

Package ‘Smisc’

April 22, 2019

Title Sego Miscellaneous

Version 0.3.9.1

Date 2017-11-20

URL <https://pnnl.github.io/Smisc>

Description A collection of functions for statistical computing and data manipulation in R. Includes routines for data ingestion, operating on dataframes and matrices, conversion to and from lists, converting factors, filename manipulation, programming utilities, parallelization, plotting, statistical and mathematical operations, and time series.

NeedsCompilation yes

Imports parallel, utils, plyr, doParallel, methods

Suggests testthat, foreach

License BSD_3_clause + file LICENSE

RoxygenNote 6.0.1

Author Landon Sego [aut, cre]

Maintainer Landon Sego <LHS@byu.net>

Repository CRAN

Date/Publication 2019-04-22 06:54:29 UTC

R topics documented:

Smisc-package	3
allMissing	3
as.numericSilent	4
comboList	5
cumMax	6
cumsumNA	7
cusum	8
dataIn	9
df2list	11
dfplapply	12

dframeEquiv	14
dkbinom	16
doCallParallel	18
factor2character	19
factor2numeric	20
findDepMat	21
formatDT	22
getExtension	24
getPath	25
grabLast	26
hardCode	27
hpd	28
integ	30
linearMap	31
list2df	32
loadObject	34
more	35
movAvg2	36
openDevice	38
padZero	39
parLapplyW	40
parseJob	41
pcbinom	43
pddply	44
plapply	46
plotFun	49
PowerData	51
pvar	51
qbind	53
rma	54
select	54
selectElements	56
sepList	57
signal	58
smartFilter	59
smartRbindMat	61
smartTimeAxis	62
sourceDir	64
stopifnotMsg	65
stripExtension	66
stripPath	67
timeData	68
timeDiff	69
timeDiff.eg	71
timeIntegration	71
timeIt	72
timeStamp	74
umvueLN	74

<i>allMissing</i>	3
<i>vertErrorBar</i>	75

Index [77](#)

Smisc-package *Smisc: Sego Miscellaneous*

Description

A collection of functions for statistical computing, data manipulation, and visualization

Details

Package: Smisc
Type: Package
Version: 0.3.9
Date: 2017-11-20
License: BSD_3_clause + file LICENSE

Author(s)

Landon Sego
Maintainer: Landon Sego <LHS@byu.net>

Examples

```
help(package = Smisc)
```

allMissing *Identifies missing rows or columns in a data frame or matrix*

Description

Indicates which rows or columns in a data frame or matrix are completely missing (all values are NA's).

Usage

```
allMissing(dframe, byRow = TRUE)
```

Arguments

dfname A data frame or a matrix
 byRow = TRUE will identify rows that have all missing values. = FALSE identifies entire missing columns

Value

A logical vector that is true if all the elements in the corresponding row (or column) are NA's.

Author(s)

Landon Sego

See Also

[complete.cases](#)

Examples

```
# Start off with a simple data frame that has a few missing values
d1 <- data.frame(x=c(3,4,NA,1,10,NA), y=c(NA,"b","c","d","e",NA))
d1

# Identify rows where the entire row is missing
allMissing(d1)

# Only removes rows where all the values are missing
d1[!allMissing(d1),]

# All missing can also be used to identify if any of the
# columns are 'all missing'
d2 <- data.frame(x=c(rnorm(3), NA, rnorm(6)), y=rep(NA,10), z=letters[1:10])
d2

# Look for columns that are all missing
allMissing(d2, byRow=FALSE)

# Remove columns where all the values are missing
d2[,!allMissing(d2, byRow = FALSE)]
```

as.numericSilent *Convert any vector to numeric, if possible*

Description

Convert any vector to numeric, if possible

Usage

```
as.numericSilent(x)
```

Arguments

x vector of any type

Value

If `as.numeric(x)` produces an error or warning, `x` is returned unchanged. Otherwise, `as.numeric(x)` is returned.

Author(s)

Landon Sego

See Also

[as.numeric](#)

Examples

```
as.numericSilent(c("this", "that"))
as.numericSilent(c("2893.9", "9423.48"))
as.numericSilent(c("392.1", "that"))
```

comboList *Produces all possible combinations of a set of linear model predictors*

Description

Produces a list representing all possible combinations of linear model predictors

Usage

```
comboList(n.pred, outFile = NULL, njobs = 1)
```

Arguments

n.pred integer indicating the number of predictors
 outFile text string indicating the .Rdata file to which the returned list of predictor combinations will be saved. If NULL, then no file is saved.
 njobs Integer indicating the number of parallel jobs to be used in calculating the combinations, using [parLapplyW](#)

Details

Uses [combn](#) to identify the combinations.

Value

A list of class `combolist` is invisibly returned with the two components shown below. If `outFile` is not NULL, this same list is saved to `outFile`:

<code>len</code>	The total number of combinations
<code>pList</code>	A list where each element contains an integer representation of one combination of the predictors

Author(s)

Landon Segó

Examples

```
x <- comboList(4)
print(x)

# A parallel job
y <- comboList(4, njobs = 2)

# Should be equal
identical(x, y)
```

`cumMax`

Computes the maximum of the vector up to the current index

Description

For each index in a vector, computes the maximum of the vector from the beginning of the vector up to the current index

Usage

```
cumMax(x)
```

Arguments

<code>x</code>	A numeric or integer vector
----------------	-----------------------------

Value

In the sequence $x[1], x[2], \dots, x[n]$, `cumMax` returns the vector y such that for each $i = 1, \dots, n$, $y[i] = \max(x[j]; j = 1, \dots, i)$

Author(s)

Landon Segó

Examples

```
cumMax(1:10)
cumMax(c(1,3,4,5,3,2,5,1,7,8,8,6))
cumMax(c(1,3,4,5,3,2,5,1,7,8,8,6) + runif(12))
```

cumsumNA	<i>Computes the cumulative sum of a vector without propagating NAs</i>
----------	--

Description

Computes the cumulative sum of a vector without propagating NAs

Usage

```
cumsumNA(x)
```

Arguments

x An integer or double vector

Details

If x is integer, then integer addition is used. Otherwise, floating point (double) addition is used. Elements in x that were NA will continue to be NA, but the NA will not be propagated.

Value

The vector of cumulative sums.

Author(s)

Landon Sego

See Also

[cumsum](#)

Examples

```
# Compare to cumsum()
x <- as.integer(c(5, 2, 7, 9, 0, -1))
cumsum(x)
cumsumNA(x)

# Now with missing values
x[c(2,4)] <- NA
print(x)
cumsum(x)
cumsumNA(x)
```

cusum	<i>Calculates a sequence of Cusum statistics</i>
-------	--

Description

Calculates a sequence of one-sided upper Cusum statistics given the reference value and the control limit.

Usage

```
cusum(X, k, h, initial = 0, reset = TRUE)

## S3 method for class 'cusum'
print(x, ...)

## S3 method for class 'cusum'
plot(x, indexes = NULL, emphOOC = TRUE, ...)

## S3 method for class 'cusum'
signal(object, ...)
```

Arguments

X	A numeric vector.
k	The reference value.
h	The upper control limit.
initial	The starting value of the Cusum ($C[0]$).
reset	Logical indicating whether the Cusum is reset to 0 after crossing the control limit.
x	Object of class cusum
...	Additional arguments to <code>print.default</code> or <code>plot.default</code> . Ignored by the <code>signal</code> method.
indexes	A vector of indexes that select the elements of the cusum statistics that will be plotted.
emphOOC	A logical indicating whether out of control points should be emphasized in red.
object	Object of class cusum

Details

Cusum is assumed to be of the form: $C[i] = \max(0, C[i-1] + X[i] - k)$, where the signal occurs when $C[i] > h$. Note that X can be the Cusum scores, or weights, given by the log-likelihood ratio, in which case $k = 0$ would make sense.

Value

A object of class `cusum`, which is a vector of the Cusum statistics, along with the following attributes: `X`, `k`, `h`, `initial`, and `reset` (which correspond to the original arguments provided to the function) and `resetCounter`, a vector of integers corresponding to `cusum` that indicates when the Cusum resets.

Methods (by generic)

- `print`: Prints the `cusum` object by only showing the Cusum statistics and suppressing the attributes.
- `plot`: Plots the `cusum` object.
- `signal`: Prints the indexes in a `cusum` object that exceed the control limit

References

Hawkins DM and Olwell DH. (1998) Cumulative Sum Charts and Charting for Quality Improvement. Springer.

Examples

```
y <- cusum(rnorm(50), 0.2, 2)
y

# Plot the cusum
plot(y)

# Show the indexes where the chart signaled
signal(y)

# A look at the attributes
attributes(y)
```

dataIn

A flexible way to import data into R.

Description

Imports `.Rdata`, `.csv`, package data sets, and regular data frames This is especially useful when a function requires data as an argument—and in some cases the data frame already exists as an object, ready to be passed into the function, but in other cases it may be more convenient to read the data from a file.

Usage

```
dataIn(data)
```

Arguments

`data` Can be a data frame or a list of data frames (in which case, the same data frame or list is simply returned), or one of the following types of single text strings: (1) the name of a .csv file, (2) the name of a .Rdata file, or (3) a data set in a particular package, using the syntax "packageName::dataSetName".

Value

A data frame (or list of data frames) containing the requested data.

Author(s)

Landon Sego

See Also

[data](#), [loadObject](#), [read.csv](#)

Examples

```
# Write a simple data set
some.data <- data.frame(a=rnorm(10), b=rpois(10, 5))
write.csv(some.data, file="tmp.file.csv", row.names=FALSE)
save(some.data, file="tmp.file.Rdata")

A <- dataIn("tmp.file.csv")
B <- dataIn("tmp.file.Rdata")
C <- dataIn(some.data)

# We expect these to be equivalent (this should be TRUE)
all(c(dframeEquiv(A, B, verbose=FALSE)$equiv,
      dframeEquiv(B, C, verbose=FALSE)$equiv,
      dframeEquiv(A, C, verbose=FALSE)$equiv))

# Delete the files
unlink(c("tmp.file.csv", "tmp.file.Rdata"))

# Loading data from a package
more.data <- dataIn("datasets::AirPassengers")
print(more.data)

# remove example objects
rm(A, B, C, more.data, some.data)
```

`df2list`*Row-wise conversion of a data frame to a list*

Description

Convert a data frame to a list, where each element of the output list consists of a named list (or a named vector) containing a single row of the data frame.

Usage

```
df2list(df, out.type = c("list", "data.frame", "vector"))
```

Arguments

<code>df</code>	A data frame
<code>out.type</code>	Character string uniquely identifying 'list', 'data.frame', or 'vector'. If 'list', then each row of the data frame is output as a list. If 'data.frame', then each row of the data frame is output as a 1-row data frame. If 'vector', then each row of the data frame is output as a named vector. However, for 'vector,' each column of the data set must be the same type. Defaults to 'list'.

Value

A list where each element consists of a named list, single-row data frame, or a named vector, containing a single row of the original data frame.

Author(s)

Landon Sego

See Also

[list2df](#), [as.list](#)

Examples

```
d <- data.frame(a = 1:3, b = letters[1:3])
df2list(d)
```

```
d1 <- data.frame(a = 1:3, b = 7:9)
df2list(d1, out.type = "v")
```

dfplapply

*Parallelized single row processing of a data frame***Description**

Applies a function to each row of a data frame in a parallelized fashion (by submitting multiple batch R jobs). It is a convenient wrapper for `plapply`, modified especially for parallel, single-row processing of data frames.

Usage

```
dfplapply(X, FUN, ..., output.df = FALSE, njobs = parallel::detectCores() -
  1, packages = NULL, header.file = NULL, needed.objects = NULL,
  needed.objects.env = parent.frame(), workDir = "plapply",
  clobber = TRUE, max.hours = 24, check.interval.sec = 1,
  collate = FALSE, random.seed = NULL, rout = NULL, clean.up = TRUE,
  verbose = FALSE)
```

Arguments

<code>X</code>	The data frame, each row of which will be processed using <code>FUN</code>
<code>FUN</code>	A function whose first argument is a single-row data frame, i.e. a single row of <code>X</code> . The value returned by <code>FUN</code> can be any object
<code>...</code>	Additional named arguments to <code>FUN</code>
<code>output.df</code>	logical indicating whether the value returned by <code>dfplapply</code> should be a data frame. If <code>output.df = TRUE</code> , then the value returned by <code>FUN</code> should be a data frame. If <code>output.df = FALSE</code> , a list is returned by <code>dfplapply</code> .
<code>njobs</code>	The number of jobs (subsets). Defaults to one less than the number of cores on the machine.
<code>packages</code>	Character vector giving the names of packages that will be loaded in each new instance of R, using <code>library</code> .
<code>header.file</code>	Text string indicating a file that will be initially sourced prior calling <code>lapply</code> in order to create an 'environment' that will satisfy all potential dependencies for <code>FUN</code> . If <code>NULL</code> , no file is sourced.
<code>needed.objects</code>	Character vector giving the names of objects which reside in the environment specified by <code>needed.objects.env</code> that may be needed by <code>FUN</code> which are loaded into the global environment of each new instance of R that is launched. If <code>NULL</code> , no additional objects are passed.
<code>needed.objects.env</code>	Environment where <code>needed.objects</code> reside. This defaults to the environment in which <code>plapply</code> is called.
<code>workDir</code>	Character string giving the name of the working directory that will be used for the files needed to launch the separate instances of R.

