Package ‘sfsmisc’

June 8, 2017

Title Utilities from 'Seminar fuer Statistik' ETH Zurich
Version 1.1-1
Date 2017-06-08
Author Martin Maechler et al.
Maintainer Martin Maechler <maechler@stat.math.ethz.ch>
Description Useful utilities ['goodies'] from Seminar fuer Statistik ETH
Zurich, quite a few related to graphics; some were ported from S-plus.
Depends R (>= 3.0.1)
Imports grDevices, methods, utils, stats
Suggests datasets, tcltk, cluster, lattice, MASS, Matrix, nlme, lokern
Enhances mgcv, rpart, nor1mix, polycor, sm, tikzDevice
Encoding latin1
ByteCompile yes
License GPL (>= 2)
NeedsCompilation no
Repository CRAN
Date/Publication 2017-06-08 16:46:33 UTC

R topics documented:

AsciiToInt .................................................. 3
axTexp .................................................... 5
cairoSwd ................................................... 6
capture.and.write ......................................... 7
col01scale ................................................ 8
compresid2way ............................................ 9
cum.Vert.funkt ........................................... 11
D1D2 ....................................................... 12
D2ss ....................................................... 14
Deprecated ................................................ 16
diagDA ..................................................... 16
R topics documented:

diagX ................................................................. 18
digitsBase .......................................................... 19
Duplicated ......................................................... 21
eaxis ................................................................. 22
ecdf.ksCI ............................................................. 25
eEllipsePoints .................................................... 26
empty_dimnames ................................................... 27
erbar ................................................................. 28
f.robftest ........................................................... 29
factorize ............................................................. 30
hatMat ............................................................... 31
histBxp ............................................................... 32
integrate.xy ......................................................... 34
inv.seq .............................................................. 35
is.whole ............................................................ 36
iterate.lin.recursion ............................................. 37
KSd ................................................................. 38
last ................................................................. 39
linesHyperb.lm ................................................... 40
loessDemo .......................................................... 41
lseq ................................................................. 43
mat2tex ............................................................ 44
missingCh .......................................................... 45
mpl ................................................................. 46
mult.fig ............................................................ 47
n.code ............................................................. 49
n.plot ............................................................. 50
nearcor ............................................................. 51
nr.sign.chg ........................................................ 53
p.arrows ........................................................... 54
p.datum ............................................................ 54
p.dnorm ........................................................... 55
p.hboxp ............................................................ 56
p.profileTraces .................................................. 57
p.res.2fact ........................................................ 58
p.res.2x ........................................................... 59
p.scales ............................................................ 61
p.tachoPlot ......................................................... 61
p.ts ................................................................. 63
paste.vec .......................................................... 64
plotDS ............................................................. 65
plotStep ........................................................... 67
polyn.eval ........................................................ 68
posdefify .......................................................... 69
potatoes ........................................................... 71
pretty10exp ....................................................... 72
primes ............................................................. 74
printTable2 ........................................................ 75
## AsciToInt

**AsciiToInt** returns **integer** codes in 0:255 for each (one byte) character in strings. **ichar** is an alias for it, for old S compatibility.

**strcodes** implements in **R** the basic engine for translating characters to corresponding integer codes.
chars8bit() is the inverse function of AscciToIn, producing “one byte” characters from integer codes. Note that it (and hence strcodes()) depends on the locale, see Sys.getlocale()

Usage

AsciiToInt(strings)
  iChar(strings)
chars8bit(i = 1:255)
strcodes(x, table = chars8bit(1:255))

Arguments

strings, x       character vector.
i             numeric (integer) vector of values in 1:255.
table       a vector of (unique) character strings, typically of one character each.

Details

Only codes in 1:127 make up the ASCII encoding which should be identical for all R versions, whereas the 'upper' half is often determined from the ISO-8859-1 (aka “ISO-Latin 1”) encoding, but may well differ, depending on the locale setting, see also Sys.setlocale.

Note that 0 is no longer allowed since, R does not allow \0 aka nul characters in a string anymore.

Value

AsciiToInt (and hence iChar) and chars8bit return a vector of the same length as their argument. strcodes(x, tab) returns a list of the same length and names as x with list components of integer vectors with codes in 1:255.

Author(s)

Martin Maechler, partly in 1991 for S-plus

Examples

chars8bit(65:70)#-> "A" "B" .. "F"
stopifnot(identical(LETTERS, chars8bit(65:90)),
identical(AscciToInt(LETTERS), 65:90))

## may only work in ISO-latin1 locale (not in UTF-8):
try( strcodes(c(a= "ABC", ch="1234", place = "Zürich")) )
## in "latin-1" gives {otherwise should give NA instead of 252}:
## Not run:
$a
[1] 65 66 67

$ch
[1] 49 50 51 52
axTexpr

Axis Ticks Expressions in Nice 10 ** k Form

Description

Produce nice $a \times 10^k$ expressions for axis labeling instead of the scientific notation "$a \times 10^k$".

Usage

axTexpr(side, at = axTicks(side, axp = axp, usr = usr, log = log),
        axp = NULL, usr = NULL, log = NULL,
        drop.1 = FALSE)

Arguments

side integer in 1:4 specifying the axis side, as for axis.
at numeric vector; with identical default as in axTicks().
axp, usr, log as for axTicks().
drop.1 logical indicating if $1 \times$ should be dropped from the resulting expressions.

Details

This is just a utility with the same arguments as axTicks, a wrapper pretty10exp(at, *).
Value

an expression of the same length as x, with elements of the form a%% 10 ^ k.

Author(s)

Martin Maechler

See Also

pretty10exp; eaxis, axis, axticks.

Examples

```r
x <- 1e7*(-10:50)
y <- dnorm(x, m=10e7, s=20e7)
plot(x,y)## not really nice, the following is better:

## For horizontal y-axis labels, need more space:
op <- par(mar = c(5,4,1))
plot(x,y, axes= FALSE, frame=TRUE)
aX <- axticks(1); axis(1, at=aX, label= aTexpr(1, aX))
## horizontal labels on y-axis:
aY <- axticks(2); axis(2, at=aY, label= aTexpr(2, aY), las=2)
par(op)

### -- only 'x' and using log-scale there:
plot(x,y, xaxt= "n", log= "x")
aX <- axticks(1); axis(1, at=aX, label= aTexpr(1, aX))

### Now an "engineer's version" ( more ticks; only label "10 ^ k" ) :

axp <- par("xaxp") --> powers of 10 *inside* 'usr'
axp[3] <- 1 # such that only 10^*. are labeled
aX <- axticks(1, axp = axp)
xu <- 10 ^ par("usr")[1:2]
e10 <- c(-1,1) + round(log10(axp[1:2])) ## exponents of 10 *outside* 'usr'
v <- c(outer(1:9, e10[1]:e10[2], function(x,E) x * 10 ^ E))
v <- v[xu[1] <= v & v <= xu[2]]

plot(x,y, xaxt= "n", log= "x", main= "engineer's version of x - axis")
axis(1, at = aX, label = aTexpr(1, aX, drop.1=TRUE)) # 'default'
axis(1, at = v, label = FALSE, tcl = 2/3 * par("tcl"))
```

cairoSwd

Cairo PDF Graphics Device useful for Sweave

Description

Provides a graphics device for Sweave, based on cairo_pdf. The advantage of cairoSwd() compared to pdf() is its support of Unicode characters.
**capture.and.write**

**Usage**

```r
cairoSwd(name, width, height, ...)
```

**Arguments**

- `name`: file name prefix to which `.pdf` will be appended.
- `width, height`: in inches, see `cairo_pdf`.
- `...`: further arguments, passed to `cairo_pdf()`.

**Note**

Sweave devices need to have an argument list as above.

Usage in a Sweave chunk:

```latex
<<some-plot, fig=TRUE, grdevice=cairoSwd>>=
```

**Author(s)**

Alain Hauser

**See Also**

`pdf`, `cairo_pdf`, `Sweave`.

---

**Description**

Capture output and print first and last parts, eliding middle parts. Particularly useful for teaching purposes, and, e.g., in Sweave (`RweaveLatex`).

By default, when `middle = NA`, `capture.output(EXPR, first, last)` basically does

```r
co <- capture.output(EXPR)
writelines(head(co, first))
cat(... dotdots ...)writelines(tail(co, last))
```

**Usage**

```r
capture.and.write(EXPR, first, last = 2, middle = NA, i.middle, dotdots = ".......", n.dots = 2)
```
Arguments

EXPR     the (literal) expression the output of which is to be captured.
first    integer: how many lines should be printed at beginning.
last     integer: how many lines should be printed at the end.
middle   numeric (or NA logical):
i.middle index start of middle part
dotdots  string to be used for elided lines
n.dots   number of dotdots lines added between parts.

Value

return value of capture.output(EXPR).

Author(s)

Martin Maechler, ETH Zurich

See Also

head, tail

Examples

x <- seq(0, 10, by = .1)

## for matrix, dataframe, .. first lines include a header line:
capture.and.write( cbind(x, log1p(exp(x))), first = 5)

## first, *middle* and last :
capture.and.write( cbind(x, x^2, x^3), first = 4, middle = 3, n.dots= 1)

Matrix Scaling Utilities

col01scale and colcenter (re)scale the columns of a matrix. These are simple one-line utilities, mainly with a didactical purpose.

Usage

colcenter (mat)
col01scale(mat, scale.func = function(x) diff(range(x)), location.func = mean)
Arguments

mat numeric matrix, to rescaled.
scale.func, location.func
two functions mapping a numeric vector to a single number.

Value

a matrix with the same attributes as the input mat.

Author(s)

Martin Maechler

See Also

The standard R function `scale()`.

Examples

```r
## See the simple function definitions:
colcenter ## simply one line
col01scale# almost as simple
```

Description

For an analysis of variance or regression with (at least) two factors: Plot components + residuals for two factors according to Tukey’s “forget-it plot”. Try it!

Usage

```r
compresid2way(aov, data=NULL, fac=1:2, label = TRUE, numlabel = FALSE,
               xlab=NULL, ylab=NULL, main=NULL,
               col=c(2,3,4,4), lty=c(1,1,2,4), pch=c(1,2))
```

Arguments

aov either an `aov` object with a formula of the form
       \( y \sim a + b \), where \( a \) and \( b \) are factors, or such a formula.
data data frame containing \( a \) and \( b \).
fac the two factors used for plotting. Either column numbers or names for argument data.
label logical indicating if levels of factors should be shown in the plot.
numlabel  logical indicating if effects of factors will be shown in the plot.
xlab,ylab,main  the usual title components, here with a non-trivial default constructed from aov and the component factors used.
col,lty,pch  colors, line types, plotting characters to be used for plotting [1] positive residuals, [2] negative residuals, [3] grid, [4] labels. If pch is sufficiently long, it will be used as the list of individual symbols for plotting the y values.

Details

For a two-way analysis of variance, the plot shows the additive components of the fits for the two factors by the intersections of a grid, along with the residuals. The observed values of the target variable are identical to the vertical coordinate.

The application of the function has been extended to cover more complicated models. The components of the fit for two factors are shown as just described, and the residuals are added. The result is a “component plus residual” plot for two factors in one display.

Value

Invisibly, a list with components

compy  data.frame containing the component effects of the two factors, and combined effects plus residual

coeff  coefficients: Intercept and effects of the factors

Author(s)

Werner Stahel <stahel@stat.math.ethz.ch>

References


See Also

interaction.plot

Examples

```r
N <- c(0,1,0,1,1,0,0,0,1,1,0,1,0,0,1,0,1,1,0,0)
P <- c(1,1,0,0,1,0,1,1,0,0,0,1,1,0,1,0,0,1,0,0)
K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,1,1,0)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,55.0,
          62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                 K=factor(K), yield=yield)
npk.cr <- compresid2way(yield ~ N+P+K, data=npk, fac=c("P","K"))
```
cum.Vert.funkt

## Fisher's 1926 data on potato yield

data(potatoes)
pot.aov <- aov(yield ~ nitrogen+potash+pos, data=potatoes)
comresid2way(pot.aov, pch=as.character(potatoes$pos))

comresid2way(yield=nitrogen+potash, data=subset(potatoes, pos == 2))

## 2 x 3 design :
data(warpbreaks)
summary(fml <- aov(breaks ~ wool * tension, data = warpbreaks))
comresid2way(fml)

cum.Vert.funkt | Kumulative Verteilung Aufzeichnen

### Description

Kumulative Verteilung von x aufzeichnen, auf Wunsch auch Median und Quartile.

This is just an old German language version of plot.ecdf() used for teaching at ETHZ.

### Usage

```r
cum.Vert.funkt(x, Quartile = TRUE, titel = TRUE, Datum = TRUE,
                rang.axis = n <= 20, xlab = "", main = ", ...)
```

### Arguments

- **x**
  - numeric vector whose empirical distribution should be plotted.
- **Quartile**
  - logical indicating if all 3 non-trivial quartiles should be drawn.
- **titel**
  - logical indicating if a German title should be drawn.
- **Datum**
  - logical indicating if `p.datum` should be added.
- **rang.axis**
  - logical indicating if all the ranks should be marked at the y-axis. Defaults to true if there are not more than 20 observations.
- **xlab, main**
  - x-axis label and main title; default to empty.
- **...**
  - optional further arguments, passed to `plotStep`.

### Value

the return value of `plotStep()` which is called internally, *invisibly*.

### Author(s)

Martin Maechler et al.
See Also

plotStep on which it is based; but you should really consider using plot.ecdf() from the stats package instead of this.

Examples

cum.Vert.funkt(runif(12))
cum.Vert.funkt(runif(20))

Z <- rnorm(50)
cum.Vert.funkt(Z)

D1D2

Numerical Derivatives of (x,y) Data via Smoothing Splines

Description

Compute numerical derivatives of \( f() \) given observations \((x, y)\), using cubic smoothing splines with GCV, see smooth.spline. In other words, estimate \( f'(x) \) and/or \( f''(x) \) for the model

\[
Y_i = f(x_i) + E_i, \quad i = 1, \ldots, n,
\]

Usage

D1D2(x, y, xout = x, spar.offset = 0.1384, deriv = 1:2, spl.spar = NULL)

Arguments

- \( x, y \): numeric vectors of same length, supposedly from a model \( y \sim f(x) \).
- \( xout \): abscissa values at which to evaluate the derivatives.
- \( spar.offset \): numeric fudge added to the smoothing parameter, see spl.par below.
- \( deriv \): integer in 1:2 indicating which derivatives are to be computed.
- \( spl.spar \): direct smoothing parameter for smooth.spline. If it is NULL (as per default), the smoothing parameter used will be \( spar.offset + sp$spar \), where \( sp$spar \) is the GCV estimated smoothing parameter, see smooth.spline.

