Package ‘rje’

February 20, 2015

Type Package
Title Miscellaneous useful functions
Version 1.9
Date 2014-04-22
Author Robin Evans <evans@stats.ox.ac.uk>
Maintainer Robin Evans <evans@stats.ox.ac.uk>
Description A series of useful functions, some available in different
forms in other packages, but which have been extended, sped up, or
otherwise modified in some way considered useful to the author.
Depends R (>= 2.0.0)
License GPL (>= 2)
LazyLoad yes
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-08-06 19:18:00

R topics documented:

rje-package .................................................. 2
and0 .......................................................... 2
armijo ......................................................... 4
arrayInd ...................................................... 5
combinations ................................................. 7
conditionTable .............................................. 8
cubeHelix .................................................... 10
Dirichlet ..................................................... 12
expit .......................................................... 13
fsapply ....................................................... 14
greaterThan ............................................... 15
indexBox ..................................................... 16
interventionTable ......................................... 17
is.subset ................................................... 18
rje-package

Miscellaneous useful functions

Description

A series of useful functions, some available in different forms in other packages, but which have been extended, sped up, or otherwise modified in some way considered useful to the author.

Details

Package: rje
Type: Package
Version: 1.9
Date: 2014-04-22
License: GPL (>= 2)
LazyLoad: yes

Author(s)

Robin Evans
Mathias Drton
Maintainer: Robin Evans <evans@stats.ox.ac.uk>

Fast pairwise logical operators
**Description**

Fast but loose implementations of AND and OR logical operators.

**Usage**

```r
and0(x, y)
or0(x,y)
```

**Arguments**

`x, y`  
A logical or numeric vector.

**Details**

Returns pairwise application of logical operators AND and OR. Vectors are recycled as usual.

**Value**

A logical vector of length `\max(\text{length}(x), \text{length}(y))` with entries `x[1] \& x[2]` etc.; each entry of `x` or `y` is `TRUE` if it is non-zero.

**Note**

These functions should only be used with well understood vectors, and may not deal with unusual cases correctly.

**Author(s)**

Robin Evans

**Examples**

```r
and0(c(0,1,0), c(1,1,0))
```

```r
## Not run:
set.seed(1234)
x = rbinom(5000, 1, 0.5)
y = rbinom(5000, 1, 0.5)

# 3 to 4 times improvement over `&`
system.time(for (i in 1:5000) and0(x,y))
system.time(for (i in 1:5000) x & y)
```

```r
## End(Not run)
```
armijo

Generic functions to aid finding local minima given search direction

Description

Allows use of an Armijo rule or coarse line search as part of minimisation (or maximisation) of a differentiable function of multiple arguments (via gradient descent or similar). Repeated application of one of these rules should (hopefully) lead to a local minimum.

Usage

armijo(fun, x, dx, beta = 3, sigma = 0.5, grad,
       maximise = FALSE, searchup = TRUE, adj.start = 1, ...)
coarseLine(fun, x, dx, beta = 3, maximise = FALSE, ...)

Arguments

fun a function whose first argument is a numeric vector
x a starting value to be passed to fun
dx numeric vector containing feasible direction for search; defaults to \(-\text{grad}\) for ordinary gradient descent
beta numeric value (greater than 1) giving factor by which to adjust step size
sigma numeric value (less than 1) giving steepness criterion for move
grad numeric gradient of \(f\) at \(x\) (will be estimated if not provided)
maximise logical: if set to TRUE search is for a maximum rather than a minimum.
searchup logical: if set to TRUE method will try to find largest move satisfying Armijo criterion, rather than just accepting the first it sees
adj.start an initial adjustment factor for the step size.
... other arguments to be passed to fun

Details

coarseLine performs a stepwise search and tries to find the integer \(k\) minimising \(f(x_k)\) where

\[ x_k = x + \beta^k dx. \]

Note \(k\) may be negative.

armijo implements an Armijo rule for moving, which is to say that

\[ f(x_k) - f(x) < -\sigma\beta^k dx \cdot \nabla_x f. \]

This has better convergence guarantees than a simple line search, but may be slower in practice. See Bertsekas (1999) for theory underlying the Armijo rule.