<code>clobber</code>	Logical indicating whether the directory designated by <code>workDir</code> will be overwritten if it exists and contains files. If <code>clobber = FALSE</code> , and <code>workDir</code> contains files, <code>plapply</code> throws an error.
<code>max.hours</code>	The maximum number of hours to wait for the <code>njobs</code> to complete.
<code>check.interval.sec</code>	The number of seconds to wait between checking to see whether all <code>njobs</code> have completed.
<code>collate</code>	= TRUE creates a 'first-in-first-out' processing order of the elements of the input list <code>X</code> . This logical is passed to the <code>collate</code> argument of <code>parseJob</code> .
<code>random.seed</code>	An integer setting the random seed, which will result in randomizing the elements of the list assigned to each job. This is useful when the computing time for each element varies significantly because it helps to even out the run times of the parallel jobs. If <code>random.seed = NULL</code> , no randomization is performed and the elements of the input list are subdivided sequentially among the jobs. This variable is passed to the <code>random.seed</code> argument of <code>parseJob</code> . If <code>collate = TRUE</code> , no randomization is performed and <code>random.seed</code> is ignored.
<code>rout</code>	A character string giving the name of the file to where all of the <code>.Rout</code> files will be gathered. If <code>rout = NULL</code> , the <code>.Rout</code> files are not gathered, but left alone in <code>workDir</code> .
<code>clean.up</code>	= TRUE will delete the working directory.
<code>verbose</code>	= TRUE prints messages which show the progress of the jobs.

Value

A list or data frame containing the results of processing each row of `X` with `FUN`.

Author(s)

Landon Sego

See Also

[plapply](#)

Examples

```
X <- data.frame(a = 1:3, b = letters[1:3])

# Function that will operate on each of x, producing a simple list
test.1 <- function(x) {
  list(ab = paste(x$a, x$b, sep = "-"), a2 = x$a^2, bnew = paste(x$b, "new", sep = "."))
}

# Data frame output
dfplapply(X, test.1, output.df = TRUE, njobs = 2)

# List output
```

```

dfplapply(X, test.1, njobs = 2)

# Function with 2 rows of output
test.2 <- function(x) {
  data.frame(ab = rep(paste(x$a, x$b, sep = "-"), 2), a2 = rep(x$a^2, 2))
}

dfplapply(X, test.2, output.df = TRUE, njobs = 2, verbose = TRUE)

# Passing in other objects needed by FUN
a.out <- 10
test.3 <- function(x) {
  data.frame(a = x$a + a.out, b = paste(x$b, a.out, sep="-"))
}

dfplapply(X, test.3, output.df = TRUE, needed.objects = "a.out", njobs = 2)

```

dframeEquiv

Examines the equivalence of two dataframes or matrices

Description

Checks whether two data objects (data frames and/or matrices) are equivalent and returns a descriptive message describing the result.

Usage

```
dframeEquiv(d1, d2, maxAbsError = 1e-12, maxRelError = 1e-14,
  verbose = TRUE)
```

Arguments

d1	The first dataframe or matrix
d2	The dataframe or matrix that will be compared to d1
maxAbsError	Numeric values whose absolute difference is less than maxAbsError will be declared equivalent
maxRelError	Numeric values whose relative difference is within maxRelError will be declared equivalent
verbose	=TRUE prints the result of the comparison

Details

d1 and d2 do not both have to be of the same mode; i.e. d1 could be a dataframe and d2 could be a matrix. If the number of rows or the number of columns differ, then no further comparisons are made. If the colnames or rownames differ, then those differences are noted and comparison continues. If two corresponding elements are both NA, then they are considered equivalent. Likewise, Inf

is considered equivalent to Inf and -Inf is considered equivalent to -Inf. Factors in dataframes are converted to character strings prior to comparison. Comparisons are made one column at a time.

If a particular column from both objects are numeric, then for two corresponding values, say, a and b, equivalence is declared if one or more of the following occurs: 1) $a == b$, 2) $abs(a - b) < maxAbsError$, 3) $abs((a - b) / b) < maxRelError$ if $abs(b) > abs(a)$, or $abs((a - b) / a) < maxRelError$ if $abs(b) >= abs(a)$.

If both columns are not numeric, they are coerced (if need be) to character and then compared directly.

Value

Invisibly returns a list with the following components. (If the matrices do not have the same dimensions or the same colnames and rownames, then `frac.equiv`, `loc.equiv`, and `equiv.matrix` are all NULL).

<code>equiv</code>	=TRUE if d1 is equivalent to d2
<code>msg</code>	Messages that describe the comparison. (These are printed when <code>verbose=TRUE</code> .)
<code>frac.equiv</code>	The fraction of matrix elements that are equivalent
<code>loc.inequiv</code>	A data frame indicating the row and column coordinate locations of the elements that are not equivalent
<code>equiv.matrix</code>	A boolean matrix with the same dimension as d1 and d2, indicating the equivalent elements

Author(s)

Landon Sego

References

<https://randomascii.wordpress.com/category/floating-point>

See Also

[all.equal](#), [identical](#)

Examples

```
# Number of rows different
dframeEquiv(matrix(rnorm(20), nrow = 4),
             matrix(rnorm(25), nrow = 5))

# Number of columns different
dframeEquiv(matrix(rnorm(16), nrow = 4),
             matrix(rnorm(20), nrow = 4))

# Rownames differ
dframeEquiv(matrix(rnorm(9), nrow = 3, dimnames = list(1:3, NULL)),
             matrix(rnorm(9), nrow = 3, dimnames = list(letters[1:3], NULL)))
```

```

# Colnames differ
dframeEquiv(matrix(rnorm(9), nrow = 3, dimnames = list(NULL, 1:3)),
             matrix(rnorm(9), nrow = 3, dimnames = list(NULL, letters[1:3])))

# Not equivalent
x <- data.frame(x = factor(c(1,1,2,2,3,3)), y = rnorm(6))
y <- data.frame(x = factor(c(1,2,2,2,3,3)), y = c(x$y[-6],rnorm(1)))
dframeEquiv(x, y)

# Look at discrepancies
out <- dframeEquiv(x, y)
out

# Equivalent
x <- data.frame(x = letters[1:6], y = 0:5)
y <- x
dframeEquiv(x, y)

```

 dkbinom

Probability functions for the sum of k independent binomials

Description

The mass and distribution functions of the sum of k independent binomial random variables, with possibly different probabilities.

Usage

```

dkbinom(x, size, prob, log = FALSE, verbose = FALSE,
        method = c("butler", "fft"))
pkbinom(q, size, prob, log.p = FALSE, verbose = FALSE,
        method = c("butler", "naive", "fft"))

```

Arguments

x	Vector of values at which to evaluate the mass function of the sum of the k binomial variates
size	Vector of the number of trials
prob	Vector of the probabilities of success
log, log.p	logical; if TRUE, probabilities p are given as $\log(p)$ (see Note).
verbose	= TRUE produces output that shows the iterations of the convolutions and 3 arrays, A, B, and C that are used to convolve and reconvolve the distributions. Array C is the final result. See the source code in dkbinom.c for more details.

method	A character string that uniquely indicates the method. <code>butler</code> is the preferred (and default) method, which uses the algorithm given by Butler, et al. The naive method is an alternative approach that can be much slower that can handle no more the sum of five binomials, but is useful for validating the other methods. The naive method only works for a single value of <code>q</code> . The <code>fft</code> method uses the fast Fourier transform to compute the convolution of <code>k</code> binomial random variates, and is also useful for checking the other methods.
q	Vector of quantiles (value at which to evaluate the distribution function) of the sum of the <code>k</code> binomial variates

Details

`size[1]` and `prob[1]` are the size and probability of the first binomial variate, `size[2]` and `prob[2]` are the size and probability of the second binomial variate, etc.

If the elements of `prob` are all the same, then `pbinom` or `dbinom` is used. Otherwise, repeating convolutions of the `k` binomials are used to calculate the mass or the distribution functions.

Value

`dkbinom` gives the mass function, `pkbinom` gives the distribution function.

Note

When `log.p` or `log` is `TRUE`, these functions do not have the same precision as `dbinom` or `pbinom` when the probabilities are very small, i.e. the values tend to go to `-Inf` more quickly.

Author(s)

Landon Sego and Alex Venzin

References

The Butler method is based on the exact algorithm discussed by: Butler, Ken and Stephens, Michael. (1993) The Distribution of a Sum of Binomial Random Variables. Technical Report No. 467, Department of Statistics, Stanford University. <http://www.dtic.mil/dtic/tr/fulltext/u2/a266969.pdf>

See Also

[dbinom](#), [pbinom](#)

Examples

```
# A sum of 3 binomials...
dkbinom(c(0, 4, 7), c(3, 4, 2), c(0.3, 0.5, 0.8))
dkbinom(c(0, 4, 7), c(3, 4, 2), c(0.3, 0.5, 0.8), method = "b")
pkbinom(c(0, 4, 7), c(3, 4, 2), c(0.3, 0.5, 0.8))
pkbinom(c(0, 4, 7), c(3, 4, 2), c(0.3, 0.5, 0.8), method = "b")
```

```

# Compare the output of the 3 methods
pkbinom(4, c(3, 4, 2), c(0.3, 0.5, 0.8), method = "fft")
pkbinom(4, c(3, 4, 2), c(0.3, 0.5, 0.8), method = "butler")
pkbinom(4, c(3, 4, 2), c(0.3, 0.5, 0.8), method = "naive")

# Some inputs
n <- c(30000, 40000, 20000)
p <- c(0.02, 0.01, 0.005)

# Compare timings
x1 <- timeIt(pkbinom(1100, n, p, method = "butler"))
x2 <- timeIt(pkbinom(1100, n, p, method = "naive"))
x3 <- timeIt(pkbinom(1100, n, p, method = "fft"))
pvar(x1, x1 - x2, x2 - x3, x1 - x3, digits = 12)

```

doCallParallel

Call a function with a vectorized input in parallel

Description

Call a function with a vectorized input in parallel, where the function is computationally intensive.

Usage

```
doCallParallel(fun, x, ..., njobs = parallel::detectCores() - 1,
  random.seed = NULL)
```

Arguments

fun	A function, or a text string with the name of the function, whose first argument is a vector and returns a corresponding vector
x	A vector of values that is the first argument to fun
...	Additional named arguments for fun
njobs	The number of parallel jobs to spawn using parLapplyW .
random.seed	If a numeric value is provided, x is randomized to better distribute the work among the jobs if some values of x take longer to evaluate than others. The original ordering is restored before fun(x, ...) is returned. If NULL, no randomization is performed.

Details

This function is a parallelized wrapper for [do.call](#) designed for the case where fun is computationally intensive. Each element of x is evaluated independently of the other elements of x. Thus, fun(c(x1,x2)) must be equivalent to c(fun(x1), fun(x2)) in order for doCallParallel to work properly.

Value

The same result that would be had by calling `fun(x, ...)`, except calculated in parallel

Author(s)

Landon Sego

Examples

```
# Get a vector of x's
x <- rnorm(18, mean = 2, sd = 2)

# 2 cores
y1 <- doCallParallel("pnorm", x, mean = 2, sd = 2, njobs = 2)

# 2 cores and randomization
y2 <- doCallParallel(pnorm, x, mean = 2, sd = 2, njobs = 2, random.seed = 1)

# Without using doCallParallel()
y3 <- pnorm(x, mean = 2, sd = 2)

# Comparisons
identical(y1, y2)
identical(y1, y3)
```

factor2character	<i>Converts all factor variables in a dataframe to character variables</i>
------------------	--

Description

Converts all factor variables in a dataframe to character variables

Usage

```
factor2character(dframe)
```

Arguments

dframe A dataframe

Value

The same dataframe with the factor variables converted to character variables.

Author(s)

Landon Sego

Examples

```
x <- data.frame(a=factor(c(rep(1,4),rep(2,4),rep(3,4))), y=rnorm(12))
str(x)
x <- factor2character(x)
str(x)
```

factor2numeric	<i>A simple function for converting factors to numeric values</i>
----------------	---

Description

A simple function for converting factors to numeric values

Usage

```
factor2numeric(x)
```

Arguments

x A vector of type factor

Value

x converted to a numeric vector

Examples

```
# Define a factor object
y <- factor(5:7)
y

# incorrectly convert to numeric
as.numeric(y)

# correctly convert
factor2numeric(y)
```

`findDepMat`*Identify linearly dependent rows or columns in a matrix*

Description

Identify linearly dependent rows or columns in a matrix

Usage

```
findDepMat(X, rows = TRUE, tol = 1e-10)
```

Arguments

<code>X</code>	A numeric matrix
<code>rows</code>	Set <code>rows = TRUE</code> to identify which rows are linearly dependent. Set <code>rows = FALSE</code> to identify columns that are linearly dependent.
<code>tol</code>	The tolerance used to determine whether one row (or column) is a linear combination of another.

Details

A row (or column) is considered to be a linear combination of another if the maximum of the absolute successive differences of the ratios of the two rows (columns) exceeds the tolerance, `tol`. This is a fairly crude criterion and may need improvement—but it at least will identify the almost-exact linear dependencies.

`findDepMat` identifies linearly dependent rows (columns) similar to the way [duplicated](#) identifies duplicates. As such, the first instance (row or column) of a set of linearly dependent rows (or columns) is not flagged as being dependent.