Details

It is well known that for derivative estimation, the optimal smoothing parameter is larger (more smoothing) than for the function itself. \( spar.offset \) is really just a fudge offset added to the smoothing parameter. Note that in R’s implementation of smooth.spline, \( spar \) is really on the log \( \lambda \) scale.

When \( deriv = 1:2 \) (as per default), both derivatives are estimated with the same smoothing parameter which is suboptimal for the single functions individually. Another possibility is to call D1D2(*, deriv = k) twice with \( k = 1 \) and \( k = 2 \) and use a larger smoothing parameter for the second derivative.
Value

- a list with several components,
  - \(x\) the abscissae values at which the derivative(s) are evaluated.
  - \(D1\) if \(deriv\) contains 1, estimated values of \(f'(x_i)\) where \(x_i\) are the values from \(xout\).
  - \(D2\) if \(deriv\) contains 2, estimated values of \(f''(x_i)\).
  - \(spar\) the smoothing parameter used in the (final) smooth.spline call.
  - \(df\) the equivalent degrees of freedom in that smooth.spline call.

Author(s)

Martin Maechler, in 1992 (for S).

See Also

- \(D2ss\) which calls smooth.spline twice, first on \(y\), then on the \(f'(x_i)\) values; smooth.spline on which it relies completely.

Examples

```r
d1d2 <- d1d2(x, y, spar.offset = off)
d2 <- d2(x, y)
spar <- d2$spar
df <- d2$df
dx <- d2$dx
xout <- d2$xout
f <- function(x) sin(x) + 0.1 * x

plot(x, y)
lines(smooth.spline(x, y), col = 4)
str(c("df", "spar"))
if(is.R()) plot(cos, 0, 10, ylim = c(-1.5, 1.5), lwd=2) else { # Splus
  xx <- seq(0,10, len=201); plot(xx, cos(xx), type = 'l', ylim = c(-1.5,1.5))
}
title(expression("Estimating \(f'(x)\) : \(\frac{\text{frac}(d,y) * \sin(x) == \cos(x))}{\text{off}}\))
offs <- c(-0.1, 0, 0.1, 0.2, 0.3)
i <- 1
for(off in offs) {
d12 <- d1d2(x, y, spar.offset = off)
  lines(d12$x, d12$D1, col = i <- i+1)
}
legend(2,1.6, c("true cos()",paste("sp.off. = ", format(off))), lwd=1,
  col = 1:(1+length(offs)), cex = 0.8, bg = NA)
par(op)
```
**Description**

Compute the numerical first or 2nd derivatives of \( f() \) given observations \((x[i], y = f(x[i]))\).

\( \text{D1tr} \) is the *trivial* discrete first derivative using simple difference ratios, whereas \( \text{D1ss} \) and \( \text{D2ss} \) use cubic smoothing splines (see `smooth.spline`) to estimate first or second derivatives, respectively.

\( \text{D2ss} \) first uses `smooth.spline` for the first derivative \( f'(t) \) and then applies the same to the predicted values \( \hat{f}'(t_i) \) (where \( t_i \) are the values of \( xout \)) to find \( \hat{f}''(t_i) \).

**Usage**

\[
\text{D1tr}(y, x = 1) \\
\text{D1ss}(x, y, xout = x, spar.offset = 0.1384, spl.spar=\text{NULL}) \\
\text{D2ss}(x, y, xout = x, spar.offset = 0.1384, spl.spar=\text{NULL})
\]

**Arguments**

- \( x, y \) numeric vectors of same length, supposedly from a model \( y = f(x) \). For \( \text{D1tr}() \), \( x \) can have length one and then gets the meaning of \( h = \Delta x \).
- \( xout \) abscissa values at which to evaluate the derivatives.
- \( \text{spar.offset} \) numeric fudge added to the smoothing parameter(s), see `spl.par` below. Note that the current default is there for historical reasons only, and we often would recommend to use \( \text{spar.offset} = 0 \) instead.
- \( \text{spl.spar} \) direct smoothing parameter(s) for `smooth.spline`. If it is \( \text{NULL} \) (as per default), the smoothing parameter used will be \( \text{spar.offset} + \text{sp$spar} \), where \( \text{sp$spar} \) is the GCV estimated smoothing parameter for both smooths, see `smooth.spline`.

**Details**

It is well known that for derivative estimation, the optimal smoothing parameter is larger (more smoothing needed) than for the function itself. \( \text{spar.offset} \) is really just a *fudge* offset added to the smoothing parameters. Note that in \( R \)'s implementation of `smooth.spline`, \( \text{spar} \) is really on the \( \log \lambda \) scale.

**Value**

- \( \text{D1tr()} \) and \( \text{D1ss()} \) return a numeric vector of the length of \( y \) or \( xout \), respectively.
- \( \text{D2ss()} \) returns a list with components
  - \( x \) the abscissae values (= \( xout \)) at which the derivative(s) are evaluated.
  - \( y \) estimated values of \( \hat{f}''(x_i) \).
The function `D2ss` computes the second derivative of a smoothing spline. It takes two `spar` arguments to the two `smooth.spline` calls. The `spar.offset` is specified on input (maybe repeated to length 2).

**Author(s)**

Martin Maechler, in 1992 (for S).

**See Also**

- `dQdR` which directly uses the 2nd derivative of the smoothing spline; `smooth.spline`.

**Examples**

```r
## First Derivative --- spar.off = 0 ok "asymptotically" (?)
set.seed(330)
mult.fig(12)
for(i in 1:12) {
  x <- runif(500, 0,10); y <- sin(x) + rnorm(500)/4
  f1 <- D1ss(x=x,y=y, spar=off=0.0)
  plot(x,f1, ylim = range(c(-1,1,f1)))
  curve(cos(x), col=3, add= TRUE)
}
set.seed(8840)
x <- runif(100, 0,10)
y <- sin(x) + rnorm(100)/4
op <- par(mfrow = c(2,1))
plot(x,y)
lines(ss <- smooth.spline(x,y), col = 4)
str(ss[c("df", "spar")])
x <- seq(0,10, len=201)
plot(x,x, -sin(x), type = 'l', ylim = c(-1.5,1.5))
title(expression("Estimating f": " * frac(d^2,dx^2) * sin(x) == -sin(x))
offs <- c(0.05, 0.1, 0.1348, 0.2)
i <- 1
for(offs in offs) {
  d12 <- D2ss(x,y, spar.offset = off)
  lines(d12, col = i <- i+1)
}
legend(2,1.6, c("true : -sin(x",paste("sp.off. = ", format(offs))), lwd=1,
  col = 1:(1+length(offs)), cex = 0.8, bg = NA)
par(op)
```
**Deprecation**

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

**Usage**

```r
pmax.sa(scalar, arr)
pmin.sa(scalar, arr)
```

**Arguments**

- `scalar`: numeric scalar.
- `arr`: any numeric R object, typically array.

**Details**

`pmax.sa(s, a)` and `pmin.sa(s, a)` return (more-dimensional) arrays. These have been deprecated, because `pmax` and `pmin` do so too, if the array is used as first argument.

---

**Diagonal Discriminant Analysis**

**Description**

This function implements a simple Gaussian maximum likelihood discriminant rule, for diagonal class covariance matrices.

In machine learning lingo, this is called “Naive Bayes” (for continuous predictors). Note that naive Bayes is more general, as it models discrete predictors as multinomial, i.e., binary predictor variables as Binomial / Bernoulli.

**Usage**

```r
dDA(x, cl, pool = TRUE)
## S3 method for class 'dDA'
predict(object, newdata, pool = object$pool, ...)
## S3 method for class 'dDA'
print(x, ...)

diagDA(ls, cl, ts, pool = TRUE)
```
Arguments

x, ls
learning set data matrix, with rows corresponding to cases (e.g., mRNA samples) and columns to predictor variables (e.g., genes).

c1l
class labels for learning set, must be consecutive integers.

object
object of class dDA.

ts, newdata
test set (prediction) data matrix, with rows corresponding to cases and columns to predictor variables.

pool
logical flag. If true (by default), the covariance matrices are assumed to be constant across classes and the discriminant rule is linear in the data. Otherwise (pool= FALSE), the covariance matrices may vary across classes and the discriminant rule is quadratic in the data.

... further arguments passed to and from methods.

Value
dDA() returns an object of class dDA for which there are print and predict methods. The latter returns the same as dDA():
diagDA() returns an integer vector of class predictions for the test set.

Author(s)

Sandrine Dudoit, <sandrine@stat.berkeley.edu> and Jane Fridlyand, <janef@stat.berkeley.edu> originally wrote stat.diag.da() in CRAN package sma which was modified for speedup by Martin Maechler <maechler@R-project.org> who also introduced dDA etc.

References


See Also

lda and qda from the MASS package; naiveBayes from e1071.

Examples

## two artificial examples by Andreas Greutert:
d1 <- data.frame(x = c(1, 5, 5, 10, 25, 25, 25, 29),
                 y = c(4, 1, 2, 4, 4, 6:8, 7))
n.plot(d1)
library(cluster)
(c11P <- pam(d1,k=4)$cluster) # 4 surprising clusters
with(d1, points(x+y, col = c11P, pch =c11P))

i1 <- c(1,3,5,6)
tri <- d1[-i1,]
The "Other" Diagonal Matrix

diagX

Description
Compute the other diagonal identity matrix. The result is basically a fast version of diag(n)[, n:1].

Usage
diagX(n)

Arguments
n
positive integer.

Value
a numeric \( n \times n \) matrix with many zeros – apart from 1s in the other diagonal.

Author(s)
Martin Maechler, 1992.
digitsBase

See Also
diag.

Examples
diagX(4)
for(m in 1:5)
  stopifnot(identical(diagX(m), diag(m)[, m:1, drop = FALSE]))

---

digitsBase

Digit/Bit Representation of Integers in any Base

Description

Integer number representations in other Bases.

Formally, for every element \( N = x[i] \), compute the (vector of) “digits” \( A \) of the base \( b \) representation of the number \( N \), \( N = \sum_{k=0}^{M} A_{M-k}b^k \).

Revert such a representation to integers.

Usage
digitsBase(x, base = 2, ndigits = 1 + floor(1e-9* log(max(x),base)))
## S3 method for class 'basedInt'
as.integer(x, ...)
## S3 method for class 'basedInt'
print(x, ...)
as.intBase(x, base = 2)
b2int(xlist, base)

Arguments

x
For digitsBase(): non-negative integer (vector) whose base base digits are wanted.

For as.intBase():
a list of numeric vectors, a character vector, or an integer matrix as returned by
digitsBase(), representing digits in base base.

base
integer, at least 2 specifying the base for representation.

ndigits
number of bits/digits to use.

... potential further arguments passed to methods, notably print.

xlist a list of integer vectors with entries typically in 0:(base-1), such as resulting from digitsBase().
digitsBase

Value

For digitsBase(), an object, say m, of class "basedInt" which is basically a (ndigits x n) matrix where m[,i] corresponds to x[i], n <- length(x) and attr(m,"base") is the input base. as.intBase() and the as.integer method for basedInt objects return an integer vector. b2int() is the low-level workhorse of as.intBase().

Note

Some of these functions existed under names digits and digits.v in previous versions of the sfsmisc package.

Author(s)

Martin Maechler, Dec 4, 1991 (for S-plus; then called digits.v).

Examples

digitsBase(0:12, 8) #-- octal representation
empty.dimnames(digitsBase(0:33, 2)) # binary

## This may be handy for just one number (and default decimal):
digits <- function(n, base = 10) as.vector(digitsBase(n, base = base))
digits(128982734) # 1 2 8 9 8 2 7 3 4
digits(128, base = 8) # 2 0 0

## one way of pretty printing (base <= 10!)
b2ch <- function(db)
  noquote(gsub("\0+(\{1\})\$"," \\1", 
  apply(db, 2, paste, collapse = "")))
b2ch(digitsBase(0:33, 2)) #-> 0 1 10 11 100 101 ... 100001
b2ch(digitsBase(0:33, 4)) #-> 0 1 2 3 10 11 12 13 20 ... 200 201

## Hexadecimal:
i <- c(1:20, 100:106)
M <- digitsBase(i, 16)
hexdig <- c(0:9, LETTERS[1:6])
cM <- hexdig[1 + M]; dim(cM) <- dim(M)
b2ch(cM) #-> 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 ... 6A

## IP (Internet Protocol) numbers coding: <n>.<n>.<n>.<n> --- longinteger
ip_ntoa <- function(n)
  apply(digitsBase(n, base = 256), 2, paste, collapse=".")
ip_ntoa(2130706430 + (0:9)) # 126.255.255.254" ... 127.0.0.7"

## and the inverse:
ip_aton <- function(a)
b2int(lapply(strsplit(a, ".", fixed=TRUE), as.integer), 256)

n <- 2130706430 + (0:9)
head(ip <- ip_ntoa(n))
head(ip_aton(ip))
stopifnot(n == ip_aton(ip_ntoa(n))),

stopifnot(n ==...
Duplicated

\[ \text{ip} \equiv \text{ip\_ntoa(ip\_aton(ip)))} \]

## Inverse of digitsBase() : as.integer method for the "basedInt" class

```r
as.integer(M)
```

## or also as.intBase() working from strings:

```r
(cb <- apply(digitsBase(0:33, 4), 2, paste, collapse = ""))
```

```r
#> "000" "001" ... "200" "201"
```

```r
all(0:33 == as.intBase(cb, base = 4))
```

---

### Description

Duplicated() generalizes the duplicated method for vectors, by returning indices of “equivalence classes” for duplicated entries and returning nomatch (NA by default) for unique entries.

Note that duplicated() is not TRUE for the first time a duplicate appears, whereas Duplicated() only marks unique entries with nomatch (NA).

### Usage

```r
Duplicated(v, incomparables = FALSE, fromLast = FALSE, nomatch = NA_integer_)
```

### Arguments

- **v**
  - a vector, often character, factor, or numeric.
- **incomparables**
  - a vector of values that cannot be compared, passed to both duplicated() and match(). FALSE is a special value, meaning that all values can be compared, and may be the only value accepted for methods other than the default. It will be coerced internally to the same type as x.
- **fromLast**
  - logical indicating if duplication should be considered from the reverse side, i.e., the last (or rightmost) of identical elements would correspond to duplicated=FALSE.
- **nomatch**
  - passed to match(): the value to be returned in the case when no match is found. Note that it is coerced to integer.

### Value

an integer vector of the same length as v. Can be used as a factor, e.g., in split, tapply, etc.

### Author(s)

Christoph Buser and Martin Maechler, Seminar fuer Statistik, ETH Zurich, Sep.2007

### See Also

- uniqueL (also in sfsmisc); duplicated, match.
Examples

```r
x <- c(9:12, 1:4, 3:6, 0:7)
data.frame(x, dup = duplicated(x),
  dupL = duplicated(x, fromLast=TRUE),
  Dup = Duplicated(x),
  DupL= Duplicated(x, fromLast=TRUE))
```

---

**eaxis**

Extended / Engineering Axis for Graphics

---

**Description**

An extended `axis()` function which labels more prettily, in particular for log-scale axes.