Each of these rules should be applied repeatedly to achieve convergence (see example below).
arrayInd

**Value**

- `best`: the value of the function at the final point of evaluation
- `adj`: the constant in the step, i.e. $\beta^n$
- `move`: the final move; i.e. $\beta^n dx$
- `code`: an integer indicating the result of the function: 0 = returned OK, 1 = very small move suggested, may be at minimum already, 2 = failed to find minimum: function evaluated to NA or was always larger than $f(x)$ (direction might be infeasible), 3 = failed to find minimum: stepsize became too small or large without satisfying rule.

**Author(s)**

Robin Evans

**References**


**Examples**

```r
# minimisation of simple function of three variables
x = c(0,-2,4)
f = function(x) ((x[1]-3)^2 + x[2]*sin(x[2]) + exp(x[3]) - x[3])
tol = .Machine$double.eps
mv = 1

while (mv > tol) {
  # or replace with coarseline()
  out = armijo(f, x, sigma=0.1)
  x = out$x
  mv = sum(out$move^2)
}

# correct solution is c(3,0,0)
x
```

---

**arrayInd**

Faster calculation of array indices from vector position.

**Description**

Calculates array indices based on their vector position in an array.
arrayInd

Usage

arrayInd(ind, .dim, .dimnames = NULL, useNames = FALSE)

Arguments

ind  integer-valued vector of indices.
.dim  integer vector giving dimensions of array.
.dimnames  optional list of character \texttt{dimnames(.)}, of which only .dimnames[1] is used.
useNames  logical indicating if the value of \texttt{arrayInd()} should have (non-null) dimnames at all.

Details

This is a C implementation of the base function of the same name. Results should be the same.

Given a vector of integers giving the vector position of entries in an array, returns the appropriate array indices.

Value

A matrix whose rows each are the indices of one element of \texttt{x}; see Examples below.

Author(s)

Robin Evans

See Also

The base version, documented as \texttt{which}.

Examples

\begin{verbatim}
arr = array(1:36, dim=c(2,3,2,3))
ind = arrayInd(c(4,9,17), c(2,3,2,3))
ind

arr[2,2,1,1]
arr[1,2,2,1]
arr[1,3,1,2]
\end{verbatim}
Descripción

Returns a matrix containing each possible combination of one entry from vectors of the lengths provided.

Uso

combinations(p)
powersetmat(n)

Argumentos

- p: vector of non-negative integers.
- n: non-negative integer.

Detalles

Returns a matrix, each row being one possible combination of integers from the vectors \((0, 1, \ldots, p_i - 1)\), for \(i\) between 1 and length(p).

Based on bincombinations from package e1071, which provides the binary case.

powersetmat is just a wrapper for combinations(rep(2, n)).

Valor

A matrix with number of columns equal to the length of p, and number of rows equal to \(p_1 \times \cdots \times p_k\), each row corresponding to a different combination. Ordering is reverse-lexographic.

Autores

Robin Evans

Ejemplos

combinations(c(2,3,3))
powersetmat(3)
**conditionTable**  
*Find conditional probability table*

### Description

Given a numeric array or matrix (of probabilities), calculates margins of some dimensions conditional on particular values of others.

### Usage

- `conditionTable(x, variables, condition = NULL, condition.value = NULL)`
- `conditionTable2(x, variables, condition)`
- `conditionMatrix(x, variables, condition = NULL, condition.value = NULL, dim = NULL, incols = FALSE)`
- `marginMatrix(x, margin, dim = NULL, incols = FALSE, reorder = FALSE)`
- `condition.table(x, margin, dim = NULL, incols = FALSE, reorder = FALSE, reorder = FALSE)`
- `condition.table2(x, variables, condition = NULL, condition.value = NULL)`
- `condition.table2(x, variables, condition)`

### Arguments

- **x**: A numeric array.
- **variables**: An integer vector containing the margins of interest from `x`.
- **margin**: An integer vector containing the margins of interest from `x`.
- **condition**: An integer vector containing the dimensions of `x` to condition on.
- **condition.value**: An integer vector or list of the same length as `condition`, containing the values to condition with. If NULL, then the full conditional distribution is returned.
- **dim**: Integer vector containing dimensions of variables. Assumed all binary if not specified.
- **incols**: Logical specifying whether not the distributions are stored as the columns in the matrix; assumed to be rows by default.
- **reorder**: Logical specifying whether the variables should be ordered as specified in the vector `margin`, or (if false) kept in their original order. Defaults to false as this is faster.