The sum of the negated output of `findDepMat` should be the number of linearly independent rows (columns). This quantity is compared to the rank produced by [qr](#), and if not equal, a warning is issued. The value of `tol` is passed to [qr](#), but the criteria of convergence for [qr](#) is assuredly different from that used here to identify the linearly dependent rows (columns).

Currently this uses nested 'for' loops within R (not C). Consequently, `findDepMat` is likely to be slow for large matrices.

Value

A logical vector, equal in length to the number of rows (columns) of `X`, with `TRUE` indicating that the row (column) is linearly dependent on a previous row (column).

Author(s)

Landon Segó

Examples

```
# A matrix
Y <- matrix(c(1, 3, 4,
              2, 6, 8,
              7, 2, 9,
              4, 1, 7,
              3.5, 1, 4.5), byrow = TRUE, ncol = 3)

# Note how row 2 is multiple of row 1, row 5 is a multiple of row 3
print(Y)

findDepMat(Y)
findDepMat(t(Y), rows = FALSE)
```

formatDT

Converts date or datetime strings into alternate formats

Description

Can be used to convert date-time character vectors into other types of date-time formats. It is designed to automatically find the appropriate date and time informats without the user having to specify them.

Usage

```
formatDT(dt, date.outformat = NULL, time.outformat = NULL, posix = TRUE,
         weekday = FALSE)
```

Arguments

dt	A character vector of date values or datetime values
date.outformat	A character string requesting the date format to be returned. The following date outformats are supported: "mm/dd/yyyy", "mm-dd-yyyy", "yyyy-mm-dd", "yyyymmdd", "ddmonyyyy", and "dd-mon-yyyy". If date.outformat = NULL, then "mm/dd/yyyy" is used.
time.outformat	A character string requesting the time format to be returned. The following time outformats are supported: "hh:mm:sspm", "hh:mm:ss pm", "hh:mm:ss", "hh:mmpm", "hh:mm pm", and "hh:mm". If time.outformat = NULL, then "hh:mm:ss pm" is used.
posix	= TRUE returns date and datetime vectors of class POSIXct that can be used for time calculations.
weekday	= TRUE returns a character vector denoting the day of the week.

Details

If the input vector contains times, formatDT assumes that the dates and times are separated by at least one space. The date format and the time format of the input vector must be the same for all cells in the vector. The input format is determined by the first non-missing entry of the dt vector. Missing values (NA or "") are carried through to the output vectors without error.

In choosing the informat, formatDT first checks if the datetime string has a format of "dd/mm/yyyy hh:mm:ss pm". If so, it moves directly to the datetime conversions. Otherwise, it searches through the date and time informats listed below for a suitable match.

Acceptable date informats for dt: mm/dd/yyyy, mm-dd-yyyy, yyyy-mm-dd, yyyymmdd, ddmonyyyy, dd-mon-yyyy

Acceptable time informats for dt: hh:mm:sspm, hh:mm:ss pm, hh:mm:ss (24 hour time), hh:mmpm, hh:mm pm, hh:mm (24 hour time), hhmm (24 hour time), hhmmss (24 hour time)

Value

A list with these components:

date	A character vector of the form requested by date.outformat.
time	A character vector of the form requested by time.outformat or an empty character vector of the form "" if the time is not present in the input vector dt.
dt	A character vector containing the combined datetime using the requested formats. If time is not present in the input vector dt, then simply the date is returned.
date.posix	A vector of class "POSIXt POSIXct" containing the date. This is only returned if posix = TRUE.
dt.posix	A vector of class "POSIXt POSIXct" containing the datetime. This is only returned if posix = TRUE and time values are present in the argument dt.
weekday	A character vector indicating the days of the week. This is only returned if weekday = TRUE.

Author(s)

Landon Sego

Examples

```
# Demonstrates conversion of different datetime informats
formatDT("03/12/2004 04:31:17pm", posix = FALSE)
formatDT("12Mar2004 04:31pm", posix = FALSE)
formatDT("2004-3-12 16:31:17", posix = FALSE)
formatDT("7-5-1998 22:13")

# Specifying different types of outformats
formatDT("03/12/2004", date.outformat = "dd-mon-yyyy", posix = FALSE)
formatDT("17-Sep-1782 12:31am", date.outformat = "yyyy-mm-dd",
        time.outformat = "hh:mm", posix = FALSE)
```

```

# Processing datetime vectors
formatDT(c("03/12/2004 04:31pm", "03/12/2005 04:32:18pm"), posix = FALSE)
formatDT(c("03/12/2004 04:31:17pm", "03/12/2005 04:32:18pm"))
formatDT(c("03/12/2004 04:31:17pm", "03/12/2005 04:32:18pm"), weekday = TRUE)

# An incorrect date (will produce an error)
try(formatDT("29-Feb-2001"))

# An incorrect time will also produce an error
try(formatDT("28-Feb-2001 00:00:00 AM"))
formatDT("28-Feb-2001 12:00:00 AM")

# Illustrate the handling of missing values
formatDT(c(NA, "", "2010-10-23 3:47PM"), weekday = TRUE)

```

getExtension	<i>Get the extension of a vector of filenames</i>
--------------	---

Description

Get the extension of a vector of filenames, assuming that the extension is the set of characters that follows the last ".". A wrapper for [grabLast](#).

Usage

```
getExtension(vec, split.char = ".")
```

Arguments

vec	Character vector (usually containing filenames)
split.char	A single character used to split the character strings

Details

Assumes paths are delineated using forward slashes. If an NA is supplied, then an NA is returned. If the desired string doesn't exist (see examples below), a "" is returned.

Value

Character vector of filename extensions

Author(s)

Landon Sego

See Also

Additional functions for filename manipulations: [stripExtension](#), [stripPath](#), [getPath](#), [grabLast](#), [basename](#), [dirname](#)

Examples

```
getExtension(c(a = "this old file.doc",
              b = "that young file.rtf",
              c = "this.good.file.doc",
              d = "this_bad_file",
              e = "thisfile.",
              f = NA,
              g = "that.this.pdf",
              h = ".", i = ""))

# An example with 'real' files
files <- dir(file.path(path.package(package = "Smisc"), "data"), full.names = TRUE)
print(files)
getExtension(files)
```

getPath

Get the path of a vector of filenames

Description

Get the path of a vector of filenames

Usage

```
getPath(vec)
```

Arguments

vec Character vector (usually containing filenames)

Details

Assumes paths are delineated using forward slashes. If an NA is supplied, then an NA is returned. If the desired string doesn't exist (see examples below), a "" is returned.

Value

Character vector with pathnames only, the filename removed

Author(s)

Landon Sego

See Also

Additional functions for filename manipulations: [stripExtension](#), [getExtension](#), [stripPath](#), [grabLast](#), [basename](#), [dirname](#)

Examples

```
getPath(c(a="this.good.path/filename.R", b="nopath.R", c="/", d=NA,
         e="path1/path2/", ""))

# An example with 'real' files
files <- dir(file.path(path.package(package = "Smisc"), "data"), full.names = TRUE)
print(files)
getPath(files)
```

grabLast

Get the final set of characters after a single-character delimiter

Description

Get the final set of characters from a vector after a single-character delimiter. This can be useful in filename manipulations, among other things.

Usage

```
grabLast(vec, split.char)
```

Arguments

`vec` Character vector (usually containing filenames)
`split.char` A single character used to split the character strings

Value

Character vector of the strings that appear after the last instance of `split.char`

Author(s)

Landon Sego

See Also

Additional functions for filename manipulations: [stripExtension](#), [getExtension](#), [stripPath](#), [getPath](#), [basename](#), [dirname](#)

Examples

```
grabLast(c(a="email@nowhere.com", "this.has.no.at.sign", "@",
          "bad.email@weird.com@", NA, "2at's@email@good.net"), "@")
```

hardCode	<i>Facilitate hard coding constants into R</i>
----------	--

Description

Hard coding isn't the best practice, but sometimes it's useful, especially in one-off scripts for analyses. An typical example would be to select a large number of columns in a dataset by their names. This function facilitate hard coding constants into R by printing the code from a vector that would be needed to create that vector.

Usage

```
hardCode(x, vname = "x", vert = TRUE, ...)
```

Arguments

x	A vector (numeric, character, logical, or complex)
vname	A string indicating the name of the vector that will be "created" in the code
vert	A logical indicating whether the vector should be coded vertically (TRUE) or horizontally (FALSE)
...	Additional arguments to <code>cat</code>

Value

Prints code (via `cat`) that will create the vector. This code can then be copied into other source code. Returns nothing.

Author(s)

Landon Segó

Examples

```
# With characters
hardCode(letters[1])
hardCode(letters[1:3], vname = "new")
hardCode(letters[1], vert = FALSE)
hardCode(letters[1:3], vert = FALSE, vname = "new")

# With numbers
hardCode(3:5)
hardCode(3:5, vert = FALSE, vname = "num")

# With logicals
hardCode(TRUE)
hardCode(c(FALSE, TRUE), vert = FALSE)
hardCode(c(TRUE, FALSE, TRUE), vname = "newLogical")
```

hpd	<i>Calculate the highest posterior density credible interval for a unimodal density</i>
-----	---

Description

Calculate the highest posterior density credible interval for a unimodal density

Usage

```
hpd(pdf, support, prob = 0.95, cdf = NULL, njobs = 1, checkUnimodal = 0)
```

```
## S3 method for class 'hpd'
print(x, ...)
```

```
## S3 method for class 'hpd'
plot(x, ...)
```

Arguments

pdf	Function that takes a single numeric vector argument that returns a vector of probability density values
support	A numeric vector of length 2 giving the interval over which the random variable has support (i.e. for which the pdf is positive). For now, this must be a finite interval. Intervals for random variables within infinite support can still be calculated by setting the values of support to be suitably large and/or small. See examples.
prob	A numeric value in (0, 1] indicating the size of the desired probability for the credible interval.
cdf	A function that takes a single (not necessarily vector) argument and returns the cumulative probability. If NULL, the pdf is integrated as needed to calculate probabilities as needed. However, providing the cdf will speed up calculations.
njobs	The number of parallel jobs to spawn (where possible) using doCallParallel . This is helpful if pdf is expensive.
checkUnimodal	An integer that, when greater than 0, indicates the number of points in support for which pdf is evaluated to determine whether the function appears unimodal. This is done in parallel if njobs > 1. If checkUnimodal is not 0, it should be a large number (like 1000 or more).
x	object of class hpd, returned by hpd
...	For the plot method, these are additional arguments that may be passed to plotFun , plot.default , or abline

Details

Parallel processing (via `njobs > 1`) may be advantageous if 1) pdf is a function that is computationally expensive, 2) the cdf is not provided, in which case pdf is integrated, and/or 3) when `checkUnimodal` is large.

Value

A list of class `hpd` that contains the following elements:

lower The lower endpoint of the highest posterior density interval

upper The upper endpoint of the highest posterior density interval

prob The achieved probability of the interval

cut The horizontal cut point that gave rise to the interval

mode The mode of the density

pdf The probability density function

support The support of the pdf

Methods (by generic)

- `print`: Prints the lower and upper limits of the credible interval, along with the achieved probability of that interval.
- `plot`: Plots the density, overlaying the lower and upper limits of the credible interval

Author(s)

Landon Sego

Examples

```
# A credible interval using the standard normal
int <- hpd(dnorm, c(-5,5), prob = 0.90, njobs = 2)
print(int)
plot(int)
```

```
# A credible interval with the gamma density
int <- hpd(function(x) dgamma(x, shape = 2, rate = 0.5), c(0, 20),
           cdf = function(x) pgamma(x, shape = 2, rate = 0.5), prob = 0.8)
print(int)
plot(int)
```

```
# A credible interval using the Beta density
dens <- function(x) dbeta(x, 7, 12)
dist <- function(x) pbeta(x, 7, 12)
int <- hpd(dens, c(0, 1), cdf = dist)
print(int)
plot(int)
```

integ *Simple numerical integration routine*

Description

Estimates the integral of a real-valued function using Simpson's or the Trapezoid approximation

Usage

```
integ(y, x = NULL, a = NULL, b = NULL, method = c("simpson",  
"trapezoid"))
```

Arguments

y	Vector of f(x) values
x	Numeric vector of sorted x values, each element of x should have a corresponding element in y. Only required for the trapezoid method. Not required for the Simpson method.
a	The lower limit of integration, only required for the Simpson method.
b	The upper limit of integration, only required for the Simpson method.
method	The method of integration (can use just the first letter). Defaults to simpson.

Details

For the Simpson method, y is a numeric vector of f(x), evaluated at an odd number of evenly spaced x's in the interval [a,b].

For the trapezoid method, the elements of x and y should correspond to one another, and x must be sorted in ascending order. The lengths of x and y should be the same, and they may be odd or even. The elements of x may be irregularly spaced.

Value

A single numeric estimate of the integral of f(x) over [a, b] (Simpson) or over the range of x (trapezoid).

Author(s)

Landon Segó

References

Ellis R, Gulick D. "Calculus: One and Several Variables," Harcourt Brace Jovanovich, Publishers: New York, 1991; 479-482.