It makes use of `plotmath` or (LaTeX) expressions of the form $k \times 10^k$ for labeling a log-scaled axis and when otherwise exponential formatting would be used (see `pretty10exp`).

**Usage**

```r
eaxis(side, at = if(log) aXTicks(side, axp=axp, log=log, nintLog=nintLog)
  else aXTicks(side, axp=axp, log=log),
  labels = NULL, log = NULL,
  f.smalltcl = 3/5, at.small = NULL, small.mult = NULL,
  small.args = list(),
  draw.between.ticks = TRUE, between.max = 4,
  outer.at = TRUE, drop.l = TRUE, sub10 = FALSE, las = 1,
  nintLog = max(10, par("lab")[2 - is.x]),
  axp = NULL, n.axp = NULL, max.at = Inf,
  lab.type = "plotmath", lab.sep = "cdot",
  ...)
```

**Arguments**

- `side`  integer in 1:4, specifying side of `axis`
- `at` numeric vector of (“normalsized”) tick locations; by default `aXTicks(side, ..)`, i.e., the same as `axis()` would use.
- `labels` NULL (default), `logical`, character or expression, as in `axis();` in addition, if NA, labels = TRUE is passed to `axis();`, i.e. `pretty10exp` is not used. Use FALSE to suppress any labeling.
- `log` logical or NULL specifying if log-scale should be used; the default depends on the current plot’s axis.
- `f.smalltcl` factor specifying the lengths of the small ticks in proportion to the normalsized, labeled ticks.
- `at.small` locations of small ticks; the default, NULL, uses `small.mult` and constructs “smart” locations.
small.mult  positive integer (or NULL), used when at\_small is NULL to indicate which multiples of at (typically \texttt{axTicks()}) should be used as “small ticks”. The default NULL will use 9 in the log case and a number in 2:5 otherwise.

small.args  optional \texttt{list} of further arguments to the \texttt{axis()} call which draws the small ticks.

draw.between.ticks  
(only if log is true): logical indicating that possible (non-small) ticks between the labeled (via at) ones should be drawn as well (and possibly also used for at\_small construction), see also between.max.

between.max  (only if log and draw\_between\_ticks are true): integer indicating ticks should be drawn (approximately) between the labeled ones.

outer.at  logical specifying that at\_small should also be constructed outside the at range, but still inside the corresponding \texttt{par(“usr”).}

drop.1  logical specifying if $1 \times$ should be dropped from labels, passed to \texttt{pretty10exp()}.  

sub10  logical, integer (of length 1 or 2) or "10", indicating if some $10^k$ should be simplified to "traditional" formats, see \texttt{pretty10exp}.

nintLog  only used in R > 2.13.x, when log is true: approximate (lower bound on) number of intervals for log scaling.

axp  to be passed to \texttt{axTicks()} if at is not specified.

n.axp  to be set to axp[3] when axp and at are not specified, in order to tweak the number of (non-small) tick marks produced from \texttt{axTicks()}, notably when log is true, set n.axp to 1, 2, or 3:

  1:  will produce tick marks at $10^j$ for integer $j$,
  2:  gives marks $k10^j$ with $k \in \{1, 5\}$,
  3:  gives marks $k10^j$ with $k \in \{1, 2, 5\}$

see 'xaxp' on the \texttt{par} help page.

max.at  maximal number of at values to be used effectively. If you don’t specify at yourself carefully, it is recommended to set this to something like 25, but this is not the default, for back compatibility reasons.

las, ...  arguments passed to (the first) \texttt{axis} call. Note that the default las = 1 differs from axis’s default las = 0.

lab.type  string, passed to \texttt{pretty10exp} to choose between default \texttt{plotmath} or LaTeX label format.

lab.sep  separator between mantissa and exponent for LaTeX labels, see \texttt{pretty10exp}.

\textbf{Author(s)}

Martin Maechler

\textbf{See Also}

\texttt{axis, axTicks, axExpr, pretty10exp}. 

Examples

```r
x <- lseq(1e-10, 0.1, length = 201)
plot(x, pt(x, df=3), type = "l", xaxt = "n", log = "x")
eaxis()
## without small ticks:
eaxis(3, at.small=FALSE, col="blue")

## If you like the ticks, but prefer traditional (non-"plotmath") labels:
plot(x, gamma(x), type = "l", log = "x")
eaxis(1, labels=NA)

x <- lseq(.001, .1, length = 1000)
plot(x, sin(1/x)*x, type = "l", xaxt = "n", log = "x")
eaxis()
eaxis(3, n.axp = 1)# -> xaxp[3] = 1: only 10^j (main) ticks

## non-log-scale: draw small ticks, but no "10^k" if not needed:
x <- seq(-100, 100, length = 1000)
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis()  # default -> (1, 2, 5) * 10^j ticks
eaxis(3, n.axp = 2)# -> xaxp[3] := 2 -- approximately two (main) ticks

x <- x/1000
plot(x, 1-sin(x)/x, type = "l", xaxt = "n", yaxt = "n")
eaxis()
eaxis(2)

## more labels than default:
op <- par(lab=c(10,5,7))
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis()  # maybe (depending on your canvas), there are too many,
## in that case, maybe use
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis(1, axticks=TRUE)[c(TRUE,FALSE)]  # drop every 2nd label
eaxis(3, labels=FALSE)

## ore use 'max.at' which thins as well:
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis(1, max.at=6)
par(op)

## Answering R-help "How do I show real values on a log10 histogram", 26 Mar 2013
## the data:
set.seed(1); summary(x <- rlnorm(100, m = 2, sdl = 3))
## the plot (w/o x-axis):
r <- hist(log10(x), xaxt = "n", xlab = "x [log scale]"
## the nice axis:
axt <- axticks(1)
eaxis(1, at = axt, labels = pretty10exp(axt, drop.1=TRUE))
```
ecdf.ksCI

Plot Empirical Distribution Function With 95% Confidence Curves

Description

Plots the empirical (cumulative) distribution function (ECDF) for univariate data, together with upper and lower simultaneous 95% confidence curves, computed via Kolmogorov-Smirnov' D, see KSD.

Usage

ecdf.ksCI(x, main = NULL, sub = NULL, xlab = deparse(substitute(x)),
          ci.col = "red", ...)

Arguments

x x numerical vector of observations.
main, sub, xlab arguments passed to title.
    ci.col color for confidence interval lines.
    ... optional arguments passed to plot.stepfun.

Value

Nothing. Used for its side effect, to produce a plot.

Note

Presently, will only work if length(x) > 9.

Author(s)

Kjetil Halvorsen

References

Bickel and Doksum, see KSD.
ellipsePoints

Compute Radially Equispaced Points on Ellipse

Description

Compute points on (the boundary of) an ellipse which is given by elementary geometric parameters.

Usage

ellipsePoints(a, b, alpha = 0, loc = c(0, 0), n = 201, keep.ab.order=FALSE)

Arguments

a, b  
length of half axes in (x,y) direction. Note that (a, b) is equivalent to (b, a) unless keep.ab.order=TRUE.

alpha  
angle (in degrees) giving the orientation of the ellipse, i.e., the original (x,y)-axis ellipse is rotated by angle.

loc  
center (LOCation) of the ellipse.

n  
count number of points to generate.

keep.ab.order  
logical indicating if (a, b) should be considered ordered. When FALSE, as per default, the orientation of the ellipse is solely determined by alpha. Note that keep.ab.order = TRUE seems a more natural default, but FALSE is there for back-compatibility.

Value

A numeric matrix of dimension n x 2, each row containing the (x,y) coordinates of a point.

Author(s)

Martin Maechler, March 2002.

See Also

the ‘ellipse’ package and ellipsoidhull and ellipsoidPoints in the ‘cluster’ package.
Examples

## Simple Ellipse, centered at (0,0), x-/y- axis parallel:

```r
ep <- ellipsePoints(5,2)
str(ep)
plot(ep, type="n",asp=1); polygon(ep, col = 2)
## (a,b) = (2,5) is equivalent to (5,2):
lines(ellipsePoints(2,5), lwd=2, lty=3)
## keep.order=TRUE: Now, (2,5) are axes in x- respective y- direction:
lines(ellipsePoints(2,5, keep.ab.order=TRUE), col="blue")
```

## rotate by 30 degrees:

```r
plot(ellipsePoints(5,2, alpha = 30), asp=1)
abline(h=0,v=0,col="gray")
abline(a=0,b= tan(30 * pi/180), col=2, lty = 2)
abline(a=0,b= tan(120 * pi/180), col=3, lty = 2)
```

## NB: use x11(type = "Xlib") for the following if you can

```r
if(dev.interactive(TRUE)) {
## Movie: rotating ellipse:
nTurns <- 4 # (#full 360 deg turns)
for(al in 1:(nTurns*360)) {
  ep <- ellipsePoints(3,6, alpha=al, loc = c(5,2))
  plot(ep,type="l",xlim=c(-1,11),ylim=c(-4,8),
       asp=1, axes = FALSE, xlab="", ylab="")
}
## Movie: rotating _filled_ ellipse {less nice to look at}
for(al in 1:180) {
  ep <- ellipsePoints(3,6, alpha=al, loc = c(5,2))
  plot(ep,type="n",xlim=c(-1,11),ylim=c(-4,8),
       asp=1, axes = FALSE, xlab="", ylab="")
  polygon(ep,col=2,border=3,lwd=2.5)
}
} # only if interactive
```

---

**empty.dimnames**

### Empty Dimnames of an Array

**Description**

Remove all dimension names from an array for compact printing.

**Usage**

```r
empty.dimnames(a)
```

**Arguments**

- `a` an **array**, i.e., as special case a **matrix**.
Value

Returns a with its dimnames replaced by empty character strings.

Author(s)

Bill Venables / Martin Maechler, Sept 1993.

See Also

unname removes the dimnames.

Examples

eemptyNdimnames(diag(5)) # looks much nicer

(a <- matrix(-9:10, 4, 5))
emptyNdimnames(a) # nicer, right?

Description

Draws a scatter plot, adding vertical “error bars” to all the points.

Usage

```r
errbar(x, y, yplus, yminus, cap = 0.015,
      ylim = range(y, yplus, yminus),
      xlab = deparse(substitute(x)),
      ylab = deparse(substitute(y)), ...)```

Arguments

- `x` vector of x values.
- `y` vector of y values.
- `yplus` vector of y values: the tops of the error bars.
- `yminus` vector of y values: the bottoms of the error bars.
- `cap` the width of the little lines at the tops and bottoms of the error bars in units of the width of the plot. Default is 0.015.
- `ylim` (numeric of length 2): the y-axis extents with a sensible default.
- `xlab, ylab` axis labels for the plot, as in `plot.default`.
- `...` Graphical parameters (see `par`) may also be supplied as arguments to this function.
**f.robftest**

**Robust F-Test: Wald test for multiple coefficients of rlm() Object.**

**Description**

Compute a robust F-Test, i.e., a Wald test for multiple coefficients of an `rlm` object.

**Usage**

```r
f.robftest(object, var = -1)
```

**Arguments**

- `object`: result of `rlm()`.
- `var`: variables. Either their names or their indices; the default, -1 means all but the intercept.

**Details**

This builds heavily on `summary.rlm()`, the summary method for `rlm` results.

**Value**

An object of class "htest", hence with the standard print methods for hypothesis tests. This is basically a list with components

- `statistic`: the F statistic, according to ...
- `df`: numerator and denominator degrees of freedom.
- `data.name`: (extracted from input object.)
- `alternative`: "two.sided", always.
- `p.value`: the P-value, using an F-test on statistic and df[1:2].

**Author(s)**

Werner Stahel, Jul.2000; updates by Martin Maechler.

---

**Author(s)**


**See Also**

- `errbar` in package `Hmisc` is similar.

**Examples**

```r
y <- rnorm(10); d <- 1 + .1*rnorm(10)
errbar(1:10, y, y + d, y - d, main="Error Bars example")
```
factorize

Prime Factorization of Integers

Description

Compute the prime factorization(s) of integer(s) n.

Usage

factorize(n, verbose = FALSE)

Arguments

n vector of integers to factorize.
verbose logical indicating if some progress information should be printed.

Details

works via primes, currently in a cheap way, sub-optimal for large composite n.

Value

A named list of the same length as n, each element a 2-column matrix with column "p" the prime factors and column~"m" their respective exponents (or multiplities), i.e., for a prime number n, the resulting matrix is cbind(p = n, m = 1).

Author(s)

Martin Maechler, Jan. 1996.
### hatMat

**Hat Matrix of a Smoother**

**Description**

Compute the hat matrix or smoother matrix, of ‘any’ (linear) smoother, smoothing splines, by default.

**Usage**

```r
hatMat(x, trace = FALSE, 
   pred.sm = function(x, y, ...) 
   predict(smooth.spline(x, y, ...), x = x)$y, 
   ...)```

**Arguments**

- **x** numeric vector or matrix.
- **trace** logical indicating if the whole hat matrix, or only its trace, i.e. the sum of the diagonal values should be computed.
- **pred.sm** a function of at least two arguments (`x, y`) which returns fitted values, i.e. $\hat{y}$, of length compatible to `x` (and `y`).
- ... optionally further arguments to the smoother function `pred.sm`.

**Value**

The hat matrix $H$ (if `trace = FALSE` as per default) or a number, $tr(H)$, the trace of $H$, i.e., $\sum_i H_{ii}$.

Note that `dim(H) == c(n, n)` where `n <- length(x)` also in the case where some `x` values are duplicated (aka ties).

**Author(s)**

Martin Maechler <maechler@stat.math.ethz.ch>

**References**

See Also

smooth.spline, etc. Note the demo, demo("hatmat-ex").

Examples

```r
require(stats) # for smooth.spline() or loess()

x1 <- c(1:4, 7:12)
H1 <- hatMat(x1, spar = 0.5) # default: smooth.spline()
matplot(x1, H1, type = "l", main = "columns of smoother hat matrix")

## Example 'pred.sm' arguments for hatMat():
pspl <- function(x,y,...) predict(smooth.spline(x,y,...), x = x)$y
pksm <- function(x,y,...) ksmooth(sort(x),y, "normal", x.points=x,...)$y
## Rather than ksmooth():
if(require("lokern"))
  pksm2 <- function(x,y,...) glkerns(x,y, x.out=x,...)$est

## Explaining 'trace = TRUE'
all.equal(sum(diag(hatMat(c(1:4, 7:12), df = 4)))),
  hatMat(c(1:4, 7:12), df = 4, trace = TRUE), tol = 1e-12)

## ksmooth():
Hk <- hatMat(x1, pr = pksm, bandwidth = 2)
cat(sprintf("df = %.2f
", sum(diag(Hk))))
image(Hk)
Matrix::printSpMatrix(as(round(Hk, 2), "sparseMatrix"))
##--- see demo("hatmat-ex") for more (and larger) examples
```

---

**histBxp**

*Plot a Histogram and a Boxplot*

**Description**

Creates a histogram and a horizontal boxplot on the current graphics device.