### Details

`conditionTable` calculates the marginal distribution over the dimensions in `variables` for each specified value of the dimensions in `condition`. Single or multiple values of each dimension in `condition` may be specified in `condition.value`; in the case of multiple values, `condition.value` must be a list.

The sum over the dimensions in `variables` is normalized to 1 for each value of `condition`. 
conditionTable2 is just a wrapper which returns the conditional distribution as an array of the same dimensions and ordering as the original x. Values are repeated as necessary.

conditionMatrix takes a matrix whose rows (or columns if incols = TRUE) each represent a separate multivariate probability distribution and finds the relevant conditional distribution in each case. These are then returned in the same format. The order of the variables under conditionMatrix is always as in the original distribution, unlike for conditionTable above.

marginMatrix is the equivalent function for performing the role of margin.table or the faster marginTable. In both these functions the probabilities are assumed in lexicographic order, i.e. the first value changes fastest: (1,1,1), (2,1,1), (1,2,1), ..., (2,2,2).

condition.table and condition.table2 are identical to conditionTable and conditionTable2.

**Value**

conditionTable returns an array whose first length(variables) corresponds to the dimensions in variables, and the remainder (if any) to dimensions in condition with a corresponding entry in condition.value of length > 1.

conditionTable2 always returns an array of the same dimensions as x, with the variables in the same order.

**Author(s)**

Mathias Drton, Robin Evans

**See Also**

marginTable, margin.table, interventionTable

**Examples**

```r
x = array(1:16, rep(2,4))
x = x/sum(x) # probability distribution on 4 binary variables x1, x2, x3, x4.

# distribution of x2, x3 given x1 = 1 and x4=2.
conditionTable(x, c(2,3), c(1,4), c(1,2))
# x2, x3 given x1 = 1,2 and x4 = 2.
conditionTable(x, c(2,3), c(1,4), list(1:2,2))

# complete conditional of x2, x3 given x1, x4
conditionTable(x, c(2,3), c(1,4))

# conditionTable2 leaves dimensions unchanged
tmp = conditionTable2(x, c(2,3), c(1,4))
aperm(tmp, c(2,3,1,4))

####
set.seed(2314)
# set of 10 2x2x2 probability distributions
x = rdirichlet(10, rep(1,8))
marginMatrix(x, c(1,3))
```
cubeHelix

Description

Cube Helix is a colour scheme designed to be appropriate for screen display of intensity images. The scheme is intended to be monotonically increasing in brightness when displayed in greyscale. This might also provide improved visualisation for colour blindness sufferers.

Usage

cubeHelix(n, start = 0.5, r = -1.5, hue = 1, gamma = 1)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>integer giving the number of colours in the scale</td>
</tr>
<tr>
<td>start</td>
<td>numeric: start gives the initial angle (in radians) of the helix</td>
</tr>
<tr>
<td>r</td>
<td>numeric: number of rotations of the helix over the scale; can be negative</td>
</tr>
<tr>
<td>hue</td>
<td>numeric controlling the saturation of colour: 0 gives pure greyscale, defaults to 1</td>
</tr>
<tr>
<td>gamma</td>
<td>numeric which can be used to emphasise lower or higher intensity values, defaults to 1</td>
</tr>
</tbody>
</table>

Details

The function evaluates a helix which moves through the RGB "cube", beginning at black (0,0,0) and finishing at white (1,1,1). Evenly spaced points on this helix in the cube are returned as RGB colours. This provides a colour palette in which intensity increases monotonically, which makes for good transfer to greyscale displays or printouts. This also may have advantages for colour blindness sufferers. See references for further details.

Value

Vector of RGB colours (strings) of length n.

Author(s)

Dave Green
Robin Evans
References


See Dave Green’s page at http://www.mrao.cam.ac.uk/~dag/cubehelix/ for other details.

See Also

rainbow (for other colour palettes).