See Also

[integrate](#)

Examples

```
# The Beta density from 0 to 0.7
integ(dbeta(seq(0, 0.7, length = 401), 2, 5), a = 0, b = 0.7)

# Checking result with the cdf
pbeta(0.7, 2, 5)

# f(x) = x^2 from 0 to 3
integ(seq(0, 3, length = 21)^2, a = 0, b = 3)

# A quadratic function with both methods
x <- seq(0, 3, length = 51)
integ(x^2, x = x, method = "t")
integ(x^2, a = 0, b = 3, method = "s")

# Now a linear function
x <- seq(0, 2, length = 3)
y <- 2 * x + 3
integ(y, x = x, method = "t")
integ(y, a = 0, b = 2)
```

linearMap

*Linear mapping of a numeric vector or scalar***Description**

Linear mapping of a numeric vector or scalar from one closed interval to another

Usage

```
linearMap(x, D = range(x), R = c(0, 1))
```

Arguments

x	a numeric vector
D	a vector with 2 elements, the first being the lower endpoint of the domain, the upper being the upper endpoint of the domain. Note R[1] must be less than R[2].
R	a vector with 2 elements indicating the range of the linear mapping. R[1] is mapped to D[1], and R[2] is mapped to D[2].

Details

The mapping is $f : D \rightarrow R$, where $f(D[1]) = R[1]$ and $f(D[2]) = R[2]$.

Value

The linear mapping of x from D to R

Author(s)

Landon Sego

Examples

```
x <- seq(0, 1, length = 5)

# An increasing linear map
linearMap(x, R = c(4, 7))

# A decreasing map
linearMap(x, R = c(7, 4))

# A shift
linearMap(x, R = c(-1, 0))

# The identity map:
y <- linearMap(x, D = c(0, 1), R = c(0, 1))
identical(y, x)
```

`list2df`*Convert a list to a data frame*

Description

Convert a list of vectors (or data frames) with same numbered lengths (or number of columns) into a data frame.

Usage

```
list2df(vList, col.names = NULL, row.names = NULL, convert.numeric = TRUE,
        strings.as.factors = FALSE)
```

Arguments

<code>vList</code>	List of vectors, data frames, or lists. See Details.
<code>col.names</code>	Optional character vector of length <code>n</code> with column names that will be given to the output data frame. If <code>col.names = NULL</code> , column names are extracted if possible from the column names (or names) of the data frames (or vectors).
<code>row.names</code>	Optional character vector with length equivalent to the length of <code>vList</code> containing the row names of the output data frame. If <code>row.names = NULL</code> , row names from the data frames (or names of the <code>vList</code> elements) if possible.
<code>convert.numeric</code>	If <code>vList</code> is list of vectors, <code>= TRUE</code> attempts to convert each column to numeric if possible using as.numericSilent
<code>strings.as.factors</code>	If <code>vList</code> is a list of vectors or lists, <code>= FALSE</code> converts factors into characters using factor2character .

Details

If the elements of `vList` are vectors, each vector must have the same length, `n`, and the resulting data frame will have `n` columns. If the elements of `vList` are data frames, each data frame must have the same structure (though they may have differing numbers of rows). If the elements of `vList` are lists, each list is first converted to a data frame via `as.data.frame` and the resulting data frames must have the same structure (though they may have differing numbers of rows).

It is permissible for `vList` to contain `NULL` elements. `list2df` performs numerous consistency checks to ensure that contents of `vList` which are combined into the resulting data frame are conformable, labeled consistently, of the equivalent class when necessary, etc.

Value

If `vList` is list of data frames, a data frame resulting from efficiently row binding the data frames in `vList` is returned. If `vList` is a list of vectors, a data frame is returned where the first column contains the first elements of the list vectors, the second column contains the second elements of the list vectors, etc.

Author(s)

Landon Sego

Examples

```
# For a list of vectors
x <- c("r1c1 1", "r2c1 2", "r3c1 3", "r4c4 4")
y <- strsplit(x, "\\ ")
y
list2df(y)
list2df(y, col.names = LETTERS[1:2])

# Here's another list of vectors
z <- list(NULL, a = c(first = 10, second = 12), NULL, b = c(first = 15, second = 17))
z
list2df(z)

# For a list of data frames
z <- list(d1 = data.frame(a = 1:4, b = letters[1:4]),
          d2 = data.frame(a = 5:6, b = letters[5:6]))
z
list2df(z)

# A list of lists
z <- list(list(a = 10, b = TRUE, c = "hi"), list(a = 12, b = FALSE, c = c("there", "bye")))
z
list2df(z)
```

loadObject	<i>Loads and returns the object(s) in one or more ".Rdata" files</i>
------------	--

Description

Loads and returns the object(s) in the ".Rdata" file. This is useful for naming the object(s) in the ".Rdata" file something other than the name it was saved as.

Usage

```
loadObject(RdataFiles)
```

Arguments

RdataFiles A character vector containing the '.Rdata' filename(s)

Value

The object(s) contained in RdataFiles, organized into lists and named as required to distinguish them completely. See Examples.

Author(s)

Landon Sego

See Also

[load](#)

Examples

```
# Create some filenames we'll use in this example
fileName1 <- "demo_load_object_1.Rdata"
fileName2 <- "demo_load_object_2.Rdata"

# Make an example object
x <- data.frame(a = 1:10, b = rnorm(10))

# Save and reload the file
save(x, file = fileName1)
rm(x)
y <- loadObject(fileName1)

# Note how 'x' is not in the global environment
ls()

# This is the object that was in 'fileName'
print(y)
```

```
# Here's an example with two objects saved in a single file
a <- rnorm(10)
b <- rnorm(20)
save(a, b, file = fileName2)

# Load the results and show them
z <- loadObject(fileName2)
print(z)

# And here's an example with more than one file
both <- loadObject(c(fileName1, fileName2))
both

# Delete the files
unlink(c(fileName1, fileName2))
```

more

Display the contents of a text file to the R console

Description

Display the contents of a text file to the R console

Usage

```
more(file, n = -1, display = c("all", "head", "tail"))
```

Arguments

file	Text string giving the file name
n	Integer specifying the maximum number of lines to read from the file. This is passed to the <code>n</code> argument of <code>readLines</code> . The default is <code>-1</code> , which will read all the lines in the file.
display	Text string that uniquely identifies one of <code>"all"</code> , <code>"head"</code> , or <code>"tail"</code> . Defaults to <code>"all"</code> , which causes all lines read from the file to be displayed, <code>"head"</code> shows the first 6 lines that were read, and <code>"tail"</code> shows the last 6 lines that were read.

Value

Returns nothing, but it does display the contents of the file on the R console.

Author(s)

Landon Sego

See Also

[readLines](#)

Examples

```
cat("Here's a file\n", "with a few lines\n",
    "to read.\n", sep = "\n", file = "tmpFile.txt")
more("tmpFile.txt")
unlink("tmpFile.txt")
```

 movAvg2

Calculate the moving average using a 2-sided, symmetric window

Description

Wrapper for [smartFilter](#) that creates a set of symmetric weights for the 2-sided window based on the Gaussian kernel, exponential decay, linear decay, or simple uniform weights, and then calculates the moving average (dot product) using those weights.

Usage

```
movAvg2(y = NULL, bw = 30, type = c("gaussian", "exponential", "linear",
  "uniform"), furthest.weight = 0.01, center.weight = 1, ...)

## S3 method for class 'movAvg2'
print(x, ...)

## S3 method for class 'movAvg2'
plot(x, ...)
```

Arguments

<code>y</code>	The numerical vector for which the moving averages will be calculated. If <code>NULL</code> , the moving averages are not calculated, but the weights are calculated and included in the attributes of the returned object.
<code>bw</code>	A single, positive whole number that indicates the 'bandwidth' of the window, which is roughly half the width of the moving window. The total width of the window is $2 * bw + 1$.
<code>type</code>	Character string which uniquely identifies the type of weights to use, corresponding to the Gaussian kernel, exponential decay, linear decay, or uniform weights. Defaults to <code>gaussian</code> .
<code>furthest.weight</code>	A single, positive number corresponding to the unnormalized value of the weights at the left and right edges of the window. Ignored when <code>type = 'uniform'</code> .
<code>center.weight</code>	A single, positive number corresponding to the unnormalized value of the weights at the center of the window.
<code>...</code>	For <code>movAvg2</code> , these are additional arguments to smartFilter . For the <code>print</code> and <code>plot</code> methods, the <code>"..."</code> are additional arguments passed to print.default and plot.default , respectively.
<code>x</code>	Object of class <code>movAvg2</code> .

Details

All the weights are normalized (so that they sum to 1) prior to calculating the moving average. The moving "average" is really the moving dot product of the normalized weights and the corresponding elements of y .

Since it uses [smartFilter](#) to calculate the moving average, the moving average for points near the edge of the series or in the neighborhood of missing values are calculated using as much of the window weights as possible.

Value

An object class `movAvg2`, which is a numeric vector containing the moving average (dot product) of the series, with attributes that describe the weights. If $y = \text{NULL}$, `numeric(0)` is returned.

Methods (by generic)

- `print`: Prints the `movAvg2` object by only showing the series of dot products and suppressing the attributes.
- `plot`: Plots the unnormalized weights.

Author(s)

Landon Sego

See Also

[smartFilter](#), [filter](#)

Examples

```
z <- movAvg2(rnorm(25), bw = 10, type = "e", center.weight = 2)
z

# Look at the attributes
attributes(z)

# Plot the weights
plot(z)

# If we just want to see the weights (without supplying data)
plot(movAvg2(bw = 20, type = "g", center.weight = 1))

# Note how it produces the same values as filter (except at the edge
# of the series
x <- rnorm(10)
movAvg2(x, bw = 2, type = "u")
filter(x, rep(1, 5) / 5)

# These are also the same, except at the edge.
movAvg2(x, bw = 1, type = "l", furthest.weight = 0.5, center.weight = 1)
filter(x, c(0.5, 1, 0.5) / 2)
```

openDevice	<i>Opens a graphics device based on the filename extension</i>
------------	--

Description

Based on the filename extension, will open plotting device using one of the following graphics functions: [postscript](#), [pdf](#), [jpeg](#), [tiff](#), [png](#), or [bmp](#).

Usage

```
openDevice(fileName, ...)
```

Arguments

fileName	Character string giving the filename for the graphics output. The following are acceptable filename extensions: ps, pdf, jpg, jpeg, tif, png, or bmp.
...	Named arguments to the device functions listed above. Arguments that do not match are silently ignored.

Value

The graphics device is opened and the filename is invisibly returned.

Author(s)

Landon Sego

Examples

```
# Open 3 example devices
openDevice("ex1.pdf", height=6, width=12)
plot(1:10, 1:10)

openDevice("ex1.jpg")
plot(1:10, 1:10)

openDevice("ex1.png")
plot(1:10, 1:10)

# List the devices and their filenames
dev.list()
dir(pattern = "ex1")

# Turn each of the 3 devices off
for (i in 1:3) {
  dev.off(dev.list()[length(dev.list())])
}

# Delete the created files
```

```
unlink(c("ex1.pdf", "ex1.png", "ex1.jpg"))

# List the current devices
dev.list()
```

padZero

Pad a vector of numbers with zeros

Description

Pad a numeric vector with zeros so that each element in the vector either has 1) the same number of characters or 2) the same number of trailing decimal places.

Usage

```
padZero(vec, num = NULL, side = c("left", "right"))
```

Arguments

vec	The numeric vector to be padded
num	The maximum number of zeros that will be added. If NULL, the value is chosen based on the longest string in the vector.
side	The side to which the zeros are added.

Details

For side = 'left', num is the number of characters that each element of the padded, output vector should have. If num = NULL, the largest number of characters that appears in the vector is chosen for num.

For side = 'right', num is the number of decimal places to be displayed. If num = NULL, the number of decimals in the element with the largest number of decimal places is used.

Note that vec must be numeric when side = 'right'. However, vec may be character when side = 'left'.

Value

Character vector with the leading (or trailing) elements padded with zeros.

Author(s)

Landon Segó

Examples

```
# Examples with 0's on the left
padZero(c(1, 10, 100))
padZero(c(1, 10, 100), num = 4)

# Examples with 0's on the right
padZero(c(1.2, 1.34, 1.399), side = "r")
padZero(c(1.2, 1.34, 1.399), num = 5, side = "r")
```

parLapplyW

A wrapper for parLapply

Description

A wrapper to make calls to [parLapply](#) easier by initializing the cluster, exporting objects and expressions to the worker nodes, and shutting down the cluster.

Usage

```
parLapplyW(X, FUN, ..., njobs = parallel::detectCores() - 1, expr = NULL,
  varlist = NULL, envir = parent.frame())
```

Arguments

X	A vector (atomic or list)
FUN	A function or character string naming a function whose first argument will be passed the elements of X
...	Additional named arguments to FUN
njobs	The number of jobs (cores) to use
expr	An expression that will be evaluated on each worker node via a call to clusterEvalQ
varlist	Character vector of names of objects to export to each worker node via clusterExport
envir	The environment containing the variables in varlist that will be exported

Details

The expression in `expr` is evaluated before the variables in `varlist` are exported.