**Usage**

```r
histBxp(x, nclass, breaks, probability=FALSE, include.lowest=TRUE,
  xlab = deparse(substitute(x)),
  ...,
  width=0.2, boxcol=3, medcol=2, medlwd=5, whisklty=2, staplelty=1)
```
Arguments

- **x**: numeric vector of data for histogram. Missing values (NAs) are allowed.
- **nclass**: recommendation for the number of classes (i.e., bars) the histogram should have. The default is a number proportional to the logarithm of the length of x.
- **breaks**: vector of the break points for the bars of the histogram. The count in the i-th bar is \( \sum(\text{breaks}[i] < x \leq \text{breaks}[i+1]) \) except that if include.lowest is TRUE (the default), the first bar also includes points equal to breaks[1]. If omitted, evenly-spaced break points are determined from nclass and the extremes of the data.
- **probability**: logical flag: if TRUE, the histogram will be scaled as a probability density; the sum of the bar heights times bar widths will equal 1. If FALSE, the heights of the bars will be counts.
- **include.lowest**: If TRUE (the default), the lowest bar will include data points equal to the lowest break, otherwise it will act like the other bars (see the description of the breaks argument).
- **xlab**: character or expression for x axis labeling.
- **...**: additional arguments to barplot. The hist function uses the function barplot to do the actual plotting; consequently, arguments to the barplot function that control shading, etc., can also be given to hist. See the barplot documentation for arguments angle, density, col, and inside. Do not use the space or histo arguments.
- **width**: width of the box relative to the height of the histogram. DEFAULT is 0.2.
- **boxcol**: color of filled box. The default is 3.
- **medcol**: the color of the median line. The special value, NA, indicates the current plotting color (par("col")). The default is 2. If boxcol=0 and medcol is not explicitly specified this is set to the current plotting color (par("col")).
- **medlwd**: median line width. The special value NA, is used to indicate the current line width (par("lwd")). The default is 5.
- **whisklty**: whisker line type. The special value NA indicates the current line type (par("lty")). The default is 2 (dotted line).
- **staplelty**: staple (whisker end cap) line type. The special value NA indicates the current line type (par("lty")). The default is 1 (solid line).

Graphical parameters (see par) may also be supplied as arguments to this function. In addition, the high-level graphics arguments described under par and the arguments to title may be supplied to this function.

Details

If include.lowest is FALSE the bottom breakpoint must be strictly less than the minimum of the data, otherwise (the default) it must be less than or equal to the minimum of the data. The top breakpoint must be greater than or equal to the maximum of the data.

This function has been called hist.bxp() for 17 years; in 2012, the increasingly strong CRAN policies required a new name (which could not be confused with an S3 method name).
integrate.xy

Cheap Numerical Integration through Data points.

Description

Given \((x_i, f_i)\) where \(f_i = f(x_i)\), compute a cheap approximation of \(\int_a^b f(x) \, dx\).

Usage

integrate.xy(x, fx, a, b, use.spline=TRUE, xtol=2e-08)

Arguments

- **x**: abscissa values.
- **fx**: corresponding values of \(f(x)\).
- **a, b**: the boundaries of integration; these default to min(x) and max(x) respectively.
- **use.spline**: logical; if TRUE use an interpolating spline.
- **xtol**: tolerance factor, typically around \(\sqrt{\text{Machine\$double\$eps}}\) ......(fixme)....

Details

Note that this is really not good for noisy \(fx\) values; probably a smoothing spline should be used in that case.

Also, we are not yet using Romberg in order to improve the trapezoid rule. This would be quite an improvement in equidistant cases.

Value

the approximate integral.
Inverse seq() – Short Expression for Index Vector

Description

Compute a short expression for a given integer vector, typically an index, that can be expressed shortly, using : etc.

Usage

inv.seq(i)

Arguments

i vector of (usually increasing) integers.

Value

a call (“the inside of an expression”) to be eval()ed to return the original i.

Author(s)

Martin Maechler, October 1995; more elegant implementation from Tony Plate.

See Also

rle for another kind of integer vector coding.
Examples

```
(rr <- inv.seq(i1 <- c(3:12, 20:24, 27, 30:33)))
eval(rr)
stopifnot(eval(rr) == i1)

e2 <- expression(c(20:13, 3:12, -1:-4, 27, 30:31))
(i2 <- eval(e2))
(r2 <- inv.seq(i2))
stopifnot(all.equal(r2, e2[[1]]))

## Had (mapply()) bug in this example:
ii <- c(1:3, 6:9, 11:16)
stopifnot(identical(ii, eval(inv.seq(ii))))
```

Description

This function tests whether a numeric or complex vector or array consists of whole numbers. The function `is.integer` is not appropriate for this since it tests whether the vector is of class `integer` (see examples).

Usage

```
is.whole(x, tolerance = sqrt(.Machine$double.eps))
```

Arguments

- `x`: integer, numeric, or complex vector or array to be tested
- `tolerance`: maximal distance to the next whole number

Value

The return value has the same dimension as the argument `x`: if `x` is a vector, the function returns a logical vector of the same length; if `x` is a matrix or array, the function returns a logical matrix or array of the same dimensions. Each entry in the result indicates whether the corresponding entry in `x` is whole.

Author(s)

Alain Hauser <alain@huschhus.ch>

See Also

- `is.integer`
Examples

```r
## Create a random array, matrix, vector
set.seed(307)
a <- array(runif(24), dim = c(2, 3, 4))
a[4:8] <- 4:8
m <- matrix(runif(12), 3, 4)
m[2:4] <- 2:4
v <- complex(real = seq(0.5, 1.5, by = 0.1),
              imaginary = seq(2.5, 3.5, by = 0.1))

## Find whole entries
is.whole(a)
is.whole(m)
is.whole(v)

## Numbers of class integer are always whole
is.whole(dim(a))
is.whole(length(v))
```

Description

Generate numeric sequences applying a linear recursion `nr.it` times.

Usage

```r
iterate.lin.recursion(x, coeff, delta = 0, nr.it)
```

Arguments

- `x`: numeric vector with initial values, i.e., specifying the beginning of the resulting sequence; must be of length (larger or) equal to `length(coeff)`.
- `coeff`: coefficient vector of the linear recursion.
- `delta`: numeric scalar added to each term; defaults to 0. If not zero, determines the linear drift component.
- `nr.it`: integer, number of iterations.

Value

numeric vector, say `r`, of length `n + nr.it`, where `n = length(x)`. Initialized as `r[1:n] = x`, the recursion is `r[k+1] = sum(coeff * r[(k-m+1):k])`, where `m = length(coeff)`.

Note

Depending on the zeroes of the characteristic polynomial of `coeff`, there are three cases, of convergence, oscillation and divergence.
Author(s)

Martin Maechler

See Also

`seq` can be regarded as a trivial special case.

Examples

```r
## The Fibonacci sequence:
iterate.lin.recursion(0:1, c(1,1), nr = 12)
## 0 1 1 2 3 5 8 13 21 34 55 89 144 233

## seq() as a special case:
stopifnot(iterate.lin.recursion(4,1, d=2, nr=20)
  == seq(4, by=2, length=1+20))

## 'Deterministic AR(2)' :
round(iterate.lin.recursion(1:4, c(-0.7, 0.9), d = 2, nr=15), dig=3)
## slowly decaying :
plot(ts(iterate.lin.recursion(1:4, c(-0.9, 0.95), nr=150)))
```

Description

Computes the critical value for Kolmogorov-Smirnov’s $D_n$, for sample sizes $n \geq 10$ and confidence level 95%.

Usage

`KSD(n)`

Arguments

- `n`: the sample size, $n \geq 10$.

Details

Based on tables values given in the reference below. For $n \leq 80$ uses interpolations from exact values, elsewhere uses asymptotic approximation.

Value

The critical value for $D$ (two-sided) for significance level 0.05 (or confidence level 95%).
last

Get Last Elements of a Vector

Description

Extract the last elements of a vector.

Usage

last(x, length.out = 1, na.rm = FALSE)

Arguments

x

any vector.

length.out

integer indicating how many element are desired. If positive, return the length.out last elements of x; if negative, the last length.out elements are dropped.

na.rm

logical indicating if the last non-missing value (if any) shall be returned. By default (it is FALSE and) the last elements (whatever its values) are returned.

Value

a vector of length abs(length.out) of last values from x.
Note

This function may eventually be deprecated for the standard R function `tail()`. Useful for the `turnogram()` function in package `pastecs`.

Author(s)

Werner Stahel (<stahel@stat.math.ethz.ch>), and independently, Philippe Grosjean (<phgrosjean@sciviews.org>), Frédéric Ibanez (<ibanez@obs-vlfr.fr>).

See Also

`first`, `turnogram`

Examples

```r
a <- c(NA, 1, 2, NA, 3, 4, NA)
last(a)
last(a, na.rm=TRUE)

last(a, length = 2)
last(a, length = -3)
```

---

**linesHyperb.lm**

*Plot Confidence or Prediction Hyperbolas around a Regression Line*

Description

Add confidence/prediction hyperbolas for \( y(x_0) \) to a plot with data or regression line.

Usage

```r
linesHyperb.lm(object, c.prob=0.95, confidence=FALSE, k=if (confidence) Inf else 1, col=2, lty=2, do.abline=TRUE)
```

Arguments

- `object`: result of `lm(.)`.
- `c.prob`: coverage probability in \((0, 1)\).
- `confidence`: logical; if true, do (small) confidence band, else, realistic prediction band for the mean of \( k \) observations.
- `k`: integer or Inf; assume \( k \) future observations; \( k = \text{Inf} \) corresponds to confidence intervals (for \( y \)).
- `col`, `lty`: attributes for the `lines` to be drawn.
- `do.abline`: logical; if true, the regression line is drawn as well.
Note
With `predict.lm(*, interval=)` is available, this function `linesHyerb.lm` is only slightly more general for its `k` argument.

Author(s)
Martin Maechler, Oct 1995

See Also
predict.lm(*, interval=) optionally computes prediction or confidence intervals.

Examples
```r
data(swhiss)
  plot(Fertility ~ Education, data = swiss) # the data
  (lmS <- lm(Fertility ~ Education, data = swiss))
  linesHyperb(lmS)
  linesHyperb.lm(lmS, conf=TRUE, col="blue")
```

Description
A graphical and interactive demonstration and visualization of how `loess` works. By clicking on the graphic, the user determines the current estimation window which is visualized together with the weights.

Usage
```r
loessDemo(x, y, span = 1/2, degree = 1, family = c("gaussian", "symmetric"),
  nearest = FALSE, nout = 501,
  xlim = numeric(0), ylim = numeric(0), strictlim = TRUE, verbose = TRUE,
  inch.sym = 0.25, pch = 4, shade = TRUE, w.symbols = TRUE,
  sym.col = "blue", w.col = "light blue", line.col = "steelblue")
```

Arguments
- `x, y` numeric vectors of the same length; the demo is about `loess(y ~ x)`.
- `span` the smoothing parameter $\alpha$.
- `degree` the degree of the polynomials to be used; must be in 0, 1, 2.
- `family` if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey’s biweight function. Can be abbreviated.
- `nearest` logical indicating how $x_0$ should be determined, the value at which $\hat{f}(x_0)$ is computed. If nearest is true, the closest data value is taken.
loessDemo

nout
the number of points at which to evaluate, i.e., determining \( u_i, i = 1, 2, \ldots, \text{nout} \), at which \( \hat{f}(u_i) \) is computed.

xlim
x-range; to extend or determine (iff strictlim is true) the \( x \)-range for plotting.

ylim
y-range; to extend or determine (iff strictlim is true) the \( y \)-range for plotting.

strictlim
logical determining if xlim and ylim should be strict limits (as e.g., in plot.default), or just a suggestion to extend the data-dependent ranges.

verbose
logical ......

inch.sym
symbol size in inches of the maximal weight circle symbol.

pch
plotting character, see points.

shade
logical; if true, polygon(..., density=..) will be used to shade off the regions where the weights are zero.

w.symbols
logical indicating if the non-zero weights should be visualized by circles with radius proportional to inch.sym and \( \sqrt{w} \) where \( w \) are the weights.

sym.col, w.col, line.col
colors for the symbols, weights and lines, respectively.

Author(s)

As function loess.demo(), written and posted to S-news, on 27 Sep 2001, by Greg Snow, Brigham Young University, it was modified by Henrik Aa. Nielsen, IMM, DTU, and subsequently spiffed up for R by Martin Maechler.

See Also

loess.

Examples

if(dev.interactive()) {

  if(requireNamespace("lattice")) {
    data("ethanol", package = "lattice")
    attach(ethanol)
    loessDemo(E,N0x, span=.25)
    loessDemo(E,N0x, span=.25, family = "symmetric")
    loessDemo(E,N0x, degree=0)# Tricube Kernel estimate
  }

  ## Artificial Example with one outlier
  n2 <- 50; x <- 1:(1+2*n2)
  fx <- (x/10 - 5)^2
  y <- fx + 4*rnorm(x)
  y[n2+1] <- 1e4
  loessDemo(x,y, span=1/3, ylim= c(0,1000))# not robust !
  loessDemo(x,y, span=1/3, family = "symm")
  loessDemo(x,y, span=1/3, family = "symm", w.symb = FALSE, ylim = c(0,40))
  loessDemo(x,y, span=1/3, family = "symm", ylim = c(0,40))
}
## lseq

### Generate Sequences, Equidistant on Log Scale

#### Description
Generate sequences which are equidistant on a log-scale.

#### Usage
```
lseq(from, to, length)
```

#### Arguments
- `from`: starting value of sequence.
- `to`: end value of the sequence.
- `length`: desired length of the sequence.

#### Value
A numeric vector of length `length`.

#### See Also
- `seq`

#### Examples
```
x <- lseq(1, 990, length = 21)
plot(x, x^4, type = "b", col = 2, log = "xy")
if(with(R.version, major >= 2 && minor >= 1))
  plot(x, exp(x), type = "b", col = 2, log = "xy")
```
mat2tex

Produce LaTeX commands to print a matrix

Description

“Translate” an R matrix (like object) into a LaTeX table, using \begin{tabular} ....