Examples

cubehelix(21)

## Not run:
cols = cubehelix(101)

plot.new()
plot.window(xlim=c(0,1), ylim=c(0,1))
axis(side=1)
for (i in 1:101) {
  rect((i-1)/101,0,(i+0.1)/101,1, col=cols[i], lwd=0)
}

## End(Not run)

## Not run:
require(grDevices)
# comparison with other palettes
n = 101
cols = cubehelix(n)
heat = heat.colors(n)
rain = rainbow(n)
terr = terrain.colors(n)

plot.new()
plot.window(xlim=c(-0.5,1), ylim=c(0,4))
axis(side=1, at=c(0,1))
axis(side=2, at=1:4-0.5, labels=1:4, pos=0)
for (i in 1:n) {
  rect((i-1)/n,3,(i+0.1)/n,3.9, col=cols[i], lwd=0)
  rect((i-1)/n,2,(i+0.1)/n,2.9, col=heat[i], lwd=0)
  rect((i-1)/n,1,(i+0.1)/n,1.9, col=rain[i], lwd=0)
  rect((i-1)/n,0,(i+0.1)/n,0.9, col=terr[i], lwd=0)
}
legend(-0.6,4,legend=c("4. cube helix", "3. heat", "2. rainbow", "1. terrain"), box.lwd=0)

## End(Not run)
Dirichlet

The Dirichlet Distribution

Description

Density function and random generation for Dirichlet distribution with parameter vector alpha.

Usage

```
rdirichlet(n, alpha)
ddirichlet(x, alpha, log = FALSE, tol = 1e-10)
```

Arguments

- `n`: number of random variables to be generated.
- `alpha`: vector of Dirichlet hyper parameter.
- `x`: vector (or matrix) of points in sample space.
- `log`: logical; if TRUE, natural logarithm of density is returned.
- `tol`: tolerance of vectors not summing to 1 and negative values.

Details

If `x` is a matrix, each row is taken to be a different point whose density is to be evaluated. If the number of columns in (or length of, in the case of a vector) `x` is one less than the length of `alpha`, the remaining column (or entry) is assumed to make the vector sum to 1.

The $k$-dimensional Dirichlet distribution has density

$$
\frac{\Gamma \left( \sum_i \alpha_i \right) \prod_i \Gamma (\alpha_i)}{\prod_i x_i^{\alpha_i - 1}}
$$

assuming that $x_i > 0$ and $\sum_i x_i = 1$, and zero otherwise.

If the sum of row entries in `x` differs from 1 by more than `tol`, or any entry takes a value less than -`tol`, the density is assumed to be zero.

Value

- `rdirichlet` returns a matrix, each row of which is an independent draw from a Dirichlet distribution with parameter vector `alpha`.
- `ddirichlet` returns a vector, each entry being the density of the corresponding row of `x`. If `x` is a vector, then the output will have length 1.

Author(s)

Robin Evans
expit

References


Examples

x = rdirichlet(10, c(1,2,3))
x

# Find densities at random points.
ddirichlet(x, c(1,2,3))
# Last column to be inferred.
ddirichlet(x[,c(1,2)], c(1,2,3))

expit  Expit and Logit.

Description

Functions to take the expit and logit of numerical vectors.

Usage

expit(x)
logit(x)

Arguments

x  vector of real numbers; for logit to return a sensible value these should be between 0 and 1.

Details

logit implements the usual logit function, which is

\[ \text{logit}(x) = \log \frac{x}{1 - x}, \]

and expit its inverse:

\[ \text{expit}(x) = \frac{e^x}{1 + e^x}. \]

It is assumed that logit(0) = -Inf and logit(1) = Inf, and correspondingly for expit.

Value

A real vector corresponding to the expits or logits of \( x \)

Warning

Choosing very large (positive or negative) values to apply to expit may result in inaccurate inversion (see example below).
Author(s)

Robin Evans

Examples

\[
\begin{align*}
  x &= \text{c}(5, -2, 0.1) \\
  y &= \text{expit}(x) \\
  \text{logit}(y)
\end{align*}
\]

# Beware large values!
\[\text{logit}(\expit(100))\]

---

**fsapply**

*Fast and loose application of function over list.*

Description

Faster highly stripped down version of `sapply()`

Usage

`fsapply(x, FUN)`

Arguments

- `x` a vector (atomic or list) or an expression object.
- `FUN` the function to be applied to each element of `x`. In the case of functions like `+`, the function name must be backquoted or quoted.

Details

This is just a wrapper for `unlist(lapply(x, FUN))`, which will behave as `sapply` if `FUN` returns an atomic vector of length 1 each time.

Speed up over `sapply` is not dramatic, but can be useful in time critical code.

Value

A vector of results of applying `FUN` to `x`.

Warning

Very loose version of `sapply` which should really only by used if you’re confident about how `FUN` is applied to each entry in `x`.