Value

The same result given by `lapply(X, FUN, ...)`

Author(s)

Landon Sego

See Also

[lapply](#), [parLapply](#), [plapply](#)

Examples

```
# Create a simple list
a <- list(a = rnorm(10), b = rnorm(20), c = rnorm(15))

# Some objects that will be needed by f1:
b1 <- rexp(20)
b2 <- rpois(10, 20)

# The function, which will depend on the Smisc package
f1 <- function(x, someText = "this.stuff") {
  textJunk <- stripExtension(someText)
  result <- mean(x) + max(b1) - min(b2)
  return(list(textJunk, result))
}

# Call parLapplyW(), loading the Smisc package and passing in the "b1" and "b2" objects
res.1 <- parLapplyW(a, f1, someText = "that.stuff", njobs = 2,
  expr = expression(library(Smisc)),
  varlist = c("b1", "b2"))

print(res.1)

# Call parLapplyW(), note that we're sending a different value for "b2" into the worker nodes
# via the 'expr' argument
res.2 <- parLapplyW(a, f1, someText = "that.stuff", njobs = 2,
  expr = expression({library(Smisc); b2 <- rnorm(10)}),
  varlist = c("b1"))

# These should not be equivalent
identical(res.1, res.2)

# Call lapply
res.3 <- lapply(a, f1, someText = "that.stuff")

# Compare results, these should be equivalent
identical(res.1, res.3)
```

parseJob

Parses a collection of elements into (almost) equal-sized groups

Description

Parses a collection of elements into (almost) equal-sized groups. Useful for splitting up an R job that operates over a large dataframe or list into multiple jobs.

Usage

```
parseJob(n, njobs, collate = FALSE, random.seed = NULL,
         text.to.eval = FALSE)
```

Arguments

<code>n</code>	The number elements to be parsed
<code>njobs</code>	The number of groups
<code>collate</code>	= TRUE alternative ordering of the grouping. See example below.
<code>random.seed</code>	An integer setting the random seed, which will result in randomizing the elements among the jobs. If NULL, no randomization is performed. Randomization cannot be performed if <code>collate = TRUE</code> or if <code>text.to.eval = TRUE</code> . Randomization is useful when the computing time for each element varies significantly because it helps to even out the run times of parallel jobs.
<code>text.to.eval</code>	If = TRUE, a text expression is returned, that when evaluated, will produce the sequence of elements for that group. This is especially useful when <code>n</code> is very large. (See Value section below).

Value

When `text.to.eval = FALSE`, a list with `njobs` elements is returned, each element containing a numeric vector of the element numbers which correspond to that group. When `text.to.eval = TRUE`, a list with `njobs` elements is returned, each element containing text (let's call it `val`), that when evaluated using `eval(parse(text = val))`, will produce the sequence of numbers corresponding to the group.

Author(s)

Landon Segó

Examples

```
x <- parseJob(29, 6)
print(x)

# To see the range of each group
lapply(x, range)

# To see the length of each group
lengths(x)

# Randomize the outcome
parseJob(32, 5, random.seed = 231)

# Example of 'text.to.eval = TRUE'
out <- parseJob(11, 3, text.to.eval = TRUE)
out
lapply(out, function(x) eval(parse(text = x)))
```

```
# Example of 'collate = TRUE' and 'text.to.eval = TRUE'  
parseJob(11, 3, collate = TRUE)  
parseJob(11, 3, collate = TRUE, text.to.eval = TRUE)
```

pbinom

A continuous version of the binomial cdf

Description

Uses the incomplete beta function to calculate a continuous version of the binomial cumulative distribution function.

Usage

```
pbinom(x, n, p, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x	Real valued vector of the number of successes.
n	Real valued vector, all elements in $[0, \text{Inf})$, of the number of trials.
p	Real valued vector, all elements in $[0, 1]$, of the probability of success.
lower.tail	logical; if TRUE, the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are returned as $\log(p)$.

Details

pbinom is equivalent to [pbinom](#) for integer values of n and x.

Note that pbinom does not recycle vectors in the usual fashion. Each of the arguments x, n, and p should have the same length, or, one or more of them can have the same length, so long as the other arguments have length 1. See examples below.

Value

Returns a continuous version of the binomial distribution function.

Note

This function was based on binom.c in the R source code.

Author(s)

Landon Sego

See Also

[pbinom](#), [pbeta](#)

Examples

```
x <- c( 2,  3,  5, 5.2,  5)
n <- c( 4,  5,  7,  7, 7.2)
p <- c(0.2, 0.1, 0.8, 0.8, 0.7)

pcbinom(x, n, p)
pbinom(x, n, p)

# These will work
pcbinom(c(7.3, 7.8), 12, 0.7)
pcbinom(c(7.3, 7.8), c(12,13), 0.7)
pcbinom(12.1, c(14.2,14.3), 0.6)

# But these won't
try(pcbinom(c(7.3, 7.8), c(12, 14, 16), 0.7))
try(pcbinom(c(7.3, 7.8), c(12, 14, 16), c(0.7, 0.8)))
```

pddply

Parallel wrapper for plyr::ddply

Description

Parallel implementation of `plyr::ddply` that suppresses a spurious warning when `plyr::ddply` is called in parallel. All of the arguments except `njobs` are passed directly to arguments of the same name in `plyr::ddply`.

Usage

```
pddply(.data, .variables, .fun = NULL, ..., njobs = parallel::detectCores()
- 1, .progress = "none", .inform = FALSE, .drop = TRUE,
.paropts = NULL)
```

Arguments

<code>.data</code>	data frame to be processed
<code>.variables</code>	character vector of variables in <code>.data</code> that will define how to split the data
<code>.fun</code>	function to apply to each piece
<code>...</code>	other arguments passed on to <code>.fun</code>
<code>njobs</code>	the number of parallel jobs to launch, defaulting to one less than the number of available cores on the machine
<code>.progress</code>	name of the progress bar to use, see <code>plyr::create_progress_bar</code>
<code>.inform</code>	produce informative error messages? This is turned off by default because it substantially slows processing speed, but is very useful for debugging

- `.drop` should combinations of variables that do not appear in the input data be preserved (FALSE) or dropped (TRUE, default)
- `.paropts` a list of additional options passed into the `foreach::foreach` function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages. Use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

An innocuous warning is thrown when `plyr::ddply` is called in parallel: <https://github.com/hadley/plyr/issues/203>. This function catches and hides that warning, which looks like this:

Warning messages:

```
1: <anonymous>: ... may be used in an incorrect context: '.fun(piece, ...)'
```

If `njobs = 1`, a call to `plyr::ddply` is made without parallelization, and anything supplied to `.paropts` is ignored. See the documentation for `plyr::ddply` for additional details.

Value

The object data frame returned by `plyr::ddply`

See Also

[plyr::ddply](#)

Examples

```
data(baseball, package = "plyr")

# Summarize the number of entries for each year in the baseball dataset with 2 jobs
o1 <- pddply(baseball, ~ year, nrow, njobs = 2)
head(o1)

# Verify it's the same as the non-parallel version of plyr::ddply()
o2 <- plyr::ddply(baseball, ~ year, nrow)
identical(o1, o2)

# Another possibility
o3 <- pddply(baseball, "lg", c("nrow", "ncol"), njobs = 2)
o3

o4 <- plyr::ddply(baseball, "lg", c("nrow", "ncol"))
identical(o3, o4)

# A nonsense example where we need to pass objects and packages into the cluster
number1 <- 7

f <- function(x, number2 = 10) {
```

```

  paste(x$id[1], padZero(number1, num = 2), number2, sep = "--")
}

# In parallel
o5 <- pddply(baseball[1:100,], "year", f, number2 = 13, njobs = 2,
             .paropts = list(.packages = "Smisc", .export = "number1"))
o5

# Non parallel
o6 <- plyr::ddply(baseball[1:100,], "year", f, number2 = 13)
identical(o5, o6)

```

plapply

Simple parallelization of lapply

Description

Parses a large list into subsets and submits a separate batch R job that calls [lapply](#) on the subset. `plapply` has some features that may not be readily available in other parallelization functions like [mclapply](#) and [parLapply](#):

- The `.Rout` files produced by each R instance are easily accessible for convenient debugging of errors or warnings. The `.Rout` files can also serve as an explicit record of the work that was performed by the workers
- Three options are available for the ordering of the processing of the list elements: the original list order, randomized, or collated (first-in-first-out).
- In each R instance, pre-processing or post-processing steps can be performed before and after the call to [lapply](#)

These pre-processing and post-processing steps can depend on the instance of R, such that each instance can be treated differently, if desired. These features give greater control over the computing process, which can be especially useful for large jobs.

Usage

```

plapply(X, FUN, ..., njobs = parallel::detectCores() - 1, packages = NULL,
        header.file = NULL, needed.objects = NULL,
        needed.objects.env = parent.frame(), workDir = "plapply",
        clobber = TRUE, max.hours = 24, check.interval.sec = 1,
        collate = FALSE, random.seed = NULL, rout = NULL, clean.up = TRUE,
        verbose = FALSE)

```

Arguments

<code>X</code>	A list or vector, each element of which will be the input to FUN
<code>FUN</code>	A function whose first argument is an element of X
<code>...</code>	Additional named arguments to FUN
<code>njobs</code>	The number of jobs (subsets). Defaults to one less than the number of cores on the machine.
<code>packages</code>	Character vector giving the names of packages that will be loaded in each new instance of R, using library .
<code>header.file</code>	Text string indicating a file that will be initially sourced prior calling lapply in order to create an 'environment' that will satisfy all potential dependencies for FUN. If NULL, no file is sourced.
<code>needed.objects</code>	Character vector giving the names of objects which reside in the environment specified by <code>needed.objects.env</code> that may be needed by FUN which are loaded into the global environment of each new instance of R that is launched. If NULL, no additional objects are passed.
<code>needed.objects.env</code>	Environment where <code>needed.objects</code> reside. This defaults to the environment in which <code>plapply</code> is called.
<code>workDir</code>	Character string giving the name of the working directory that will be used for the files needed to launch the separate instances of R.
<code>clobber</code>	Logical indicating whether the directory designated by <code>workDir</code> will be overwritten if it exists and contains files. If <code>clobber = FALSE</code> , and <code>workDir</code> contains files, <code>plapply</code> throws an error.
<code>max.hours</code>	The maximum number of hours to wait for the <code>njobs</code> to complete.
<code>check.interval.sec</code>	The number of seconds to wait between checking to see whether all <code>njobs</code> have completed.
<code>collate</code>	= TRUE creates a 'first-in-first-out' processing order of the elements of the input list X. This logical is passed to the <code>collate</code> argument of parseJob .
<code>random.seed</code>	An integer setting the random seed, which will result in randomizing the elements of the list assigned to each job. This is useful when the computing time for each element varies significantly because it helps to even out the run times of the parallel jobs. If <code>random.seed = NULL</code> , no randomization is performed and the elements of the input list are subdivided sequentially among the jobs. This variable is passed to the <code>random.seed</code> argument of parseJob . If <code>collate = TRUE</code> , no randomization is performed and <code>random.seed</code> is ignored.
<code>rout</code>	A character string giving the name of the file to where all of the <code>.Rout</code> files will be gathered. If <code>rout = NULL</code> , the <code>.Rout</code> files are not gathered, but left alone in <code>workDir</code> .
<code>clean.up</code>	= TRUE will delete the working directory.
<code>verbose</code>	= TRUE prints messages which show the progress of the jobs.

Details

plapply applies FUN to each element of the list X by parsing the list into njobs lists of equal (or almost equal) size and then applies FUN to each sublist using lapply.

A separate batch instance of R is launched for each sublist, thus utilizing another core of the machine. After the jobs complete, the njobs output lists are reassembled. The global environments for each batch instance of R are created by writing/reading data to/from disc.

If collate = TRUE or random.seed = Integer value, the output list returned by plapply is reordered to reflect the original ordering of the input list, X.

An object called process.id (consisting of an integer indicating the process number) is available in the global environment of each instance of R.

Each instance of R runs a script that performs the following steps:

1. Any other packages indicated in the packages argument are loaded via calls to library()
2. The process.id global variable is assigned to the global environment of the R instance (having been passed in via a command line argument)
3. The header file (if there is one) is sourced
4. The expression pre.process.expression is evaluated if an object of that name is present in the global environment. The object pre.process.expression may be passed in via the header file or via needed.objects
5. lapply is called on the sublist, the sublist is called X.i
6. The expression post.process.expression is evaluated if an object of that name is present in the global environment. The object post.process.expression may be passed in via the header file or via needed.objects
7. The output returned by lapply is assigned to the object X.i.out, and is saved to a temporary file where it will be collected after all jobs have completed
8. Warnings are printed

If njobs = 1, none of the previous steps are executed, only this call is made: lapply(X, FUN, ...)

Value

A list equivalent to that returned by lapply(X, FUN, ...).

Author(s)

Landon Sego

See Also

[parLapplyW](#), [dfplapply](#), [parLapply](#), [mclapply](#)

Examples

```

# Create a simple list
a <- list(a = rnorm(10), b = rnorm(20), c = rnorm(15), d = rnorm(13),
          e = rnorm(15), f = rnorm(22))

# Some objects that will be needed by f1:
b1 <- rexp(20)
b2 <- rpois(10, 20)

# The function
f1 <- function(x) mean(x) + max(b1) - min(b2)

# Call plapply
res1 <- plapply(a, f1, njobs = 2, needed.objects = c("b1", "b2"),
                check.interval.sec = 0.5, max.hours = 1/120,
                workDir = "example1", rout = "example1.Rout",
                clean.up = FALSE)

print(res1)

# Look at the collated 'Rout' file
more("example1.Rout")

# Look at the contents of the working directory
dir("example1")

# Remove working directory and Rout file
unlink("example1", recursive = TRUE, force = TRUE)
unlink("example1.Rout")

# Verify the result with lapply
res2 <- lapply(a, f1)

# Compare results
identical(res1, res2)

```

plotFun

Plot one or more functions on a single plot

Description

A convenient wrapper function for plotting one or more functions on a single plot. If the function(s) is/are expensive to calculate, function values can be calculated in parallel.

Usage