Usage

mat2tex(x, file = "mat.tex", envir = "tabular",
         nam.center = "1", col.center = "c",
         append = TRUE, digits = 3, title)

Arguments

x a matrix
file names the file to which LaTeX commands should be written
envir a string, the LaTeX environment name; default is "tabular"; useful maybe "array", or other versions of tabular environments.
nam.center character specifying row names should be center; default "l".
col.center character (vector) specifying how the columns should be centered; must have values from c("l", "c", "r"); defaults to "c".
append logical; if FALSE, will destroy the file file before writing commands to it; otherwise (by default), simply adds commands at the end of file file.
digits integer; setting of options(digits=...) for purpose of number representation.
title a string, possibly using LaTeX commands, which will span the columns of the LaTeX matrix

Value

No value is returned. This function, when used correctly, only writes LaTeX commands to a file.

Author(s)

For S: Vincent Carey <vjc@sph.unix.sph.jhu.edu>, from a post on Feb.19, 1991 to S-news. Port to R (and a bit more) by Martin Maechler <maechler@stat.math.ethz.ch>.

See Also

latex in package Hmisc is more flexible (but may surprise by its auto-printing ..).
Examples

```r
mex <- matrix(c(pi, pi/2, pi/4, exp(1), exp(2), exp(3)), nrow=2, byrow=TRUE,
              dimnames = list(c("$\pi$", "$e$"), c("a", "b", "c")))
mat2tex( mex, title="$\pi$, e, etc." )

## The last command produces the file "mat.tex" containing
## \begin{tabular} {| l|| c| c| c|}
## \hline
## multicolor{ 4 }{c}{ $\pi$, e, etc. } \ \hline
## \hline
## \ & a & b & c \ \hline
## \hline $\pi$ & 3.14 & 1.57 & 0.785 \ \hline
## \hline $e$ & 2.72 & 7.39 & 20.1 \ \hline
## \end{tabular}

## Now you have to properly embed the contents of this file
## in a LaTeX document -- for example, you will need a
## preamble, the \begin{document} statement, etc.

## Note that the backslash needs protection in dimnames
## or title actions.
mat2tex(mex, stdout(), col.center = c("r", "r", "c"))
```

missingCh

Has a Formal Argument been Set or is it Missing?

Description

missingCh can be used to test whether a value was specified as an argument to a function. Very much related to the standard R function `missing`, here the argument is given by its name, a character string.

As missingCh() calls missing(), do consider the caveats about the latter, see `missing`.

Usage

```r
missingCh(x, envir = parent.frame())
```

Arguments

- `x` a character string.
- `envir` a (function evaluation) environment, in which the variable named `x` is to be “missing”.

Value

a logical indicating if the argument named `x` is `missing` in the function “above”, typically the caller of missingCh, but see the use of `envir` in the vapply example.
**mpl**

**Simple Matrix Plots**

Do simple matrix plots, providing an easy interface to `matplot` by using a default x variable.

**Usage**

```r
mpl(mat, ...)  
p.m(mat, ...)  
```

**Arguments**

- `mat` numeric matrix.
- `...` further arguments passed to `matplot`, e.g., `type`, `xlab`, etc.
Details

\(p \cdot m(n)\) use the first column of \(m\) as \(x\) variable, whereas \(mpl(m)\) uses the integers 1, 2, \ldots, \(\text{ncol}(m)\) as coordinates and \(\text{rownames}(m)\) as axis labels if possible.

Note

These were really created for playing around with curves etc, and probably should be deprecated since in concrete examples, using \texttt{matplot()} directly is more appropriate.

Author(s)

Martin Maechler

See Also

\texttt{matplot}, \texttt{plot}, \texttt{plot.mts}, \texttt{plot.types}.

Examples

\begin{verbatim}
data(animals, package = "cluster") mpl(animals, type = "l")
\end{verbatim}

Description

Easy Setup for plotting multiple figures (in a rectangular layout) on one page. It allows to specify a main title and uses smart defaults for several \texttt{par} calls.

Usage

\begin{verbatim}
mult.fig(nr.plots, mfrow, mfcoll, marP = rep(0, 4), mgp = c(if(par("las") != 0) 2. else 1.5, 0.6, 0), mar = marP + 0.1 + c(4,4,2,1), oma = c(0,0, tit.wid, 0), main = NULL, tit.wid = if (is.null(main)) 0 else 1 + 1.5*cex.main, cex.main = par("cex.main"), line.main = cex.main - 1/2, col.main = par("col.main"), font.main = par("font.main"), \ldots)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{nr.plots} \hspace{1cm} integer; the number of plot figures you'll want to draw.
  \item \texttt{mfrow, mfcoll} \hspace{1cm} instead of \texttt{nr.plots}; integer(2) vectors giving the rectangular figure layout for \texttt{par(mfrow = *)}, or \texttt{par(mfcol=*)}, respectively. The default is to use \texttt{mfrow = n2mfrow(nr.plots)}.
  \item \texttt{marP} \hspace{1cm} numeric(4) vector of figure margins to \texttt{add} ("Plus") to default \texttt{mar}, see below.
\end{itemize}
mgp 

argument for par(mgp= .) with a smaller default than usual.

mar 

argument for par(mar= .) with a smaller default than usual, using the marP argument, see above.

oma 

argument for par(oma= .), by default for adding space for the main title if necessary.

main 

title character. The main title to be used for the whole graphic.

tit.wid 

numeric specifying the vertical width to be used for the main title; note that this is only used for the default value of oma (s. above).

cex.main 

numeric; the character size to be used for the main title.

line.main 

numeric; the margin line at which the title is written (via mtext(main, side=3, outer=TRUE, line = 1:3)).

col.main, font.main 

color and font for main title, passed to mtext(), see also par(*).

... 

further arguments to mtext() for the main title.

Value

A list with two components that are lists themselves, a subset of par(),

new.par 

the current par settings.

old.par 

the par before the call.

Author(s)

Martin Maechler, UW Seattle, 1990 (for S).

See Also

par, layout.

Examples

opl <- mult.fig(5, main= expression("Sine Functions " * sin(n * pi * x)))
x <- seq(0, 1, len = 201)
for (n in 1:5)
  plot(x, sin(n * pi * x), ylab ="", main = paste("n = ",n))
par(opl$old.par)

rr <- mult.fig(mfrow=c(5,1), main= "Cosinus Funktionen", cex = 1.5,
  marP = - c(0, 1, 2, 0))
for (n in 1:5)
  plot(x, cos(n * pi * x), type = 'l', col="red", ylab ="")
str(rr)
par(rr$old.par)
## The *restored* par settings:
str(do.call("par", as.list(names(rr$new.par))))
n.code

Convert "Round" Integers to Short Strings and Back

Description

n.code convert “round integers” to short character strings. This is useful to build up variable names in simulations, e.g.

code2n is the inverse function of n.code().

Usage

n.code(n, ndig = 1, dec.codes = c(“”, “d”, “c”, “k”))
code2n(ncod, ndig = 1, dec.codes = c(“”, “d”, “c”, “k”))

Arguments

n integer vector.
ncod character vector, typically resulting from n.code.
ndig integer giving number of digits before the coding character.
dec.codes character code for 1, 10, 100, 1000 (etc).

Value

n.code(n) returns a character vector of the same length as n.
code2n(ncod) returns a integer vector of the same length as ncod.

Usually, code2n(n.code(n)) == n.

Author(s)

Martin Maechler

Examples

n10 <- c(10, 20, 90, 100, 500, 2000, 10000)
(c10 <- n.code(n10))#-> “1d” “2d” “9d” “1c” ..
stopifnot(code2n(c10) == n10)
n.plot

Name Plot: Names or Numbers instead of Points in Plot

Description

A utility function which basically calls \texttt{plot(\textbullet, type="n")} and \texttt{text}. To have names or numbers instead of points in a plot is useful for identification, e.g., in a residual plot, see also \texttt{TA.plot}.

Usage

\begin{verbatim}
n.plot(x, y = NULL, nam = NULL, abbr = n >= 20 || max(nchar(nam))>=8, xlab = NULL, ylab = NULL, log = "", cex = par("cex"), col = par("col"), ...)\end{verbatim}

Arguments

- **x, y** coordinates at which to plot. If \texttt{y} is missing, \texttt{x} is used for both, if it’s a \texttt{data.frame, list}, 2-column matrix etc – via \texttt{xy.coords}; formula do not work.
- **nam** the labels to plot at each (x,y). Per default, these taken from the data \texttt{x} and \texttt{y}; case numbers 1:n are taken if no names are available.
- **abbr** logical indicating if the \texttt{nam} labels should be abbreviated – with a sensible default.
- **xlab, ylab** labels for the x- and y-axis, the latter being empty by default.
- **log** character specifying if log scaled axes should be used, see \texttt{plot.default}.
- **cex** plotting character expansion, see \texttt{par}.
- **col** color to use for \texttt{text}().
- **...** further arguments to be passed to the \texttt{plot} call.

Value

invisibly, a character vector with the labels used.

Author(s)

Martin Maechler, since 1992

See Also

\texttt{plot.default, text}.

Examples

\begin{verbatim}
n.plot(1:20, cumsum(rnorm(20)))
data(cars)
with(cars, n.plot(speed, dist, cex = 0.8, col = "forest green"))\end{verbatim}
nearcor

Find the Nearest Proper Correlation Matrix

Description

This function “smoothes” an improper correlation matrix as it can result from `cor` with use="pairwise.complete.obs" or `hetcor`.

It is deprecated now, in favor of `nearPD()` from package `Matrix`.

Usage

```r
nearcor(R, eig.tol = 1e-06, conv.tol = 1e-07, posd.tol = 1e-08,
        maxits = 100, verbose = FALSE)
```

Arguments

- `R`: a square symmetric approximate correlation matrix
- `eig.tol`: defines relative positiveness of eigenvalues compared to largest, default=1.0e-6.
- `conv.tol`: convergence tolerance for algorithm, default=1.0e-7
- `posd.tol`: tolerance for enforcing positive definiteness, default=1.0e-8
- `maxits`: maximum number of iterations
- `verbose`: logical specifying if convergence monitoring should be verbose.

Details

This implements the algorithm of Higham (2002), then forces symmetry, then forces positive definiteness using code from `posdefify`. This implementation does not make use of direct LAPACK access for tuning purposes as in the MATLAB code of Lucas (2001). The algorithm of Knol DL and ten Berge (1989) (not implemented here) is more general in (1) that it allows contraints to fix some rows (and columns) of the matrix and (2) to force the smallest eigenvalue to have a certain value.

Value

A `list`, with components

- `cor`: resulting correlation matrix
- `fnorm`: Froebenius norm of difference of input and output
- `iterations`: number of iterations used
- `converged`: logical

Author(s)

Jens Oehlschlägel
References

See those in posdefify.

See Also

the slightly more flexible nearPD which also returns a classed matrix (class dpoMatrix). For new code, nearPD() is really preferred to nearcor(), which hence is considered deprecated.
hetcor, eigen; posdefify for a simpler algorithm.

Examples

cat("pr is the example matrix used in Knol DL, ten Berge (1989)\n")
pr <- matrix(c(1, 0.477, 0.644, 0.478, 0.651, 0.826, 0.477, 1, 0.516, 0.233, 0.682, 0.75, 0.644, 0.516, 1, 0.599, 0.581, 0.742, 0.478, 0.233, 0.599, 1, 0.741, 0.8, 0.651, 0.682, 0.581, 0.741, 1, 0.798, 0.826, 0.75, 0.742, 0.8, 0.798, 1),
nrow = 6, ncol = 6)
ncr <- nearcor(pr)
nr <- ncr$cor
plot(pr[lower.tri(pr)],
     nr[lower.tri(nr)]); abline(0,1, lty=2)
round(cbind(eigen(pr)$values, eigen(nr)$values), 8)
cat("The following will fail:\n")
try(factanal(cov=pr, factors=2))
cat("and this should work\n")
try(factanal(cov=nr, factors=2))

if(require("polycor") {  
  n <- 400
  x <- rnorm(n)
  y <- rnorm(n)
  x1 <- (x + rnorm(n))/2
  x2 <- (x + rnorm(n))/2
  x3 <- (x + rnorm(n))/2
  x4 <- (x + rnorm(n))/2
  y1 <- (y + rnorm(n))/2
  y2 <- (y + rnorm(n))/2
  y3 <- (y + rnorm(n))/2
  y4 <- (y + rnorm(n))/2
  dat <- data.frame(x1, x2, x3, x4, y1, y2, y3, y4)
x1 <- ordered(as.integer(x1 > 0))
nr.sign.chg

Description

Compute the number of sign changes in the sequence y.

Usage

sr.sign.chg(y)

Arguments

y       numeric vector.

Value

an integer giving the number of sign changes in sequence y. Note that going from positive to 0 to positive is not a sign change.

Author(s)

Martin Maechler, 17 Feb 1993.

Examples

(y <- c(1:2,1:-1,0:-2))
nr.sign.chg(y)# = 1
p.arrows  

**Prettified Arrows Plots**

**Description**

Draws arrows, like the `arrows` function, but with “nice” filled arrow heads.

**Usage**

```
p.arrows(x1, y1, x2, y2, size = 1, width, fill = 2, ...)
```

**Arguments**

- `x1, y1`: coordinates of points from which to draw.
- `x2, y2`: coordinates of points to which to draw.
- `size`: symbol size as a fraction of a character height; default 1.
- `width`: width of the arrow head; defaults to ...
- `fill`: color for filling the arrow head.
- `...`: further arguments passed to `segments()`.

**Author(s)**

Andreas Ruckstuhl, 19 May 1994; (cosmetic by MM).

**See Also**

`arrows`.

**Examples**

```
example(arrows, echo = FALSE) #-> x, y, s
plot(x,y, main="p.arrows(.)")
p.arrows(x[3], y[3], x[3]+1, y[3]+1, col= 1:3, fill = "dark blue")
```

---

p.datum  

**Plot 'Datum' (deutsch!) unten rechts**

**Description**

Plot the date (and time, if required) in German, at the lower right hand margin of your plot.

**Usage**

```
p.datum(outer = FALSE, cex = 0.75, ...)
```
**p.dnorm**

Arguments

- **outer**: logical; passed to `mtext`.
- **cex**: non-negative; passed to `mtext`.
- **...**: any arguments to `u.Datumvonheute`.

See Also

- `u.date`, `date`.

Examples

```r
plot(1)
p.datum()
```

---

**Plot Parametric Density Functions**

Description

These are utilities for pretty plotting of often used parametric densities.