Author(s)

Robin Evans
greaterThan

Examples

```r
x = list(1:1000)
tmp = fsapply(x, sin)

## Not run:
x = list()
set.seed(142313)
for (i in 1:1000) x[[i]] = rnorm(100)

system.time(for (i in 1:100) sapply(x, function(x) last(x)))
system.time(for (i in 1:100) fsapply(x, function(x) last(x)))

## End(Not run)
```

---

greaterThan

Comparing numerical values

Description

Just a wrapper for comparing numerical values, for use with quicksort.

Usage

```
greaterThan(x, y)
```

Arguments

- `x`: A numeric vector.
- `y`: A numeric vector.

Details

Just returns -1 if `x` is less than `y`, 1 if `x` is greater, and 0 if they are equal (according to `==`). The vectors wrap as usual if they are of different lengths.

Value

An integer vector.

Author(s)

Robin Evans

See Also

`<` for traditional Boolean operator.
Examples

greaterThan(4,6)

# Use in sorting algorithm.
quickSort(c(5,2,9,7,6), f=greaterThan)
order(c(5,2,9,7,6))

indexBox

Get indices of adjacent entries in array

Description

Determines the relative vector positions of entries which are adjacent in an array.

Usage

indexBox(upp, lwr, dim)

Arguments

upp     A vector of non-negative integers, giving the distance in the positive direction from the centre in each co-ordinate.
lwr     A vector of non-positive integers, giving the negative distance from the centre.
dim     integer vector of array dimensions.

Details

Given a particular cell in an array, which are the entries within (for example) 1 unit in any direction?
This function gives the (relative) value of such indices. See examples.
Indices may be repeated if the range exceeds the size of the array in any dimension.

Value

An integer vector giving relative positions of the indices.

Author(s)

Robin Evans

See Also

arrayInd.
Examples

```r
arr = array(1:144, dim=c(3,4,3,4))
arr[2,2,3]
# which are entries within 1 unit each each direction of 2,2,2,3?

inds = 89 + indexBox(1,-1,c(3,4,3,4))
inds = inds[inds > 0 & inds <= 144]
arrayInd(inds, c(3,4,3,4))

inds = 89 + indexBox(c(0,1,0,0),c(0,-1,0,0),c(3,4,3,4))
inds = inds[inds > 0 & inds <= 144]
arrayInd(inds, c(3,4,3,4))
```

---

**interventionTable**  
*Calculate interventional distributions.*

**Description**

Calculate interventional distributions from a probability table or matrix of multivariate probability distributions.

**Usage**

```r
interventionTable(x, variables, condition)
interventionMatrix(x, variables, condition, dim=NULL, incols=FALSE)
intervention.table(x, variables, condition)
```

**Arguments**

- **x**  
  An array of probabilities.

- **variables**  
  The margin for the intervention.

- **condition**  
  The dimensions to be conditioned upon.

- **dim**  
  Integer vector containing dimensions of variables. Assumed all binary if not specified.

- **incols**  
  Logical specifying whether not the distributions are stored as the columns in the matrix; assumed to be rows by default.

**Details**

This just divides the joint distribution \( p(x) \) by \( p(v|c) \), where \( v \) is variables and \( c \) is condition. Under certain causal assumptions this is the interventional distribution \( p(x \mid do(v)) \) (i.e. if the direct causes of \( v \) are precisely \( c \)).

`intervention.table()` is identical to `interventionTable()`.
is.subset

Value
A numerical array of the same dimension as \( x \).

Author(s)
Robin Evans

References

See Also
conditionTable, marginTable

Examples
```r
set.seed(413)
# matrix of distributions
p = rdirichlet(10, rep(1, 16))
interventionMatrix(p, 3, 2)

# take one in an array
ap = array(p[1,], rep(2, 4))
interventionTable(ap, 3, 2)
```

---

**is.subset**  
*Check subset inclusion*

**Description**
Determines whether one vector contains all the elements of another.

**Usage**
```r
is.subset(x, y)
```

**Arguments**
- \( x \)  
  vector.
- \( y \)  
  vector.

**Details**
Determines whether or not every element of \( x \) is also found in \( y \). Returns TRUE if so, and FALSE if not.
is.wholenumber

Value

A logical of length 1.

Author(s)

Robin Evans

See Also

setmatch.

Examples

isNsubsetHQ:RL Q:SI
isNsubsetHQ:RL R:SI

is.wholenumber

Determine whether number is integral or not.