```

plotFun(fun, xlim, col = rainbow(length(fun)), lty = 1:length(fun),
        type = "l", legendLabels = NULL, relX = 0.7, relY = 0.9,
        nPoints = 1000, njobs = 1, ...)

```

Arguments

fun	A function or a list of functions to be plotted. These functions should take a single, numeric vector argument and return a corresponding vector of outputs.
xlim	A numeric vector with two elements that define the domain over which the function(s) will be evaluated and plotted, just as in <code>plot.default</code> in the graphics package.
col	A vector of colors to use in the plotting. It's length should match the length of fun. See <code>par</code> for more info about the <code>col</code> graphics parameter.
lty	A vector of line types to use in the plotting. It's length should match the length of fun. See <code>par</code> for more info about the <code>lty</code> graphics parameter.
type	A single character indicating the type of plotting. This is passed to the <code>type</code> argument of <code>plot.default</code> .
legendLabels	A character vector with descriptive names that will appear in the legend, corresponding to each function. If <code>NULL</code> , no legend is drawn. This character vector is passed to the <code>legend</code> argument in <code>legend</code> from the graphics package.
relX	A numeric value in <code>[0, 1]</code> designating the relative horizontal (x) position of the legend in the plot.
relY	A numeric value in <code>[0, 1]</code> designating the relative vertical (y) position of the legend in the plot.
nPoints	The number of points that are evaluated and plotted for each function over the interval given by <code>xlim</code> .
njobs	The number of parallel jobs to spawn using <code>doCallParallel</code> .
...	Additional graphical arguments passed to <code>plot.default</code> , <code>lines</code> , and <code>legend</code> . If an argument name specified in ... matches an argument name in any of these three functions, the argument is passed to that function. For example, the line width, <code>lwd</code> , would be passed to all three (<code>plot.default</code> , <code>lines</code> , and <code>legend</code>).

Value

The plot of the function(s)

Author(s)

Landon Sego

Examples

```
# A single function with a single argument
f <- function(x) x^2
plotFun(f, c(-2, 3), col = "Black", lty = 2, las = 1)

# A handful of beta density functions, note how they take a single argument
fList <- list(function(x) dbeta(x, 10, 10),
              function(y) dbeta(y, 3, 3),
              function(z) dbeta(z, 0.5, 0.50))
```

```
# Plot them all on the same plot
plotFun(fList, c(0.0001, 0.9999), ylim = c(0, 3.5),
        col = c("Red", "Black", "Blue"), lty = rep(1, 3),
        xlab = "x", ylab = expression(f(x)),
        legendLabels = c("a = 10, b = 10", "a = 3, b = 3", "a = 0.5, b = 0.5"),
        relX = 0.6, relY = 1, lwd = 3, main = "Gamma Densities")
```

PowerData

An example of power data

Description

A small, altered, subset of total rack power for one of the racks in a data center. Used for illustration of [timeIntegration](#)

Format

The format is: Named num [1:12] 5.15 5.15 5.15 5.16 5.16 ... - attr(*, "names")= chr [1:12] "2008-05-06 17:00:01" "2008-05-06 17:00:17" "2008-05-06 17:00:21" "2008-05-06 17:00:32" ...

Examples

```
data(PowerData)
print(PowerData)
rm(PowerData, envir=.GlobalEnv)
```

pvar

Prints the name and value of one or more objects

Description

A convenience function for writing the names and values of objects to the session window (and/or to another object). Especially useful to keep track of variables within loops.

Usage

```
pvar(..., digits = NULL, abbrev = NULL, sep = ";", verbose = TRUE)
```

Arguments

...	Objects whose names and values are to be printed, separated by commas. Can also be a simple list.
digits	Number of digits to display for numeric objects. Defaults to NULL, which corresponds to no restriction on the number of digits. This is passed to the digits argument of <code>round</code> .
abbrev	Number of characters to display for character objects. Defaults to NULL, which corresponds to no restriction on the number of characters.
sep	Character string that separates the objects that are printed
verbose	=TRUE writes the value of the object(s) to the session window

Details

Input objects can be numeric, character, and/or logical. They can also be atomic or vectors. It will accept data frames and matrices without error, but the results won't be easily readable.

Value

Invisibly returns a character string containing the names of the objects and their values

Author(s)

Landon Sego

Examples

```
x <- 10
y <- 20.728923
z <- "This.long.string"

pvar(x, y, z)
pvar(x, y, z, digits = 2)
pvar(x, y, z, abbrev = 4)
pvar(x, y, z, digits = 2, abbrev = 4)
pvar(x, y, z, sep = ",")

# values can be vectors too
x <- 10:12
y <- c("This", "That")
v2 <- pvar(x, y, verbose = FALSE)
v2

# Or a simple list
pvar(list(x = 1:2, y = "this", z = TRUE))

# Can be useful for keeping track of iterations in loops
for (i in 1:2) {
  for (j in letters[1:2]) {
    for (k in c("this", "that")) {
      pvar(i, j, k)
    }
  }
}
```

```

    }
  }
}

```

qbind

Quickly row and column bind many objects together at once

Description

A wrapper for `rbind` and `cbind` to quickly bind numerous objects together at once using a single call to `rbind` or `cbind`. This function is most helpful when there are many objects to bind and the object names are easily represented in text.

Usage

```
qbind(objects, type = c("row", "col", "c"))
```

Arguments

objects	A character vector with the names of the objects to be bound together
type	The type of binding, "row" for <code>rbind</code> or "col" for <code>cbind</code> , and "c" for concatenating using <code>c</code> . Defaults to "row".

Value

The bound object

Author(s)

Landon Sego

Examples

```

# Row binding
a1 <- data.frame(a = 1:3, b = rnorm(3), c = runif(3))
a2 <- data.frame(a = 4:6, b = rnorm(3), c = runif(3))
a3 <- data.frame(a = 7:9, b = rnorm(3), c = runif(3))

qbind(paste("a", 1:3, sep = ""))

# Column binding
b1 <- matrix(1:9, nrow = 3, dimnames = list(1:3, letters[1:3]))
b2 <- matrix(10:18, nrow = 3, dimnames = list(4:6, letters[4:6]))
b3 <- matrix(19:27, nrow = 3, dimnames = list(7:9, letters[7:9]))

qbind(paste("b", 1:3, sep = ""), type = "col")

# Concatenating a vector
a1 <- c(x = 1, y = 2)

```

```
a2 <- c(z = 3, w = 4)
qbind(c("a1", "a2"), type = "c")
```

rma	<i>Remove all objects from the global environment</i>
-----	---

Description

An alias for `rm(list = ls())`.

Usage

```
rma()
```

Author(s)

Landon Sego

select	<i>Select rows or columns from data frames or matrices while always returning a data frame or a matrix</i>
--------	--

Description

The primary contribution of this function is that if a single row or column is selected, the object that is returned will be a matrix or a dataframe—and it will not be collapsed into a single vector, as is the usual behavior in R.

Usage

```
select(data, selection, cols = TRUE)
```

Arguments

data	A matrix or dataframe from whence data will be selected
selection	A character vector with column (or row) names, or, a numeric vector with column (or row) indexes.
cols	A logical, that when TRUE, indicates that columns will be selected. If FALSE, rows will be selected.

Details

Selecting no rows or no columns is possible if `selection = 0` or if `length(selection) == 0`. In this case, a data frame or matrix with either 0 columns or 0 rows is returned.

Value

The matrix or data frame is returned with the selected columns or rows.

Author(s)

Landon Sego

Examples

```
# Consider this data frame
d <- data.frame(a = 1:5, b = rnorm(5), c = letters[1:5], d = factor(6:10),
               row.names = LETTERS[1:5], stringsAsFactors = FALSE)

# We get identical behavior when selecting more than one column
d1 <- d[, c("d", "c")]
d1c <- select(d, c("d", "c"))
d1
d1c
identical(d1, d1c)

# Different behavior when selecting a single column
d[, "a"]
select(d, "a")

# We can also select using numeric indexes
select(d, 1)

# Selecting a single row from a data frame produces results identical to default R behavior
d2 <- d[2,]
d2c <- select(d, "B", cols = FALSE)
identical(d2, d2c)

# Now consider a matrix
m <- matrix(rnorm(20), nrow = 4, dimnames = list(LETTERS[1:4], letters[1:5]))

# Column selection with two or more or more columns is equivalent to default R behavior
m1 <- m[,c(4, 3)]
m1c <- select(m, c("d", "c"))
m1
m1c
identical(m1, m1c)

# Selecting a single column returns a matrix of 1 column instead of a vector
m[,2]
select(m, 2)

# Selecting a single row returns a matrix of 1 row instead of a vector
m2 <- m["C",]
m2c <- select(m, "C", cols = FALSE)
m2
m2c
is.matrix(m2)
```

```
is.matrix(m2c)

# Selecting no rows or no columns occurs if 0 or an object of length 0
# is passed to 'selection'
select(d, 0)
select(d, which("bizarre" %in% colnames(d)))
select(d, 0, cols = FALSE)
```

selectElements *Validate selected elements from a character vector*

Description

Validate selected elements from a character vector using a variety of selection mechanisms: logical, names, or numerical indexes

Usage

```
selectElements(elements, cVec)
```

Arguments

elements	elements to select from cVec. Can be a logical vector, a vector of numeric indexes, or a character vector of element names. Note that logical vectors are not recycled as usual: they must be the same length as cVec. If elements == NULL, NULL is returned.
cVec	A character vector from which to select elements (such as row names or column names)

Details

This function is especially useful for selecting rows or columns from data frames, while providing informative error messages if the elements for selection are specified incorrectly.

Value

A character vector with elements that were selected from cVec using elements

Examples

```
# Define some "column names"
cnames <- letters[1:5]
cnames

# Select the 1st and 3rd column names using a variety of approaches
selectElements(c("a", "c"), cnames)
selectElements(c(1, 3), cnames)
selectElements(c(TRUE, FALSE, TRUE, FALSE, FALSE), cnames)
```



```
# Select the 1st, 3rd, and 1st columns
selectElements(c("a", "c", "a"), cnames)
selectElements(c(1, 3, 1), cnames)

# If you don't want to select any of them
selectElements(NULL, cnames)
```

sepList

Separate a list into distinct objects

Description

Separate a list into distinct objects

Usage

```
sepList(X, envir = parent.frame(), objNames = names(X), verbose = FALSE)
```

Arguments

X	a list to be separated
envir	The environment where the objects in the list are assigned, defaults to <code>parent.frame()</code> , the environment where <code>sepList</code> was called.
objNames	Character vector indicating the names of the objects that will be created. The length of names must match the length of the list, X.
verbose	Logical indicating whether to print the names of the objects that have been created by splitting the list

Details

The names of the objects are determined by the names in the list. If names in the list are not present, the objects are named o1, o2, o3, etc.

Value

Invisibly returns a character vector of the names of the objects that were created, and assigns the objects to the environment specified by `envir`

Author(s)

Landon Segó

Examples

```

# Simplest way to use sepList()
aList <- list(a = 1:10, b = letters[1:5], d = TRUE)
sepList(aList)
ls()
a
b
d

# Keeping the object names, and listing them via "verbose"
objs <- sepList(list(1:5, c("bits", "bytes"), c(TRUE, FALSE)), verbose = TRUE)
objs
o1
o2
o3

# Note that it doesn't recurse into sublists, only the top level object
# a and b are created
sepList(list(a = 1:2, b = list(b1 = 5, b2 = FALSE)), verbose = TRUE)

# Separate the original list inside a function, notice where the objects are written
sepTest <- function(x) {

  # Keep objects inside the local environment
  cat("Objects in the local environment before separating the list:\n")
  print(ls())

  sepList(x)

  cat("Objects in the local environment after separating the list:\n")
  print(ls())

  # Place objects in the global environment instead
  cat("Objects in the global environment before separating the list:\n")
  print(ls(.GlobalEnv))

  sepList(x, envir = .GlobalEnv)

  cat("Objects in the local environment after separating the list:\n")
  print(ls(.GlobalEnv))

} # sepTest

sepTest(list(z1 = 10, z2 = "that"))

# Clean up example objects
rm(aList, a, b, d, objs, o1, o2, o3, sepTest, z1, z2)

```

Description

Generic for signal

Usage

```
signal(object, ...)
```

Arguments

object	The object on which the generic operates
...	Arguments passed to specific methods

smartFilter	<i>Calculate a moving dot product (or filter) over a numeric vector</i>
-------------	---

Description

Calculate a moving dot product over a vector (typically a time series). It dynamically accounts for the incomplete windows which are caused by missing values and which occur at the beginning and end of the series. It does not propagate NAs.

Usage

```
smartFilter(y, weights, min.window = 1, start = 1, skip = 1,
            balance = TRUE)
```

Arguments

y	A numeric vector (can be labeled)
weights	Vector of weights that will be used to calculate the moving dot product. Should be odd in length and should sum to unity.
min.window	The minimum number of non-missing data points in a window that are required to calculate the dot product
start	The index of the center of the first window
skip	The number of indexes to advance the center of the moving window each time the dot product is calculated.
balance	= TRUE requires that the first non-missing value in a window occur on or before the center point of the window, and that the last non-missing value occur on or after the center point of the window.

Details

smartFilter has very similar behavior to [filter](#), except it calculates at the edge of a series and it does not propagate NAs which may be imbedded within the series.

When the window contains missing values, either due to being at the edge of the series or due to NAs imbedded within the series, the weights corresponding to the non-missing data points are re-normalized and the dotproduct is calculated using the available data. If the number of non-missing data points in the window is less than `min.window`, an NA is produced for the corresponding index. Likewise, if `balance = TRUE`, and the required conditions (described above in the argument description of `balance`) are not met, an NA is returned for the corresponding index.

Value

Returns the moving dot product

Author(s)

Landon Sego

See Also

[movAvg2](#), [filter](#)

Examples

