. |
| verbose | Logical flag indicating whether to display processing messages. Defaults to FALSE. |

Value

The returned data frame will contain:

| | |
|---------|--|
| columns | all columns present in any provided data frame |
| rows | a set of rows from each provided data frame, with values in columns not present in the given data frame filled with missing (NA) values. |

The data type of columns will be preserved, as long as all data frames with a given column name agree on the data type of that column. If the data frames disagree, the column will be converted into a character strings. The user will need to coerce such character columns into an appropriate type.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[rbind](#), [cbind](#)

Examples

```
df1 <- data.frame(A=1:10, B=LETTERS[1:10], C=rnorm(10) )
df2 <- data.frame(A=11:20, D=rnorm(10), E=letters[1:10] )

# rbind would fail
## Not run:
rbind(df1, df2)
# Error in match.names(clabs, names(xi)) : names do not match previous
# names:
# D, E

## End(Not run)
# but smartbind combines them, appropriately creating NA entries
smartbind(df1, df2)

# specify fill=0 to put 0 into the missing row entries
smartbind(df1, df2, fill=0)
```

stars.pval

Generate significance stars from p-values

Description

Generate significance stars (e.g. '****', '***', '**', '+') from p-values using R's standard definitions.

Usage

```
stars.pval(p.value)
```

Arguments

p.value numeric vector of p-values

Details

Mapping from p-value ranges to symbols:

0 - 0.001 '***'

0.001 - 0.01 '**'

0.01 - 0.05 '*'

0.05 - 0.1 '+'

0.1 - 1.0 '' (No symbol)

Value

A character vector containing the same number of elements as p-value, with an attribute "legend" providing the conversion pattern.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

[symnum](#)

Examples

```
p.val <- c(0.0004, 0.0015, 0.013, 0.044, 0.067, 0.24)
stars.pval(p.val)
```

Index

- *Topic **IO**
 - ask, [3](#)
- *Topic **arith**
 - odd, [21](#)
- *Topic **character**
 - ASCIIfy, [2](#)
- *Topic **datasets**
 - ELISA, [11](#)
- *Topic **distribution**
 - permute, [22](#)
 - rdirichlet, [24](#)
- *Topic **documentation**
 - keywords, [16](#)
- *Topic **manip**
 - combinations, [7](#)
 - mixedsort, [20](#)
 - quantcut, [23](#)
 - smartbind, [29](#)
- *Topic **math**
 - foldchange, [12](#)
 - logit, [18](#)
- *Topic **misc**
 - gtools-defunct, [14](#)
 - gtools-deprecated, [15](#)
 - running, [25](#)
 - setTCPNoDelay, [28](#)
 - stars.pval, [30](#)
- *Topic **optimize**
 - binsearch, [4](#)
- *Topic **print**
 - scat, [27](#)
- *Topic **programming**
 - binsearch, [4](#)
 - defmacro, [8](#)
 - invalid, [15](#)
 - lastAdd, [17](#)
 - setTCPNoDelay, [28](#)
- *Topic **univar**
 - mixedsort, [20](#)
- *Topic **utilites**
 - ASCIIfy, [2](#)
- *Topic **utilities**
 - checkRVersion, [6](#)
 - getDependencies, [13](#)
 - setTCPNoDelay, [28](#)
 - .Last, [17](#)
- addLast (gtools-defunct), [14](#)
- ASCIIfy, [2](#)
- ask, [3](#)
- assert (gtools-defunct), [14](#)
- available.packages, [14](#)
- binsearch, [4](#)
- capture (gtools-defunct), [14](#)
- capture.output, [14](#)
- cat, [28](#)
- cbind, [30](#)
- checkRVersion, [6](#)
- choose, [8](#)
- combinations, [7](#)
- cut, [23](#)
- dbeta, [25](#)
- ddirichlet (rdirichlet), [24](#)
- defmacro, [8](#)
- Defunct, [14](#)
- Deprecated, [15](#)
- ELISA, [11](#)
- eval, [10](#)
- even (odd), [21](#)
- foldchange, [12](#)
- foldchange2logratio (foldchange), [12](#)
- function, [10](#)
- getDependencies, [13](#)
- gtools-defunct, [14](#)

gtools-deprecated, 15

help, 16

installed.packages, 14

inv.logit (logit), 18

invalid, 15

is.na, 15

is.null, 15

keywords, 16

lapply, 26

lastAdd, 14, 17

logit, 18, 19

logratio2foldchange (foldchange), 12

make.socket, 29

missing, 15

mixedorder (mixedsort), 20

mixedsort, 20

odd, 21

optim, 5

optimize, 5

options, 8

order, 21

parse, 10

permutations (combinations), 7

permute, 22

quantcut, 23

quantile, 23

R.Version, 7

rbeta, 25

rbind, 30

rdirichlet, 24

readLines, 3

round, 22

running, 25

sample, 22

sapply, 26

scan, 3

scat, 27

setTCPNoDelay, 28

showNonASCII, 2

smartbind, 29

socketConnection, 29

sort, 21

source, 10

sprint (gtools-defunct), 14

stars.pval, 30

stopifnot, 14

strmacro (defmacro), 8

substitute, 10

symnum, 31

uniroot, 5

wapply, 26