Usage

```r
p.dnorm(mu = 0, s = 1, h0.col = "light gray",
        ms.lines = TRUE, ms.col = "gray", ...)
p.dchisq(nu, h0.col = "light gray", ...)
p.dgamma(shape, h0.col = "light gray", ...)
```

Arguments

- **mu, s**: numbers, the mean and standard deviation of the normal distribution.
- **nu**: positive number, the degrees of freedom df argument for the \( \chi^2 \)-density function `dchisq`.
- **shape**: number, the shape parameter for the Gamma distribution.
- **h0.col**: color specification for the line \( y = 0 \).
- **ms.lines**: logical, used for the normal only: should lines be drawn at the mean and \( \pm 1 \) standard deviation.
- **ms.col**: color for the ms lines if ms.lines is TRUE.
- **...**: further parameter passed to `curve()`, e.g., `add = TRUE` for adding to current plot.

Author(s)

Werner Stahel et al.
See Also

the underlying density functions, `dnorm`, `dchisq`, `dgamma`.

Examples

```r
p.dnorm()
p.dnorm(mu=1.5, add = TRUE, ms.lines = FALSE) # add to the plot above

p.dchisq(2, main= "Chi^2 Densities -- nu = 2,3,4")
p.dchisq(3, add = TRUE, col = "red")
p.dchisq(4, add = TRUE, col = "blue")

op <- par(mfrow = c(2,2), mgp = c(1.6, 0.6,0), mar = c(3,3,1,1))
for(sh in 1:4)
  p.dgamma(sh)
par(op)
```

---

Add a Horizontal Boxplot to the Current Plot

Description

Add a horizontal boxplot to the current plot. This is mainly an auxiliary function for `histBxp`, since `boxplot(*, horizontal = TRUE, add = TRUE)` is usually much preferable to this.

Usage

```r
p.hboxp(x, y.lo, y.hi, boxcol = 3,
  medcol = 2, medlwd = 5, whisklty = 2, staplelty = 1)
```

Arguments

- `x` : univariate data set.
- `y.lo`, `y.hi` : minimal and maximal *user* coordinates or `y.lo = c(ylo,hyi)`.
- `boxcol`, `medcol` : color of the box and the median line.
- `medlwd` : line width of median line.
- `whisklty`, `staplelty` : line types of the whisker and the staple, the latter being used for the outmost non-outliers.

Details

```
```

Author(s)

Martin Maechler building on code from Markus and Christian Keller.
See Also

boxplot(**, horizontal = TRUE, add = TRUE).

Examples

```r
## => See code in 'histBxp' (.) and example(histBxp)!
##
```

---

### p.profileTraces

*Plot a profile.nls Object With Profile Traces*

**Description**

Displays a series of plots of the profile t function and the likelihood profile traces for the parameters in a nonlinear regression model that has been fitted with `nls` and profiled with `profile.nls`.

**Usage**

```r
p.profileTraces(x, cex = 1, 
    subtitle = paste("t-Profiles and traces of ",
                         deparse(attr(x,"summary")$formula)))
```

**Arguments**

- `x` an object of class "profile.nls", typically resulting from `profile(nls(.))`, see `profile.nls`.
- `cex` character expansion, see `par(cex =)`.
- `subtitle` a subtitle to set for the plot. The default now includes the `nls()` formula used.

**Note**

the stats::stats::plot.profile.nls plot method just does “the diagonals”.

**Author(s)**

Andreas Ruckstuhl, R port by Isabelle Flückiger and Marcel Wolbers

**See Also**

`profile`, and `nls` (which has unexported profile and stats::stats::plot.profile.nls methods).
Examples

```r
require(stats)
data(Puromycin)
Treat <- Puromycin[Puromycin$state == "treated", ]
fm <- nls(rate ~ T1*conc/(T2+conc), data=Treat,
          start = list(T1=200,T2=0.0001))
(pr <- profile(fm)) # quite a few things..
op <- par(mfcol=1:2)
plot(pr) # -> 2 'standard' plots
par(op)
## ours:
p.profileTraces(pr)
```

Description

Plots a numeric “residual like” variable against two factor covariates, using boxplots.

Usage

```r
p.res.2fact(x, y, z, restricted, notch = FALSE,
           xlab = NULL, ylab = NULL, main = NULL)
```

Arguments

- `x, y` two factors or numeric vectors giving the levels of factors.
- `z` numeric vector of same length as `x` and `y`, typically residuals.
- `restricted` positive value which truncates the size. The corresponding symbols are marked by stars.
- `notch` logical indicating if the boxplots should be notched, see `boxplot(*, notch)`.
- `xlab, ylab` axis labels, see `plot.default`, per default the actual argument expressions.
- `main` main title passed to `plot`, defaulting to the deparsed `z` argument.

Details

if values are restricted, this make use of the auxiliar function `u.boxplot.x`.

Author(s)

Lorenz Gygax <logyg@wild.unizh.ch> and Martin Maechler, Jan.95; starting from `p.res.2x()`.

See Also

`p.res.2x, boxplot.plot.lm, TA.plot`. 
Examples

```r
I <- 8; J <- 3; K <- 20
xx <- factor(rep(rep(1:1, rep(K,I)),J))
yy <- factor(rep(I:1, rep(I*K,J)))
zz <- rt(I*K+J, df=5) #-- Student t with 5 d.f.
p.res.2fact(xx,yy,zz, restr= 4, main= "i.i.d. t <= 5 random |.| <= 4")
```

```r
p.res.2fact(xx,yy,zz, restr= 4, ..)
```

```r
mtext("p.res.2fact(xx,yy,zz, restr= 4, ..)",
line=1, adj=1, outer=TRUE, cex=1)
```

```r
## Real data
data(warpbreaks)
(fml <- lm(breaks ~ wool*tension, data = warpbreaks))
## call via formula method of p.res.2x():
p.res.2x(~ ., fml) # is shorter than, but equivalent to
## p.res.2x(~ wool + tension, fml) ## or the direct
## with(warpbreaks, p.res.2fact(wool, tension, residuals(fml)))
##
## whereas this is "transposed":
p.res.2x(~ tension+wool, fml)
```

**p.res.2x**

Stahel’s Residual Plot against 2 X’s

Description

Plot Residuals (e.g., of a multiple linear regression) against two (predictor) variables. This is now a (S3) generic function with a default and a formula method.

Usage

```r
p.res.2x(x, ...)```

### Default S3 method:

```r
p.res.2x(x, y, z, restricted, size = 1, slwd = 1, scol = 2:3,
  xlab = NULL, ylab = NULL, main = NULL,
  xlim = range(x), ylim = range(y), ...)
```

### S3 method for class 'formula'

```r
p.res.2x(x = ~ ., data, main = deparse(substitute(data)),
  xlab = NULL, ylab = NULL, ...)
```

Arguments

- `x, y` numeric vectors of the same length specifying 2 covariates. For the formula method, `x` is a formula.
- `z` numeric vector of same length as `x` and `y`, typically residuals.
- `restricted` positive value which truncates the size. The corresponding symbols are marked by stars.
size

the symbols are scaled so that size is the size of the largest symbol in cm.

slwd, scol

line width and color(s) for the residual segments. If scol has length 2 as per default, the two colors are used for positive and negative \( z \) values, respectively.

xlab, ylab, main

axis labels, and title see title, each with a sensible default. To suppress, use, e.g., \( \text{main} = "" \).

xlim, ylim

the basic x- and y- axis extents, see \text{plot.default}. Note that these will be slightly extended such that segments are not cut off.

... further arguments passed to \text{plot}, or \text{p.res.2x.default()}, respectively.

data (for the \text{formula} method:) a data frame or a fitted "lm" object.

Details

The formula interface will call \text{p.res.2fact()} when both \( x \) and \( y \) are factors.

.................

Author(s)


References


See Also

\text{p.res.2fact, plot.lm, TA.plot}.

Examples

\begin{verbatim}
xx <- rep(1:10,7)
yy <- rep(1:7, rep(10,7))
zz <- rnorm(70)
p.res.2x(xx,yy,zz, restr = 2, main = "i.i.d. N(0,1) random residuals")
example(lm.influence, echo = FALSE)

op <- mult.fig(2, marP=c(-1,-1,-1,0), main="p.res.2x(*,* , residuals(lm.SR))")$old.par
with(LifeCycleSavings,
  ( p.res.2x(pop15, dpi, residuals(lm.SR), scol=c("red", "blue"))
    p.res.2x(pop75, dpi, residuals(lm.SR), scol=2:1)
  ))

## with formula interface:
p.res.2x(~ pop15 + dpi, lm.SR, scol=c("red", "blue"))
p.res.2x(~ pop75 + dpi, lm.SR, scol=2:1)

par(op) # revert par() settings above
\end{verbatim}
p.scales

Conversion between plotting scales: `usr`, `cm`, `symbol`

Description
Give scale conversion factors of three coordinate systems in use for traditional R graphics: `use`, `cm`, `symbol`.

Usage
`p.scales(unit = relsysize * 2.54 * min(pin), relsysize = 0.05)`

Arguments
- `unit`: length of unit (or x and y units) of symbol coordinates in cm.
- `relsysize`: same, as a proportion of the plotting area.

Value
A numeric 2x2 matrix, with rows named `x` and `y`, and columns, named `sy2usr` and `usr2cm` which give the scale conversion factors from `symbol` (as given) to `usr` coordinates and from these to `cm`, respectively.

Author(s)
Werner Stahel, 1990; simplification: M.Maechler, 1993, 2004

See Also
- `par("usr")`, of also `("pin")` on which this is based.

Examples
`p.scales()`

p.tachoPlot

Draw Symbol on a Plot

Description
Puts a symbol (pointer) on a plot at each of the specified locations.
Usage

p.tachoPlot(x, y, z, angle=c(pi/4,3*pi/4), size,
  method = c("robust", "sensitive", "rank"),
  legend = TRUE, show.method = legend,
  xlab = deparse(substitute(x)), ylab = deparse(substitute(y)),
  xlim, ylim, ...) 

Arguments

x, y, z  coordinates of points. Numeric vectors of the same length. Missing values (NAs) are allowed.
angle   numeric vector whose elements give the angles between the horizontal baseline and the minimum and maximum direction of the pointer measured clockwise in radians.
size    length of the pointers in cm.
method  string specifying the method to calculate the angle of the pointer. One of "sensitive", "robust" or "rank". Only the first two characters are necessary. The minimum and maximum direction of the pointer corresponds to min(z) and max(z) if method is "sensitive" or "rank" and to the upper and lower extreme of z if method is "robust" (see boxplot or rrange for details). The angle is proportional to z or rank(z) in case of method="rank".
legend  logical flag: if TRUE (default), a legend giving the values of the minimum and maximum direction of the pointer is drawn.
show.method  logical flag, defaulting to legend; if true, the method name is printed.
xlab, ylab labels for x and y axis; defaults to the ‘expression’ used in the function call.
xlim, ylim  numeric of length 2, the limits for the x and y axis, respectively; see plot.default.
...        further arguments to plot. Graphical parameters (see par) may also be supplied as arguments to this function.

Details

A scatter plot of the variables x and y is plotted. The value of the third variable z is given by the direction of a pointer (similar to a tachometer). Observations whose z-coordinate is missing are marked by a dot.

Side Effects

A plot is created on the current graphics device.

Author(s)

Christian Keller, June 1995

See Also

symbols
Examples

data(state)
data(USArrests)
p.tachoPlot(state.center $x$, state.center $y$, USArrests[,"UrbanPop"])

data(mtcars)
par(mfrow=c(2,2))
## see the difference between the three methods (not much differ. here!)
p.tachoPlot(mtcars$hp, mtcars$disp, mtcars$mpg, method="sens")
p.tachoPlot(mtcars$hp, mtcars$disp, mtcars$mpg, method="rank")
p.tachoPlot(mtcars$hp, mtcars$disp, mtcars$mpg, method="rob")

Description

For longer time-series, it is sometimes important to spread the time-series plots over several sub-
plots. p.ts(.) does this both automatically, and under manual control.
Actually, this is a generalization of plot.ts (with different defaults).

Usage

p.ts(x, nrplots = max(1, min(8, n/16)), overlap = nk/16, 
date.x = NULL, do.x.axis = !is.null(date.x), do.x.rug = FALSE, 
ax.format, main.tit = NULL, ylim = NULL, ylab = "", xlab = "Time", 
quiet = FALSE, mgp = c(1.25, .5, 0), ...) 

Arguments

x timeseries (possibly multivariate) or numeric vector.
nrplots number of sub-plots. Default: in [1..8], approximately n/400 if possible.
overlap by how much should subsequent plots overlap. Defaults to about 1/16 of sub-
length on each side.
date.x a time “vector” of the same length as x and coercable to class "POSIXct" (see 
Date Time Classes).
do.x.axis logical specifying if an x axis should be drawn (i.e., tick marks and labels).
do.x.rug logical specifying if rug of date.x values should drawn along the x axis.
ax.format when do.x.axis is true, specify the format to be used in the call to axis.POSIXct.
main.tit Main title (over all plots). Defaults to name of x.
ylim numeric(2) or NULL; if the former, specifying the y-range for the plots. Defaults
to a common pretty range.
ylab, xlab labels for y- and x-axis respectively, see description in plot.default.
quiet logical; if TRUE, there’s no reporting on each subplot.
mgp numeric(3) to be passed to mult.fig(), see par(mgp = ).
... further graphic parameters for each plot.ts(..).

Side Effects
A page of nrplots subplots is drawn on the current graphics device.

Author(s)
Martin Maechler, <maechler@stat.math.ethz.ch>; July 1994 (for S).

See Also
p.ts() calls mult.fig() for setup. Further, plot.ts and plot.

Examples
stopifnot(require(stats))
## stopifnot(require(datasets))

data(sunspots)
p.ts(sunspots, nr=1) # == usual  plot.ts(..)
p.ts(sunspots)
p.ts(sunspots, nr=3, col=2)

data(EuStockMarkets)
p.ts(EuStockMarkets,"SMI")
## multivariate :
p.ts(log10(EuStockMarkets), col = 2:5)

## with Date - x-axis (dense random dates):
set.seed(12)
x <- as.Date("2000-02-29") + cumsum(1+ rpois(1000, lambda= 2.5))
z <- cumsum(.1 + 2*rt(1000, df=3))
p.ts(z, 4, date.x = x)
p.ts(z, 6, date.x = x, ax.format = "%b %Y", do.x.rug = TRUE)
Arguments

name          string with a variable name which must exist in the current environment (R session).
digits        how many decimal digits to be used; passed to format.

Value

a string of the form "NAME = x1 x2 ..."

Author(s)

Martin Maechler, about 1992.

Examples

```r
x <- 1:4
paste.vec(x)  ###->  "x = 1 2 3 4"
```

---

**plotDS**  
*Plot Data and Smoother / Fitted Values*

Description

For one-dimensional nonparametric regression, plot the data and fitted values, typically a smooth function, and optionally use segments to visualize the residuals.

Usage

```r
plotDS(x, yd, ys, xlab = "", ylab = "", ylim = rrange(c(yd, ys)),
       xpd = TRUE, doseg = TRUE, segp = 0.95,
       segP = list(lty = 2, lwd = 1, col = 2),
       linP = list(lty = 1, lwd = 2.5, col = 3),
       ...)```

Arguments

- `x, yd, ys` numeric vectors all of the same length, representing \((x_i, y_i)\) and fitted (smooth) values \(\hat{y}_i\). \(x\) will be sorted increasingly if necessary, and \(yd\) and \(ys\) accordingly. Alternatively, \(ys\) can be an \(x\)-\(y\) list (as resulting from `xy.coords`) containing fitted values on a finer grid than the observations \(x\). In that case, the observational values \(x[]\) **must** be part of the larger set; `seqXtend()` may be applied to construct such a set of abscissa values.