```
# Define a simple vector
x <- 2^(0:8)
names(x) <- letters[1:9]

# Define weights for a simple moving average of 3 points
# (1 point in the past, the present point, and 1 point in the future)
wts <- rep(1, 3) / 3

# Note how they are the same, except at the edges of the series.
smartFilter(x, wts)
filter(x, wts)

# filter() and smartFilter() apply the weights in reverse order of each other,
# which makes a difference if the weights are not symmetric. Note how these
# two statements produce the same result (with the exception of the first and
# last elements)
filter(x, 1:3 / 6)
smartFilter(x, 3:1 / 6)

# Notice how filter() propagates missing values
y <- 3^(0:8)
y[5] <- NA
smartFilter(y, wts)
filter(y, wts)

# Compare starting on the second value and skip every other point
```

```

smartFilter(x, wts)
smartFilter(x, wts, start = 2, skip = 2)

# Demonstrate how the 'min.window' and 'balance' work
y <- round(rnorm(1:20),2)
names(y) <- letters[1:20]
y[7:9] <- NA
y
smartFilter(y, rep(1,5)/5, min.window = 2, balance = TRUE)
smartFilter(y, rep(1,5)/5, min.window = 2, balance = FALSE)

```

smartRbindMat

Row bind matrices whose column names may not be the same

Description

Row bind matrices whose column names may not be the same

Usage

```
smartRbindMat(..., distinguish = FALSE, filler = NA)
```

Arguments

...	matrix objects (separated by commas), a list of matrices, or a character vector containing the names of matrix objects
distinguish	if TRUE, then rownames of the returned matrix are assigned a name consisting of the source object name as a prefix, followed by the row name, separated by a ":". Otherwise, the original rownames are used.
filler	The character to insert into the final matrix for those empty elements which occur when not all the matrices have the same column names.

Value

Produces a matrix with a union of the column names. Empty elements resulting from different column names are set to the value of filler.

Author(s)

Landon Segó

See Also

[rbind](#)

Examples

```

x <- matrix(rnorm(6), ncol = 2, dimnames = list(letters[1:3], letters[4:5]))
y <- matrix(rnorm(6), ncol = 3, dimnames = list(letters[7:8], letters[4:6]))
z <- matrix(rnorm(2), nrow = 1, dimnames = list("c", letters[3:4]))

x
y
z

smartRbindMat(x,y,z)
smartRbindMat(list(x, y, z), distinguish = TRUE)
smartRbindMat(y,z,x, distinguish = TRUE)
smartRbindMat(c("y","x","z"), filler = -20, distinguish = TRUE)

w1 <- matrix(sample(letters[1:26], 6), ncol = 2,
             dimnames = list(c("3", "", "4"), c("w", "v")))
x1 <- matrix(sample(letters[1:26], 6), ncol = 2,
             dimnames = list(NULL, letters[4:5]))
y1 <- matrix(sample(letters[1:26], 6), ncol = 3,
             dimnames = list(NULL, letters[4:6]))
z1 <- matrix(sample(letters[1:26], 2), nrow = 1,
             dimnames = list(NULL, letters[3:4]))

w1
x1
y1
z1

smartRbindMat(w1,x1,y1,z1)
smartRbindMat(list(w1 = w1, x1 = x1, y1 = y1, z1 = z1), distinguish = TRUE)

smartRbindMat(w1,x1,y,z1,z)
smartRbindMat(w1,x1,y,z1,z, distinguish = TRUE)

```

smartTimeAxis

Produces a time axis with smart spacing

Description

Produces a time axis on a plot with interval spacing units that are intuitive. It is intended to be applied to periods of time that do not exceed 24 hours (i.e. it does not produce a date stamp in the time axis).

Usage

```

smartTimeAxis(time.vec, nticks = 15, side = 1, time.format = c("hh:mm",
"hh:mm:sspm", "hh:mm:ss pm", "hh:mm:ss", "hh:mmpm", "hh:mm pm"))

```

Arguments

<code>time.vec</code>	A time object (vector) that was used to construct the plot, presumed to be ordered chronologically
<code>nticks</code>	The target number of ticks to use in the axis
<code>side</code>	Same as the <code>side</code> argument in axis
<code>time.format</code>	Character string indicating the time format to display on the axis. The choices are displayed in the Usage. Defaults to <code>hh:mm</code> .

Details

`smartTimeAxis` attempts to choose a "natural" spacing for the time axis ticks that results in the number of ticks being as close as possible to `nticks`. Possibilities for natural spacings include 1, 5, 10, 15 seconds, etc., or 1, 2, 5, 10, minutes etc., or 0.5, 1, 1.5 hours, etc.

Value

Places the axis on the plot.

Author(s)

Landon Sego

See Also

[axis.POSIXct](#)

Examples

```
# Get data and set the options to the horizontal axis labels will be
# oriented vertically
data(timeData)
op <- par(las = 2, mfrow = c(3, 1), mar = c(4, 4, 2, 0.5))

# Make the default plot
plot(timeData, xlab = "", main = "Default intervals")

# Make the plot with specialized time axis
plot(timeData, axes = FALSE, frame.plot = TRUE, xlab = "", main = "10 minute intervals")

# Add y-axis
axis(2)

# Add the time axis
smartTimeAxis(timeData$time, nticks = 10)

# Only look at a small portion of the data with a different time format
par(mar = c(7, 4, 2, 0.5))

plot(timeData[200:237,], type = "b", axes = FALSE, frame.plot = TRUE,
      xlab = "", main = "15 second intervals")
```

```
axis(2)

smartTimeAxis(timeData[200:237,"time"], nticks = 20, time.format = "hh:mm:ss pm")

# Restore the original par settings
par(op)
```

sourceDir

Sources all files with '.R' or '.r' extensions in a directory

Description

Sources all files with '.R' or '.r' extensions in a directory using a try-catch for each file

Usage

```
sourceDir(directory, recursive = FALSE, tryCatch = TRUE, ...)
```

Arguments

directory	Character string indicating the path of the directory containing the R files to be sourced.
recursive	=TRUE descends into subdirectories of directory
tryCatch	if TRUE, sourcing is protected in a try catch, i.e., if there is an error, sourceDir will continue to the next file. If FALSE, sourceDir will stop if a file sources with an error.
...	Additional arguments to source

Details

In addition to sourcing files for general use, this function is also useful in package development to verify there are no syntax errors prior to building and compilation.

Value

Invisibly returns a character vector containing the files that were identified for sourcing. Also prints a message indicating whether each file was sourced correctly or not.

Author(s)

Landon Sego

stopifnotMsg	<i>Check multiple conditions and return corresponding error messages</i>
--------------	--

Description

A more flexible version of `stopifnot` that allows you to control the error message returned by each condition that doesn't test true

Usage

```
stopifnotMsg(..., level = 1)
```

Arguments

...	Pairs of logical conditions and error messages. See Examples.
level	Whole number indicating how far back in the call stack the error should be attributed to. <code>level = 1</code> goes back to the calling function, <code>level = 2</code> goes back 2 levels, etc. See examples.

Value

A call to `stop` with the error messages from each condition that was FALSE. If all conditions are TRUE, returns NULL.

Author(s)

Landon Segio

Examples

```
# A simple function
aFunction <- function(x, a = 10, b = "text") {

  # Check the arguments of the function
  stopifnotMsg(is.numeric(x), "'x' must be numeric",
              is.numeric(a), "'a' must be numeric",
              a > 9, "'a' must be 10 or more",
              is.character(b), "'b' must be character")

  return(paste(x, a, b, sep = " -- "))
}

# This should run without error
aFunction(12, a = 13, b = "new")

## Not run:
# This should produce an error with 3 messages:
aFunction("new", a = 7, b = 5)
```

```

# And this should produce an error with 1 message:
aFunction(33, a = "bad")

## End(Not run)

### This illustrates how the 'level' argument works

# A check function that will be called within another
check <- function(a, lev) {
  stopifnotMsg(is.numeric(a), "a must be numeric", level = lev)
}

# A function that uses the check.
f <- function(a, lev = 1) {
  check(a, lev)
  return(a + 10)
}

## Not run:
# Note how the error is attributed to 'check'
f("a")

# But if we change the level to 2, the error will be attributed to 'f'
f("a", lev = 2)

## End(Not run)

```

stripExtension	<i>Remove the extension of a vector of filenames</i>
----------------	--

Description

Remove the extension of a vector of filenames, assuming that the extension is the set of characters that follows the last "."

Usage

```
stripExtension(vec, split.char = ".")
```

Arguments

vec	Character vector (usually containing filenames)
split.char	A single character used to split the character strings

Details

Assumes paths are delineated using forward slashes. If an NA is supplied, then an NA is returned. If the desired string doesn't exist (see examples below), a "" is returned.

Value

Character vector with the last "." and the filename extension removed. Alternatively, another split character could be used.

Author(s)

Landon Sego

See Also

Additional functions for filename manipulations: [getExtension](#), [stripPath](#), [getPath](#), [grabLast](#), [basename](#), [dirname](#)

Examples

```
stripExtension(c("this old file.doc", "that young file.rtf",
                "this.good.file.doc", "this_bad_file"))

stripExtension(c("this old file*doc", "that young file*rtf",
                "this*good*file*doc", "this_bad_file"), split.char = "*")

# Named vectors are not required, but are included here to make the
# output easier to read. This example demonstrates a number of
# pathological cases.
stripExtension(c(a = NA, b = ".doc", c = "this.pdf", d = "this.file.", e = ".",
                f = "noExtension", g = "direc.name/filename.txt", h = ""))

# An example with 'real' files
files <- dir(file.path(path.package(package = "Smisc"), "data"), full.names = TRUE)
print(files)
stripExtension(files)
stripExtension(stripPath(files))
```

stripPath

Remove the path from a vector of filenames

Description

Remove the path from a vector of filenames

Usage

```
stripPath(vec)
```

Arguments

vec Character vector (usually containing filenames)

Details

Assumes paths are delineated using forward slashes. If an NA is supplied, then an NA is returned. If the desired string doesn't exist (see examples below), a "" is returned.

Value

Character vector with leading path removed from the filenames

Author(s)

Landon Sego

See Also

Additional functions for filename manipulations: [stripExtension](#), [getExtension](#), [getPath](#), [grabLast](#), [basename](#), [dirname](#)

Examples

```
stripPath(c(a = "this.good.path/filename.R", b = "nopath.R", c = "/", d = NA,
            e = "only.paths.1/only.paths.2/", ""))

# An example with 'real' files
files <- dir(file.path(path.package(package = "Smisc"), "data"), full.names = TRUE)
print(files)
stripPath(files)
stripPath(stripExtension(files))
```

timeData

Generic data frame with a time variable

Description

Generic data frame with a time variable to support the example in [smartTimeAxis](#).

Format

A data frame with 414 observations on the following 2 variables.

time a POSIXt time variable

x a numeric vector

Examples

```
data(timeData)
```

timeDiff	<i>Subtracts two time series by matching irregular time indexes</i>
----------	---

Description

Subtracts two time series by matching irregular time indexes. Can also be used to align the indexes of a time series to a set of standard time indexes.

Usage

```
timeDiff(v1, v2, n.ind = 1, full = FALSE)
```

Arguments

v1	A time series vector: vector with dates or datetimes for names which are non repeating and in chronological order
v2	Another time series vector
n.ind	An integer ≥ 1 that indicates how many elements of the longer of the two vectors will be averaged and matched to the closest timestamp of the shorter vector. See details below.
full	=TRUE returns a data frame that shows in detail how the two vectors were matched and the difference calculated

Details

The format for the timestamps (in the vector names) can be virtually any reasonable format, which will be converted to POSIXct by [formatDT](#).

Suppose that v1 is shorter than v2. For each index of the v1, the n.ind indices of v2 that are closest in time to the v1 index are identified and "matched" (by averaging them) to the v1 index. In the case of ties, for example, the nearest v2 indexes are both 5 seconds before and 5 seconds after the v1 index of interest, then the closest v2 index in the past (5 seconds before) is matched to the v1 index.

Hence, the average of the n.ind elements of v2 that best "match" (are closest in time to) the element of v1 are subtracted from v1, which creates a series of differences with the same length (and timestamps) as v1.

If instead, v2 is shorter than v1, then the time stamps of v2 become the 'standard' to which the times of v1 are matched.

If v1 and v2 are the same length, then the timestamps of v2 are matched to v1 and the resulting vector of differences has the same timestamps as v1.

Value

if full = FALSE then the difference (after matching) of (v1 - v2) is returned. Otherwise, a data frame is returned that shows how the vectors were matched and the resulting difference vector.

Author(s)

Landon Sego

Examples

```

data(timeDiff.eg)

# Show the objects
print(timeDiff.eg)

# Extract the objects from the list for easier use in the example
sepList(timeDiff.eg)

# Print warnings as they occur
op <- options(warn = 1)

# Show various differences
timeDiff(x1, x2, full = TRUE)
timeDiff(x2.d, x1.d, full = TRUE)
timeDiff(x1, x1)

options(op)