Description

Checks whether a numeric value is integral, up to machine or other specified precision.

Usage

is.wholenumber(x, tol = .Machine$double.eps*0.5)

Arguments

x numeric vector to be tested.

tol The desired precision.

Value

A logical vector of the same length as x, containing the results of the test.

Author(s)

Robin Evans

Examples

x = c(0.5, 1, 2L, 1e-20)
is.wholenumber(x)
Description

Returns the last element of a list or vector.

Usage

`last(x)`

Arguments

- `x`: a list or vector.

Details

Designed to be faster than using `tail()` or `rev()`, and cleaner than writing `x[length(x)]`.

Value

An object of the same type as `x` of length 1 (or empty if `x` is empty).

Author(s)

Robin Evans

See Also

`tai`, `rev`.

Examples

`last(1:10)`

---

Description

For a contingency table in array form, compute the sum of table entries for a given index.

Usage

```
marginTable(x, margin = NULL, order = TRUE)
propTable(x, margin = NULL)
```
patternRepeat

Arguments

- x: a numeric array
- margin: vector of index numbers (1 for rows, etc.)
- order: logical indicating whether indices of output should be ordered as in the vector margin?

Details

With order = TRUE this is the same as the base function margin.table(), but faster.

With order = FALSE the function is even faster, but the indices in the margin are returned in their original order, regardless of the way they are specified in margin.

propTable() is equivalent to prop.table(), but faster.

Value

The relevant marginal table. The class of 'x' is copied to the output table, except in the summation case.

Note

Original functions are margin.table and prop.table.

Author(s)

Robin Evans

Examples

m <- matrix(1:4, 2)
marginTable(m, 1)
marginTable(m, 2)

propTable(m, 2)

patternRepeat

Complex repetitions

Description

Recreate patterns for collapsed arrays

Usage

patternRepeat(x, which, n, careful = TRUE)
Arguments

x A vector to be repeated
which Which indices of the implicit array are given in x
n Dimensions of implicit array.
careful logical indicating whether to check validity of other arguments or work quickly
(mainly for recursion)

Details

This function allows for complex repeating patterns. Consider an array with dimensions n; then for each value of the dimensions in which (contained in x), this function returns the pattern of values in which for each entry in the larger array.
Hence the length of x must be equal to prod(n[which]).

Value

A vector of length prod(n) containing suitably repeated and ordered elements of x.

Author(s)

Robin Evans

See Also

rep

Examples

patternRepeat(1:4, c(1,2), c(2,2,2))
c(array(1:4, c(2,2,2)))

patternRepeat(1:4, c(1,3), c(2,2,2))
patternRepeat(1:4, c(2,3), c(2,2,2))

describe powerSet

powerSet Power Set

Description

Produces the power set of a vector.

Usage

powerSet(x, rev = FALSE)
Arguments

x  vector of elements (the set).
rev logical indicating whether to reverse the order of subsets.

Details

Creates a list containing every subset of the elements of the vector x.

Value

A list of vectors of the same type as x.

With \( \text{rev} = \text{FALSE} \) (the default) the list is ordered such that all subsets containing the last element of x come after those which do not, and so on.

Author(s)
Robin Evans

See Also
powerSetMat.

Examples

\[
\text{powerset}(1:3) \\
\text{powerset}(\text{letters}[3:5], \text{rev}=\text{TRUE})
\]

printPercentage  \hspace{1cm} Print Percentage of Activity Completed to stdout

Description

Prints percentage (or alternatively just a count) of loop or similar process which has been completed to the standard output.

Usage

\[
\text{printPercentage}(i, n, dp = 0, \text{first} = 1, \text{last} = n, \text{prev} = i - 1) \\
\text{printCount}(i, \text{first} = 1, \text{prev} = i - 1, \text{last} = \text{NULL})
\]

Arguments

i  the number of iterations completed.
n  total number of iterations.
dp number of decimal places to display.
first number of the first iteration for which this percentage was displayed
last number of the final iteration for which this percentage will be displayed
prev number of the previous iteration for which this percentage was displayed
Details

printPercentage will use cat to print the proportion of loops which have been completed (i.e. \( i/n \)) to the standard output. In doing so it will erase the previous such percentage, except when \( i = \text{first} \). A new line is added when \( i = \text{last} \), assuming that the loop is finished.

Value

NULL

Warning

This will fail to work nicely if other information is printed to the standard output during the process.