- `xlab, ylab` `x`- and `y`- axis labels, as in `plot.default`.

- `ylim` limits of `y`-axis to be used; defaults to a **robust** range of the values.

- `xpd` see `par(xpd=.)`; by default do allow to draw outside the plot region.
do.seg logical indicating if residual segments should be drawn, at \( x[i] \), from \( yd[i] \) to \( ys[i] \) (approximately, see seg.p).

seg.p segment percentage of segments to be drawn, from \( yd \) to \( seg.p \cdot yd + (1-seg.p) \cdot ys \).

segP list with named components \( \text{lty, lwd, col} \) specifying line type, width and color for the residual segments, used only when \( \text{do.seg} \) is true.

linP list with named components \( \text{lty, lwd, col} \) specifying line type, width and color for “smooth curve lines”.

... further arguments passed to \text{plot}.

\textbf{Note}

Non-existing components in the lists \( \text{segP} \) or \( \text{linP} \) will result in the \text{par} defaults to be used.

\text{plotDS()} \ used to be called \text{pl.ds} \ up to November 2007.

\textbf{Author(s)}

Martin Maechler, since 1990

\textbf{See Also}

\text{seqXtend()} \ to construct more smooth \( \text{ys} \) “objects”.

\textbf{Examples}

data(cars)
x <- cars$speed
yd <- cars$dist
ys <- lowess(x, yd, f = .3)$y
plotDS(x, yd, ys)

## More interesting : Version of example(Theoph)
data(Theoph)
Th4 <- subset(Theoph, Subject == 4)
## just for “checking” purposes -- permute the observations:
Th4 <- Th4[sample(nrow(Th4)), ]
fm1 <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Th4)

## Simple
plotDS(Th4$Time, Th4$conc, fitted(fm1),
    sub = "Theophylline data - Subject 4 only",
    segP = list(lty=1, col=2), las = 1)

## Nicer: Draw the smoother not only at \( x = x[i] \) (observations):
xsm <- unique(sort(c(Th4$Time, seq(0, 25, length = 201))))
ysm <- c(predict(fm1, newdata = list(Time = xsm)),
    sub = "Theophylline data - Subject 4 only",
    segP = list(lwd=2), las = 1)
Plot a Step Function

Description

Plots a step function \( f(x) = \sum_i y_i 1_{[t_i-1, t_i]}(x) \), i.e., a piecewise constant function of one variable. With one argument, plots the empirical cumulative distribution function.

Usage

```
plotStep(ti, y, 
    cad.lag = TRUE, 
    verticals = !cad.lag, 
    left.points= cad.lag, right.points= FALSE, end.points= FALSE, 
    add = FALSE, 
    pch = par('pch'), 
    xlab=deparse(substitute(ti)), ylab=deparse(substitute(y)), 
    main=NULL, ...) 
```

Arguments

- `ti` numeric vector = \( X[1:N] \) or \( t[0:n] \).
- `y` numeric vector \( y[1:n] \); if omitted take \( y = k/N \) for empirical CDF.
- `cad.lag` logical: Draw ‘cad.lag’, i.e., “continue à droite, limite à gauche”. Default = TRUE.
- `verticals` logical: Draw vertical lines? Default= ! cad.lag
- `left.points` logical: Draw left points? Default= cad.lag
- `right.points` logical: Draw right points? Default= FALSE
- `end.points` logical: Draw 2 end points? Default= FALSE
- `add` logical: Add to existing plot? Default= FALSE
- `pch` plotting character for points, see `par()`.
- `xlab,ylab` labels of x- and y-axis
- `main` main title; defaults to the call’ if you do not want a title, use `main = ""`. 
- `...` Any valid argument to `plot(..)`.

Value

invisibly: List with components `t` and `y`.

Side Effects

Calls `plot(..)`, `points(..)`, `segments(..)` appropriately and plots on current graphics device.
polyn.eval

Evaluate Polynomials

Description

Evaluate one or several univariate polynomials at several locations, i.e. compute coef[1] + coef[2]*x + ... + coef[p+1] (in the simplest case where x is scalar and coef a vector).

Usage

polyn.eval(coef, x)

Arguments

coef numeric vector or matrix. If a vector, x can be an array and the result matches x. If coef is a matrix it specifies several polynomials of the same degree as rows, x must be a vector, coef[,k] is for \(x^{k-1}\) and the result is a matrix of dimension length(x) * nrow(coef).

x numeric vector or array. Either x or coef must be a vector.

Details

The stable “Horner rule” is used for evaluation in any case.
**Find a Close Positive Definite Matrix**

From a matrix \( m \), construct a "close" positive definite one.

**Usage**

\[
\text{posdefify}(m, \text{method} = \text{c("someEVadd", "allEVadd")}, \\
\text{symmetric} = \text{TRUE}, \text{eigen.m} = \text{eigen}(m, \text{symmetric} = \text{symmetric}), \\
\text{eps.ev} = 1e-07)
\]

**Arguments**

- \( m \) a numeric (square) matrix.
- \( \text{method} \) a string specifying the method to apply; can be abbreviated.
- \( \text{symmetric} \) logical, simply passed to \text{eigen} (unless \text{eigen.m} is specified); currently, we do not see any reason for \text{not} using \text{TRUE}.
- \( \text{eigen.m} \) the \text{eigen} value decomposition of \( m \), can be specified in case it is already available.
- \( \text{eps.ev} \) number specifying the tolerance to use, see Details below.

**Description**

The function `posdefify` is designed to take a matrix and return a positive definite matrix that is as close as possible to the original one, using various methods to achieve this goal. The `method` argument allows specifying the method used for the transformation, with options for `someEVadd` and `allEVadd`. The `symmetric` argument ensures that the resulting matrix is symmetric, which is a requirement for positive definite matrices. The `eigen.m` argument allows providing an existing eigen decomposition, which can be useful in certain scenarios. The `eps.ev` argument specifies the tolerance level for determining the closeness of the resulting matrix to the original one.

**Examples**

```r
polyn.eval(c(1,-2,1), x = 0:3)# (x - 1)^2
polyn.eval(c(0, 24, -50, 35, -10, 1), x = matrix(0:5, 2,3))# 5 zeros!
(cf <- rbind(diag(3), c(1,-2,1)))
polyn.eval(cf, 0:5)
```

**Value**

A numeric vector or array, depending on input dimensionalities, see above.
Details

We form the eigen decomposition

\[ m = V \Lambda V' \]

where \( \Lambda \) is the diagonal matrix of eigenvalues, \( \Lambda_{j,j} = \lambda_j \), with decreasing eigenvalues \( \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \).

When the smallest eigenvalue \( \lambda_n \) are less than \( \text{Eps} \approx \text{eps} \cdot \text{ev} \cdot \text{abs} \{ \text{lambda}[1] \} \), i.e., negative or “almost zero”, some or all eigenvalues are replaced by positive (\( \geq \text{Eps} \)) values, \( \tilde{\Lambda}_{j,j} = \tilde{\lambda}_j \).

Then, \( \tilde{m} = V \tilde{\Lambda} V' \) is computed and rescaled in order to keep the original diagonal (where that is \( \geq \text{Eps} \)).

Value

a matrix of the same dimensions and the “same” diagonal (i.e. \( \text{diag} \)) as \( m \) but with the property to be positive definite.

Note

As we found out, there are more sophisticated algorithms to solve this and related problems. See the references and the \texttt{nearPD()} function in the \texttt{Matrix} package.

Author(s)

Martin Maechler, July 2004

References


See Also

\texttt{eigen} on which the current methods rely. \texttt{nearPD()} in the \texttt{Matrix} package.
Examples

```r
set.seed(12)
m <- matrix(round(rnorm(25), 2), 5, 5); m <- 1 + m + t(m); diag(m) <- diag(m) + 4
m
posdefify(m)
1000 * zapsmall(m - posdefify(m))
```

---

### Description

Fisher’s potato crop data set is of historical interest as an early example of a multi-factor block design.

### Usage

```r
data(potatoes)
```

### Format

A data frame with 64 observations on the following 5 variables.

- **pos** a factor with levels 1:4.
- **treat** a factor with 16 levels A to H and J to Q, i.e., LETTERS[1:17][-9].
- **nitrogen** a factor specifying the amount of nitrogen sulfate (\(NH_4\)), with the four levels 0, 1, 2, 4.
- **potash** a factor specifying the amount of potassium (K, ‘kalium’) sulfate, with the four levels 0, 1, 2, 4.
- **yield** a numeric vector giving the yield of potatoes in ...

### Source


### References

Examples

```r
data(potatoes)
## See the experimental design:
with(potatoes, {
  cat("4 blocks of experiments; ", 
      "each does every (nitrogen, potash) combination (aka 'treatment') once. ", 
      ", sep="\n")
  print(ftable(table(nitrogen, potash, treat)))
  print(ftable(tt <- table(pos, potash, nitrogen)))
  tt[cbind(pos, potash, nitrogen)] <- as.character(treat)
  cat("The 4 blocks pos = 1, 2, 3, 4: \n")
  ftable(tt)
})
## First plot:
with(potatoes, interaction.plot(potash, nitrogen, response=yield))
## ANOVAs:
summary(aov(yield ~ nitrogen * potash + Error(pos), data = potatoes))
  # "=" can use simply
summary(aov(yield ~ nitrogen + potash + pos, data = potatoes))
  # and
summary(aov(yield ~ nitrogen + potash, data = potatoes))
```

---

**pretty10exp**

*Nice $10^k$ Label Expressions*

**Description**

Produce nice $a \times 10^k$ expressions to be used instead of the scientific notation "$a \ E<k>$".

**Usage**

```r
pretty10exp(x, drop.1 = FALSE, sub10 = FALSE, digits = 7, digits.fuzz, lab.type = c("plotmath","latex"), lab.sep = c("cdot","times"))
```

**Arguments**

- `x` numeric vector (e.g. axis tick locations)
- `drop.1` logical indicating if $1 \times$ should be dropped from the resulting expressions.
- `sub10` logical, "$10^k$", a non-negative integer number or an integer vector of length two, say $(k_1, k_2)$, indicating if some $10^j$ expressions for $j \in J$ should be formatted traditionally, notably e.g., $10^0 \equiv 1$.

When a (non-negative) number, say $k$, $J = \{ j; j \leq k \}$ are all simplified, when a length-2 vector, $J = \{ j; k_1 \leq j \leq k_2 \}$ are.

Special cases: $\text{sub10} = \text{TRUE}$ means to use $1$ instead of $10^0$ and $\text{sub10} = "10^k"$ uses both $1$ for $10^0$ and $10$ for $10^1$; these are short forms of $\text{sub10} = \text{c}(0,0)$ and $\text{sub10} = \text{c}(0,1)$ respectively.
digits number of digits for mantissa (a) construction; the number of **significant** digits, see `signif`.

digits.fuzz the old deprecated name for `digits`.

lab.type a string indicating how the result should look like. By default, (plotmath-compatible) **expressions** are returned. Alternatively, `lab.type = "plotmath"` returns LaTeX formatted strings for labels. (The latter is useful, e.g., when using the `tikzDevice` package to generate LaTeX-processed figures.)

lab.sep character separator between mantissa and exponent for LaTeX labels; it will be prepended with a backslash, i.e., "\cdot" will use "\cdot"

**Value**

For the default `lab.type = "plotmath"`, an expression of the same length as `x`, typically with elements of the form a %*% 10 ^ k. Exceptions are `0` which is kept simple, if `drop.1` is true and `a = 1, 10 ^ k` is used, and if `sub10` is not false, a %*% 10 ^ 0 as `a`, and a %*% 10 ^ k as as the corresponding formatted number a * 10 ^ k independently of `drop.1`.

Otherwise, a character vector of the same length as `x`. For `lab.type = "latex"`, currently the only alternative to the default, these strings are LaTeX (math mode) compatible strings.

**Note**

If `sub10` is set, it will typically be a small number such as 0, 1, or 2. Setting `sub10 = TRUE` will be interpreted as `sub10 = 1` where resulting exponents `k` will either be negative or `k ≥ 2`.

**Author(s)**

Martin Maechler; Ben Bolker contributed `lab.type = "latex"` and `lab.sep`.

**See Also**

`axTexpr` and `eaxis()` which build on `pretty10exp()`, notably the `eaxis()` example plots.

The new `tolatex.numeric` method which gives very similar results with option `scientific = TRUE`.

Further, `axis`, `axTicks`.

**Examples**

```r
pretty10exp(-1:3 * 1000)
pretty10exp(-1:3 * 1000, drop.1 = TRUE)
pretty10exp(c(1,2,5,10,20,50,100,200) * 1e3)
pretty10exp(c(1,2,5,10,20,50,100,200) * 1e3, drop.1 = TRUE)

set.seed(17); lx <- rlnorm(10, m=8, s=6)
pretty10exp(lx, digits = 3)
pretty10exp(lx, digits = 3, sub10 = 2)

unlist(pretty10exp(lx, digits = 3, lab.type="latex"))
unlist(pretty10exp(lx, digits = 3, lab.type="latex",
  lab.sep="times", sub10=2))
```
primes <- 10^(-6:0) - 2e-16
pretty10exp(ax, drop.1=TRUE) # nice for plotting
pretty10exp(ax, drop.1=TRUE, sub10=TRUE)
pretty10exp(ax, drop.1=TRUE, sub10=c(-2,2))

## in sfsmisc version <= 1.0-16, no 'digits',
## i.e., implicitly had digits := #(double precision digits) ==
## (dig. <- .Machine$double.digits * log10(2)) # 15.95
pretty10exp(ax, drop.1=TRUE, digits= dig.) # 'ugly'

| primes
| ----- | ----- |
| Find all Primes Less Than n

**Description**

Find all prime numbers aka ‘primes’ less than \( n \).

Uses an obvious sieve method (and some care), working with **logical** and **integer**s to be quite fast.

**Usage**

```r
primes(n, pSeq = NULL)
```

**Arguments**

- **n**: a (typically positive integer) number.
- **pSeq**: optionally a vector of primes (2,3,5,...) as if from a `primes()` call; **must** be correct. The goal is a speedup, but currently we have not found one single case, where using a non-NULL `pSeq` is faster.