### If we need to average a time-series at 30 second intervals:

# Create the vector that will be averaged, with time stamps occurring
# about every 10 seconds
v1.names <- seq(formatDT("2009-09-12 3:20:31")$dt.posix,
                formatDT("2009-09-12 3:29:15")$dt.posix, by = 10)

# Now jitter the times a bit and look at the time spacing
v1.names <- v1.names + round(rnorm(length(v1.names), sd = 1.5))
diff(v1.names)

# Create the vector
v1 <- abs(rnorm(length(v1.names), mean = 7, sd = 3))
names(v1) <- v1.names

# Now create a standard vector with values of 0 with time stamps every 30 seconds
standard.names <- seq(formatDT("2009-09-12 3:21:30")$dt.posix,
                    formatDT("2009-09-12 3:28:30")$dt.posix, by = 30)
standard <- double(length(standard.names))
names(standard) <- standard.names

# Now average the v1 values by matching the 3 closest values to each standard time:
timeDiff(v1, standard, n.ind = 3, full = TRUE)
v1.avg <- timeDiff(v1, standard, n.ind = 3)

# Check that every 3 obs were averaged
v1.avg.check <- tapply(v1[6:50], rep(1:15, each = 3), mean)
max(abs(v1.avg.check - v1.avg))

```

timeDiff.eg	<i>Four short time series</i>
-------------	-------------------------------

Description

Four short times series, stored in a list, for use in the `timeDiff` example.

Examples

```
data(timeDiff.eg)
```

timeIntegration	<i>Approximate the integral of a vector of data over time</i>
-----------------	---

Description

Integrate a series over time by calculating the area under the "curve" of the linear interpolation of the series (akin to the Trapezoid rule). This is especially useful in calculating energy usage: kilowatt-hours, watt-seconds, etc.

Usage

```
timeIntegration(data, time = names(data), lower = time[1],
  upper = time[length(time)], check.plot = FALSE, units = c("hours",
  "minutes", "seconds"))
```

Arguments

<code>data</code>	Vector of numerical data
<code>time</code>	Vector of timestamps which correspond to data. These can either character or POSIXct.
<code>lower</code>	The time (character or POSIXct) of the lower bound of the integration
<code>upper</code>	The time (character or POSIXct) of the upper bound of the integration
<code>check.plot</code>	=TRUE makes a plot which illustrates the integration.
<code>units</code>	The units of integration, defaults to hours. It is only required to supply enough characters to uniquely complete the name.

Details

If `upper` or `lower` does not correspond to a data point, a linear interpolation is made between the two neighboring time points to predict the resulting data value.

Value

The approximation of the integral by joining the points in the series in a linear fashion and calculating the area under this "curve".

Author(s)

Landon Sego

Examples

```
# Some example power data
data(PowerData)

par(mfrow = c(2, 1))

# Calculate the kilowatt-minutes, display graph which shows how the
# integration is done. This example calculates the integral using
# a contiguous subset of the data
int1 <- timeIntegration(PowerData,
                        # Convert to POSIXct in order to subtract time
                        lower = "5/6/2008 17:00:09",
                        upper = "5/6/2008 17:01:36",
                        check.plot = TRUE, units = "m")

# This example calculates the integral for all the data in 'powerData'
int2 <- timeIntegration(PowerData, check.plot = TRUE, units = "m")

# Print the outcome
pvar(int1, int2)
```

timeIt

Times the execution of an expression

Description

Times the execution of an expression

Usage

```
timeIt(expr, units = c("automatic", "seconds", "minutes", "hours", "days"),
       return.time = FALSE, verbose = TRUE)
```

Arguments

expr Any R [expression](#).

units A character expression long enough to uniquely identify one of "automatic", "seconds", "minutes", or "hours". Defaults to "automatic".

`return.time` = TRUE returns the elapsed time as one of the elements in a list. See "Value" below.

`verbose` = TRUE prints the elapsed time in the requested units.

Details

If `units = "automatic"`, then the units are chosen according to the following rule: If the duration is < 2 min, use seconds. Else if duration < 2 hours, use minutes. Else if < 2 days, use hours. Otherwise, use days.

Value

If `return.time = FALSE`, invisibly returns the evaluation of `expr`. If `return.time = TRUE`, invisibly returns a list with the following components:

<code>out</code>	The evaluation of <code>expr</code>
<code>elapsed</code>	The elapsed time to evaluate <code>expr</code>
<code>units</code>	The time units of the elapsed time

Author(s)

Landon Sego

See Also

[proc.time](#)

Examples

```
# We can assign the object within the call to timeIt():
timeIt(x1 <- rnorm(10^6))
str(x1)

# We can just run the expression without assigning it to anything
timeIt(rnorm(10^7), units = "m")

# Or we can assign the result externally
x2 <- timeIt(rnorm(10^7))
str(x2)

# To store the elapsed time:
x3 <- timeIt(rnorm(10^7), verbose = FALSE, return.time = TRUE)
x3[c("elapsed", "units")]
```

timeStamp *Embeds the present datetime into a file name*

Description

Embeds the present datetime into a file name

Usage

```
timeStamp(description, extension)
```

Arguments

description	Character vector giving the base name(s) of the file(s)
extension	Character vector giving the extension(s) of the file(s) (excluding the period)

Value

Character string of the form description_YYYY-MM-DD_HHMMSS.extension

Author(s)

Landon Sego

Examples

```
timeStamp("aFilename", "txt")
```

umvueLN *Computes UMVUEs of lognormal parameters*

Description

Computes uniformly minimum variance unbiased (UMVU) estimates of the mean, the standard error of the mean, and the standard deviation of lognormally distributed data.

Usage

```
umvueLN(x, tol = 1e-15, verbose = FALSE)
```

Arguments

x	Vector of lognormal data
tol	Tolerance level for convergence of the infinite series, <i>Psi</i> . Convergence occurs when the absolute value of the current term in the series is less than tol.
verbose	Logical indicating whether iteration steps for convergence of <i>Psi</i> are printed.

Details

Calculates equations 13.3, 13.5, and 13.6 of Gilbert (1987).

Value

Returns a named vector with the following components

mu	The UMVUE of the mean
se.mu	The UMVUE standard error of the mean
sigma	The UMVUE of the standard deviation

Author(s)

Landon Sego

References

Gilbert, Richard O. (1987) Statistical Methods for Environmental Pollution Monitoring, John Wiley & Sons, Inc. New York, pp 164-167.

Examples

```
# Test from Gilbert 1987, Example 13.1, p 166
x <- c(3.161, 4.151, 3.756, 2.202, 1.535, 20.76, 8.42, 7.81, 2.72, 4.43)
y <- umvueLN(x)
print(y, digits = 8)

# Compare to results from PRO-UCL 4.00.02:

# MVU Estimate of Mean          5.6544289
# MVU Estimate of Standard Error of Mean  1.3944504
# MVU Estimate of SD           4.4486438

# Compare these to Gilbert's printed results (which have rounding error)
Gilbert <- c(5.66, sqrt(1.97), sqrt(19.8))
print(round(abs(y - Gilbert), 2))
```

 vertErrorBar

Draw vertical error bar(s) on a plot

Description

Draw vertical error bar(s) on a plot

Usage

```
vertErrorBar(x, width, center = NULL, barLength = NULL, min.y = NULL,
  max.y = NULL, blankMiddle = NULL, ...)
```

Arguments

<code>x</code>	Vector of x value on the plot around which the vertical error bar will be drawn
<code>width</code>	The total width of the cross hatches on top and bottom of the bars, can be a vector or a single value
<code>center</code>	Vector of values designating the vertical center of the error bars, can be a vector or a single value
<code>barLength</code>	The total vertical length of the bars, can be a vector or a single value
<code>min.y</code>	Vector of values indicating the vertical bottoms of the bars
<code>max.y</code>	Vector of values indicating the vertical tops of the bars
<code>blankMiddle</code>	the vertical length of a blank spot that will be produced in the middle of the error bar. This is useful when the bar is placed around symbols (so as not to overwrite them). Defaults to NULL, in which case a solid error bar is drawn. Can also be a vector or a single value.
<code>...</code>	additional arguments to lines .

Details

Buyer beware! It's up to the user to determine what the statistically correct length of the error bar should be.

Either `center` and `barLength` must be specified, or `min.y` and `max.y` must be specified.

Note that `width`, `center`, `barLength`, `min.y`, `max.y`, and `blankMiddle` should be NULL, numeric values of length 1, or numeric values with the same length as `x`. If they have the same length of `x`, the bars can have different vertical lengths, widths, and `blankMiddle` values if desired.

Value

Nothing is returned, the error bar(s) is/are drawn on the plot

Author(s)

Landon Sego

Examples

```
set.seed(343)

# Make a plot of some standard normal observations
x <- 1:9
y <- rnorm(9)

plot(x, y, pch = as.character(1:9), ylim = c(-2, 2) + range(y),
     ylab = "Z", xlab = "Indexes")

# Draw the error bars
vertErrorBar(x, 0.3, center = y, barLength = 2 * 1.96, blankMiddle = 0.25)
```

Index

- *Topic **datasets**
 - PowerData, 51
 - timeData, 68
 - timeDiff.eg, 71
- *Topic **manip**
 - factor2character, 19
- *Topic **math**
 - integ, 30
- *Topic **misc**
 - allMissing, 3
 - as.numericSilent, 4
 - comboList, 5
 - cumMax, 6
 - cumsumNA, 7
 - dataIn, 9
 - df2list, 11
 - dfplapply, 12
 - dframeEquiv, 14
 - dkbinom, 16
 - factor2character, 19
 - findDepMat, 21
 - formatDT, 22
 - getExtension, 24
 - getPath, 25
 - grabLast, 26
 - movAvg2, 36
 - openDevice, 38
 - padZero, 39
 - pcbinom, 43
 - plapply, 46
 - pvar, 51
 - rma, 54
 - smartFilter, 59
 - smartRbindMat, 61
 - smartTimeAxis, 62
 - sourceDir, 64
 - stripExtension, 66
 - stripPath, 67
 - timeDiff, 69
 - timeIntegration, 71
 - timeIt, 72
 - timeStamp, 74
 - umvueLN, 74
- *Topic **package**
 - Smisc-package, 3
- abline, 28
- all.equal, 15
- allMissing, 3
- as.data.frame, 33
- as.list, 11
- as.numeric, 5
- as.numericSilent, 4, 32
- axis, 63
- axis.POSIXct, 63
- basename, 24–26, 67, 68
- bmp, 38
- c, 53
- cat, 27
- cbind, 53
- clusterEvalQ, 40
- clusterExport, 40
- combn, 5
- comboList, 5
- complete.cases, 4
- cumMax, 6
- cumsum, 7
- cumsumNA, 7
- cusum, 8
- data, 10
- dataIn, 9
- dbinom, 17
- df2list, 11
- dfplapply, 12, 48
- dframeEquiv, 14
- dirname, 24–26, 67, 68

dkbinom, 16
 do.call, 18
 doCallParallel, 18, 28, 50
 duplicated, 21

 expression, 72

 factor2character, 19, 32
 factor2numeric, 20
 filter, 37, 60
 findDepMat, 21
 foreach::foreach, 45
 formatDT, 22, 69

 getExtension, 24, 25, 26, 67, 68
 getPath, 24, 25, 26, 67, 68
 grabLast, 24, 25, 26, 67, 68

 hardCode, 27
 hpd, 28

 identical, 15
 integ, 30
 integrate, 30

 jpeg, 38

 lapply, 12, 41, 46–48
 legend, 50
 library, 12, 47
 linearMap, 31
 lines, 50, 76
 list2df, 11, 32
 load, 34
 loadObject, 10, 34

 mclapply, 46, 48
 more, 35
 movAvg2, 36, 60

 openDevice, 38

 padZero, 39
 par, 50
 parLapply, 40, 41, 46, 48
 parLapplyW, 5, 18, 40, 48
 parseJob, 13, 41, 47
 pbeta, 43
 pbinom, 17, 43
 pcbinom, 43

 pddply, 44
 pdf, 38
 pkbinom (dkbinom), 16
 plapply, 12, 13, 41, 46
 plot.cusum (cusum), 8
 plot.default, 8, 28, 36, 50
 plot.hpd (hpd), 28
 plot.movAvg2 (movAvg2), 36
 plotFun, 28, 49
 plyr::create_progress_bar, 44
 plyr::ddply, 44, 45
 png, 38
 postscript, 38
 PowerData, 51
 print.cusum (cusum), 8
 print.default, 8, 36
 print.hpd (hpd), 28
 print.movAvg2 (movAvg2), 36
 proc.time, 73
 pvar, 51

 qbind, 53
 qr, 21

 rbind, 53, 61
 read.csv, 10
 readLines, 35
 rma, 54
 round, 52

 select, 54
 selectElements, 56
 sepList, 57
 signal, 58
 signal.cusum (cusum), 8
 smartFilter, 36, 37, 59
 smartRbindMat, 61
 smartTimeAxis, 62, 68
 Smisc (Smisc-package), 3
 Smisc-package, 3
 source, 64
 sourceDir, 64
 stop, 65
 stopifnot, 65
 stopifnotMsg, 65
 stripExtension, 24–26, 66, 68
 stripPath, 24–26, 67, 67

 tiff, 38

timeData, [68](#)
timeDiff, [69](#), [71](#)
timeDiff.eg, [71](#)
timeIntegration, [51](#), [71](#)
timeIt, [72](#)
timeStamp, [74](#)

umvueLN, [74](#)

vertErrorBar, [75](#)