Author(s)

Robin Evans

Examples

\[ x = \text{numeric}(100) \]

\[
\text{for} \ (i \ \text{in} \ 1:100) \ {
\text{x[i]} = \text{mean(rnorm(1e5))}
\text{printPercentage(i,100)}
}\]

\[ i = 0 \]

\text{repeat} \ {
\text{i} = i+1
\text{if} \ (\text{runif(1)} > 0.99) \ {
\text{break}
}\text{printCount(i)}
}\text{print("\n")}

quickSort  Quicksort for Partial Orderings

Description

Implements the quicksort algorithm for partial orderings based on pairwise comparisons.

Usage

\[ \text{quickSort}(x, f = \text{greaterThan}, \text{random} = \text{TRUE}) \]
**Arguments**

- `x`: A list or vector of items to be sorted.
- `f`: A function on two arguments for comparing elements of `x`. Returns `-1` if the first argument is less than the second, `1` for the reverse, and `0` if they are equal or incomparable.
- `random`: logical - should a random pivot be chosen? (this is recommended) Otherwise middle element is used.

**Details**

Implements the usual quicksort algorithm, but may return the same positions for items which are incomparable (or equal). Does not test the validity of `f` as a partial order.

If `x` is a numeric vector with distinct entries, this behaves just like `order`.

**Value**

Returns an integer vector giving each element’s position in the order (minimal element(s) is 1, etc).

**Warning**

Output may not be consistent for certain partial orderings (using random pivot), see example below. All results will be consistent with a total ordering which is itself consistent with the true partial ordering.

`f` is not checked to see that it returns a legitimate partial order, so results may be meaningless if it is not.

**Author(s)**

Robin Evans

**References**


**See Also**

- `order`.

**Examples**

```r
set.seed(1)
quickSort(powerSet(1:3), f=subsetOrder)
quickSort(powerSet(1:3), f=subsetOrder)
# slightly different answers, but both corresponding
# to a legitimate total ordering.
```
Description

Series of functions extending existing vector operations to lists of vectors.

Usage

```r
setmatch(x, y, nomatch = NA_integer_)
subsetmatch(x, y, nomatch = NA_integer_)
setsetdiff(x, y)
setsetequal(x, y)
```

Arguments

- **x**: list of vectors.
- **y**: list of vectors.
- **nomatch**: value to be returned in the case when no match is found. Note that it is coerced to integer.

Details

- `setmatch` checks whether each vector in the list `x` is also contained in the list `y`, and if so returns position of the first such vector in `y`. The ordering of the elements of the vector is irrelevant, as they are considered to be sets.
- `subsetmatch` is similar to `setmatch`, except vectors in `x` are searched to see if they are subsets of vectors in `y`.
- `setsetdiff` is a setwise version of `setdiff`, and `setsetequal` a setwise version of `setequal`.

Value

- `setmatch` and `subsetmatch` return a vector of integers of length the same as the list `x`.
- `setsetdiff` returns a sublist `x`.
- `setsetequal` returns a logical of length 1.

Note

These functions are not recursive, in the sense that they cannot be used to test lists of lists. They also do not reduce to the vector case.

Author(s)

Robin Evans
See Also

match, setequal, setdiff

Examples

\[
\begin{align*}
x & = \text{list}(1:2, 1:3) \\
y & = \text{list}(1:4, 1:3) \\
\text{setmatch}(x, y) & \\
\text{subsetmatch}(x, y) & \\
\text{setsetdiff}(x, y) & \\
x & = \text{list}(1:3, 1:2) \\
y & = \text{list}(2:1, c(2,1,3)) \\
\text{setsetequal}(x, y) & 
\end{align*}
\]

Description

Produces a matrix whose rows indicate what subsets of a set are included in which other subsets.

Usage

subsetMatrix(n)

Arguments

n integer containing the number of elements in the set.

Details

This function returns a matrix, with each row and column corresponding to a subset of a hypothetical set of size \( n \), ordered lexicographically. The entry in row \( i \), column \( j \) corresponds to whether or not the subset associated with \( i \) is a superset of that associated with \( j \).

A 1 or -1 indicates that \( i \) is a superset of \( j \), with the sign referring to the number of fewer elements in \( j \). 0 indicates that \( i \) is not a superset of \( j \).

Value

An integer matrix of dimension \( 2^n \times 2^n \).

Note

The inverse of the output matrix is just \( \text{abs}(\text{subsetMatrix}(n)) \).

Author(s)

Robin Evans
See Also

`combinations`, `powerSet`.

Examples

```r
subsetMatrix(3)
```
Description

More flexible calls of [ on an array.

Usage

subarray(x, levels, drop = TRUE)
subtable(x, variables, levels, drop = TRUE)
'subarray<=~'(x, levels, value)
'subtable<=~'(x, variables, levels, value)

Arguments

x An array.
variables An integer vector containing the dimensions of x to subset.
levels A list or vector containing values to retain.
drop Logical indicating whether dimensions with only 1 retained should be dropped. Defaults to TRUE.
value Value to assign to entries in table.

Details

Essentially just allows more flexible calls of [ on an array.

subarray requires the values for each dimension should be specified, so for a 2 x 2 x 2 array x,
subarray(x, list(1,2,1:2)) is just x[1,2,1:2].

subtable allows unspecified dimensions to be retained automatically. Thus, for example subtable(x, c(2,3), list(1, 1)
is x[,1,1:2].

Value

Returns an array of dimension sapply(value, length) if drop=TRUE, otherwise specified dimensions of size 1 are dropped. Dimensions which are unspecified in subtable are never dropped.

Author(s)

Mathias Drton, Robin Evans

See Also

Extract
Examples

x = array(1:8, rep(2,3))
subarray(x, c(2,1,2)) == x[2,1,2]

x[2,1:2,2,drop=FALSE]
subarray(x, list(2,1:2,2), drop=FALSE)

subtable(x, c(2,3), list(1, 1:2))
Index

*Topic **IO**
  printPercentage, 23
*Topic **arith**
  combinations, 7
  expit, 13
  greaterThan, 15
  interventionTable, 17
  is.subset, 18
  is.wholenumber, 19
  powerSet, 22
  quickSort, 24
  rje-package, 2
  setmatch, 26
  subsetMatrix, 27
  subsetOrder, 28
*Topic **array**
  arrayInd, 5
  conditionTable, 8
  indexBox, 16
  marginTable, 20
  patternRepeat, 21
  rje-package, 2
  subtable, 29
*Topic **color**
  cubeHelix, 10
*Topic **distribution**
  Dirichlet, 12
*Topic **iteration**
  printPercentage, 23
*Topic **list**
  fsapply, 14
*Topic **logic**
  and0, 2
*Topic **manip**
  last, 20
*Topic **optimize**
  armijo, 4
  quickSort, 24
  rje-package, 2

*Topic **package**
  rje-package, 2
*Topic **print**
  printPercentage, 23
  %subof%(is.subset), 18
  and0, 2
  armijo, 4
  arrayInd, 5, 16
  coarseLine (armijo), 4
  combinations, 7, 28
  condition.table (conditionTable), 8
  condition.table2 (conditionTable), 8
  conditionMatrix (conditionTable), 8
  conditionTable, 8, 18
  conditionTable2 (conditionTable), 8
  cubeHelix, 10
  ddirichlet (Dirichlet), 12
  dimnames, 6
  Dirichlet, 12
  expit, 13
  Extract, 29
  fsapply, 14
  greaterThan, 15
  indexBox, 16
  intervention.table (interventionTable), 17
  interventionMatrix (interventionTable), 17
  interventionTable, 9, 17
  interventionTable, 16
  is.subset, 18, 28
  is.wholenumber, 19
  last, 20
  logit (expit), 13
margin.table, 9, 21
marginMatrix (conditionTable), 8
marginTable, 9, 18, 20
match, 27

or0 (and0), 2
order, 25

patternRepeat, 21
powerSet, 22, 28
powerSetMat, 23
powerSetMat (combinations), 7
printCount (printPercentage), 23
printPercentage, 23
prop.table, 21
propTable (marginTable), 20

quickSort, 24

rainbow, 11
rdirichlet (Dirichlet), 12
rep, 22
rev, 20
rje (rje-package), 2
rje-package, 2

setdiff, 27
setequal, 27
setmatch, 19, 26
setsetdiff (setmatch), 26
setsetequal (setmatch), 26
subarray (subtable), 29
subarray<-(subtable), 29
subsetmatch (setmatch), 26
subsetMatrix, 27
subsetOrder, 28
subtable, 29
subtable<-(subtable), 29

tail, 20

which, 6