**Details**

As the function only uses `max(n)`, \( n \) can also be a **vector** of numbers.

The famous prime number theorem states that \( \pi(n) \), the *number* of primes below \( n \) is asymptotically \( n/\log(n) \) in the sense that \( \lim_{n \to \infty} \pi(n) \cdot \log(n)/n \sim 1 \).

Equivalently, the inverse of \( \pi(n) \), the \( n \)-th prime number \( p_n \) is around \( n \log n \); recent results (Pierre Dusart, 1999), prove that

\[
\log n + \log \log n - 1 < \frac{p_n}{n} < \log n + \log \log n \quad \text{for} \ n \geq 6.
\]

**Value**

numeric vector of all prime numbers \( \leq n \).
Author(s)

Bill Venables (<= 2001); Martin Maechler gained another 40% speed, carefully working with logi-
cals and integers.

See Also

factorize. For large $n$, use the gmp package and its isprime and nextprime functions.

Examples

```r
(p1 <- primes(100))
system.time(p1k <- primes(1000)) # still lightning fast
stopifnot(length(p1k) == 160)

system.time(p.e7 <- primes(1e7)) # still only 0.3 sec (2015 i7)
stopifnot(length(p.e7) == 664579)
## The famous p(n) := number of primes <= n:
p.i.n <- approxfun(p.e7, seq_along(p.e7), method = "constant")
p.i.n(c(10, 100, 1000)) # 4 25 168
plot(p.i.n, 2, 1e7, n = 1024, log="xy", axes = FALSE,
    xlab = "n", ylab = quote(pi(n)),
    main = quote("The prime number function " ~ pi(n))))
exaxis(1); eaxis(2)

## Exploring p(n) := the n-th prime number :
## pnn(n) := log n + log log n
pnn <- function(n) { L <- log(n); L + log(L) }
n <- 6:(N <- length(PR <- primes(1e5)))
m.pn <- cbind(l.pn = ceiling(n*(pnn(n)-1)), pn = PR[n], u.pn = floor(n*pnn(n)))
matplot(n, m.pn, type="l", ylab = quote(pn), main = quote(pn) ~~
    "with lower/upper bounds" ~ n*(log(n) + log(log(n))) -(1-"or"-0)))
plot(n, PR[n]/n - (pnn(n)-1), type = 'l', cex = 1/8, log="x", xaxt="n")
exaxis(1); abline(h=0, col=adjustcolor(1, 0.5))
```

printTable2

**Add and Print Marginals for 2-way Contingency Tables**

Description

printTable2() prints a 2-way contingency table “with all bells and whistles” (currently using
German labeling).

margin2table() computes marginals, adds them to the table and returns a margin2table object
the print method for which adds text decorations (using "-" and "+").

Usage

```r
printTable2(table2, digits = 3)
margin2table(x, totName = "sum", name.if.empty=FALSE)
## S3 method for class 'margin2table'
print(x, digits = 3, quote = FALSE, right = TRUE, ...)
```
Arguments

- `table2`: a matrix with non-negative integer entries, i.e. the contingency table.
- `x`: a matrix; for `print()`, the result of `margin2table`.
- `digits`: Anzahl Dezimalstellen, auf die die Häufigkeiten gerundet werden sollen.
- `quote, right`: logicals passed to `print.default()`, but with different default values.
- `totName`: string to use as row- and column- name if `x` has corresponding `dimnames`.
- `name.if.empty`: logical indicating if the margin “totals” should be named in any case.
- `...`: further potential arguments, unused currently.

Value

`margin2table` returns a matrix with *added marginals*, i.e., an extra row and column, and is of class “`margin2table`” (and "`table`" still) which has a nice print method.

`printTable2` is just producing output.

Author(s)

Martin Maechler, Feb.1993; then Dec 2003

See Also

table, ftable.

Examples

```r
margin2table(diag(4), TRUE)
m <- diag(3): colnames(m) <- letters[1:3]
margin2table(m)
margin2table(m / sum(m))

data(HairEyeColor)
margin2table(HairEyeColor[, "Male"])
printTable2(HairEyeColor[, "Male"])
printTable2(HairEyeColor[, "Female"])
```

---

### Description

This is `defunct` now: The global `DEBUG` has been a cheap precursor to R’s `options(verbos e= .)` (or a verbose function argument).

This function prints out its arguments as `cat()` does, additionally printing the name of function in which it’s been called — only when a global variable `DEBUG` exists and is `TRUE`. 
Usage

ps.end(prt.DEBUG(..., LEVEL = 1))

Arguments

... arguments to be passed to cat(...) for printing.
LEVEL integer (or logical) indicating a debugging level for printing.

Author(s)

Martin Maechler, originally for S-PLUS.

Description

Closes the PostScript or PDF file (postscript.pdf), openend by a previous ps.do (or pdf.latex, or ...) call, using dev.off, and additionally opens a previewer for that file, unless the previewer is already up. This almost provides an ‘interactive’ device (like x11) for postscript or pdf.

Usage

ps.end(call.gv= NULL, command = getOption("eps_view"),
       debug = getOption("verbose"))
pdf.end(call.viewer= NULL, command = getOption("pdfviewer"),
        debug = getOption("verbose"))

Arguments

call.gv,call.viewer
  logical, indicating if the postscript or acrobat reader (e.g., ghostview or acroread or the command given by command) should be called. By default, find out if the viewer is already running on this file and only call it if needed.

command character, giving a system command for PostScript previewing. By default, getOption("eps_view") is set to gv -watch -geometry -8+0 -magstep -2 -media BBBox -noantialias which assumes gv (aka ghostview) to be in your OS path.

debug logical; if TRUE print information during execution.

Details

Depends on Unix tools, such as ps.
Author(s)
Martin Maechler

See Also
postscript, postscript pdf.do, ps.do, ...

Examples
if(interactive())
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
 |
Arguments

- **file**: character giving the PostScript/PDF file name to be written.
- **height**: device height in *inches*, height * 2.54 are *cm*. The default is 5 plus 1.25 if not main.space.
- **width**: device width in *inches*; for this and height, see *postscript*.
- **ONeFile**, **horizontal**: logicales passed to *postscript*() or *pdf*(), most probably to be left alone.
- **title**: PostScript/PDF (not plot!) title passed to *postscript*() or *pdf*(); by default use a title with *R* version and file in it.
- **version**: a string describing the PDF version that will be required to view the output, see *pdf*; our (high) default ensures alpha-transparency.
- **quiet**: logical specifying that some (informative/warning) messages should not be issued.
- **main.space**: logical; if true, leave space for a main title (unusual for LaTeX figures!).
- **lab.space**: logical; if true, leave space for x- and y- labels (by not subtracting from *mar*).
- **paper**: character (or missing), typically "a4" or "a4r" in non-America, see *postscript*. Only if this is "special" (or missing) are your choices of width and height completely honored (and this may lead to files that cannot print on A4) with resizing.
- **lab**: integer of length 3, lab[1:2] are desired number of tick marks on x- and y-axis, see *par*(lab=).
- **mgp.lab**: three decreasing numbers determining space for axis labeling, see *par*(mgp=), the default is here smaller than usual.
- **mar**: four numbers, indicating marginal space, see *par*(mar=), the default is here smaller than usual.
- **...**: arguments passed to *ps.do*() or *pdf.do*() from *ps.latex / pdf.latex* and to *ps.options* from *ps.do/pdf.do*.

Details

*ps.latex* and *pdf.latex* have an additional LaTeX flavor, and just differ by some extra *par* settings from the *.do* siblings: E.g., after *ps.do* (...) is called, the graphical parameters c("mar", "mgp", "lab") are reset (to values that typically are better than the defaults for LaTeX figures).

Whereas the defaults for paper, width, and height differ between *pdf* and *postscript*, they are set such as to provide very similar functionality, for the functions *ps.do*() and *pdf.do*(); e.g., by default, both use a full plot on portrait-oriented page of the default paper, as per *getOption*("papersize"). *pdf.do*() sets the default paper to "special" when both width and height are specified.

Value

A list with components

- **old.par**: containing the old *par* values
- **new.par**: containing the newly set *par* values
quadrant

Give the Quadrant Number of Planar Points

Description

Determine the quadrant of planar points, i.e. in which of the four parts cut by the x- and y-axis the points lie. Zero values (i.e. points on the axes) are treated as if positive.

Usage

quadrant(x, y=NULL)

Arguments

x, y numeric vectors of the same length, or x is an x-y structure and y=NULL, see xy.coords.
QUnif

Value
numeric vector of same length as x (if that’s a vector) with values in 1:4 indicating the quadrant number of the corresponding point.

Examples
xy <- as.matrix(expand.grid(x = -7:7, y = -7:7)); rownames(xy) <- NULL
(qu <- quadrant(xy))
plot(xy, col = qu+1, main = "quadrant() number", axes = FALSE)
abline(h=0, v=0, col="gray") # the x- and y- axis
text(xy, lab = qu, col = qu+1, adj = c(1.4,0))

QUnif Quasi Random Numbers via Halton Sequences

Description
These functions provide quasi random numbers or space filling or low discrepancy sequences in the \( p \)-dimensional unit cube.

Usage
\( \text{shalton}(n, n.min = 1, \text{base} = 2, \text{leap} = 1) \)
\( \text{QUnif}(n, \text{min} = 0, \text{max} = 1, n.min = 1, p, \text{leap} = 1, \text{silent} = \text{FALSE}) \)

Arguments
- \( n.max \) maximal (sequence) number.
- \( n.min \) minimal sequence number.
- \( n \) number of \( p \)-dimensional points generated in QUnif. By default, \( n.min = 1, \text{leap} = 1 \) and the maximal sequence number is \( n.max = n.min + (n-1)*\text{leap} \).
- \( \text{base} \) integer \( \geq 2 \): The base with respect to which the Halton sequence is built.
- \( \text{min}, \text{max} \) lower and upper limits of the univariate intervals. Must be of length 1 or \( p \).
- \( p \) dimensionality of space (the unit cube) in which points are generated.
- \( \text{leap} \) integer indicating (if > 1) if the series should be leaped, i.e., only every \( \text{leap} \)th entry should be taken.
- \( \text{silent} \) logical asking to suppress the message about enlarging the prime table for large \( p \).

Value
\( \text{shalton}(n,m) \) returns a numeric vector of length \( n-m+1 \) of values in \( [0,1] \).
\( \text{QUnif}(n, \text{min}, \text{max}, n.min, p=p) \) generates \( n-n.min+1 \) \( p \)-dimensional points in \( [\text{min},\text{max}]^p \) returning a numeric matrix with \( p \) columns.
Note
For leap Kocis and Whiten recommend values of \( L = 31, 61, 149, 409 \), and particularly the \( L = 409 \) for dimensions up to 400.

Author(s)
Martin Maechler

References

Examples
```r
32*shalton(20, base=2)

stopifnot(shalton(20, base=3, leap=2) == 
  shalton(20, base=3)[1+2*(0:9)])
## ------ a 2D Visualization ------

uplot <- function(xy, axes=FALSE, xlab="", ylab="", ...) {
  plot(xy, xaxs="i", yaxs="i", xlim=0:1, ylim=0:1, xpd = FALSE, 
       axes=axes, xlab=xlab, ylab=ylab, ...)
  box(lty=2, col="gray40")
}

do4 <- function(n, ...) {
  op <- multi.fig(4, main=paste("n =", n,"": Quasi vs. (Pseudo) Random"),
                  marP=c(-2,-2,-1,0))$old.par
  on.exit(par(op))
  for(i in 1:2) {
    uplot(qunif(n, p=2), main="QUnif", ...)
    uplot(cbind(runif(n), runif(n)), main="runif", ...)
  }
}
do4(100)
do4(500)
do4(1000, cex = 0.8, col="slateblue")
do4(10000, pch = ".", col="slateblue")
do4(40000, pch = ".", col="slateblue")
```

---

**read.org.table**

**Read.table for an Emacs Org Table**

**Description**

Read an emacs “Org” table (in file or text) by `read.table()`.
Usage

read.org.table(file, header = TRUE, skip = 0, fileEncoding = "", text, ...)

Arguments

file       a file name, a file or other connection.
header     logical indicating if the org table has header line (in the usual "|"-separated org
table format).
skip       integer number of initial lines to skip.
fileEncoding see read.table
text       instead of file, a character or string (of a few lines, typically).
...         further arguments passed to read.table. You should not use encoding (but
            possibly fileEncoding!) here, as we do not call read.table on file (but on
            a textConnection).

Value

a data.frame

Note

TODO: It should be easy to extend read.org.table() to also work for some of the proposed
Markdown formats for tables. Please write to maintainer("sfsmisc") or open a github issue if
you are interested.

References


See Also

CRAN package ascii can write org tables. read.table

Examples

t1 <-
<p>|</p>
<table>
<thead>
<tr>
<th>a</th>
<th>var2</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>may</td>
<td>3.4</td>
</tr>
<tr>
<td>7</td>
<td>feb</td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
d <- read.org.table(text = t1)
d
stopifnot(dim(d) == c(2, 3),
  identical(names(d), c("a", "var2", "C")),
  d[,"a"] == c(2,7))
**Description**

Simple constructors of a constant character string from one character, notably a “blank” string of given string length.

M.M. is now ‘mentally deprecating’ bl.string in favor of using repChar() in all cases.

**Usage**

```
repChar(char, no)
bl.string(no)
```

**Arguments**

- `char` single character (or arbitrary string).
- `no` non-negative integer.

**Value**

One string, i.e., `character(1)`, for bl.string a blank string, fulfilling `n == nchar(bl.string(n))`.

**Author(s)**

Martin Maechler, early 1990's (for bl.string).

**See Also**

`paste`, `character`, `nchar`.

**Examples**

```
r <- sapply(0:8, function(n) ccat(repChar(" ", n), n))
cbind(r)

repChar("-", 4)
repChar("_", 6)
## it may make sense to a string of more than one character:
repChar("=- ", 6)

## show the very simple function definitions:
repChar
bl.string
```
Description

Rotate planar (xy) points by angle phi (in radians).

Usage

rot2(xy, phi)

Arguments

- **xy**: numeric 2-column matrix, or coercable to one.
- **phi**: numeric scalar, the angle in radians (i.e., phi=pi corresponds to 180 degrees) by which to rotate the points.

Value

A two column matrix as xy, containing the rotated points.

Author(s)

Martin Maechler, Oct.1994

Examples

```r
# Rotate three points by 60 degrees :
(xy0 <- rbind(c(1,0.5), c(1,1), c(0,1)))
(Txy <- rot2(xy0, phi = 60 * pi/180))
plot(xy0, col = 2, type = "b", asp = 1,
     xlim=c(-1,1), ylim=c(0,1.5), main = "rot2(*, pi/3) : 2d rotation by 60°")
points(Txy, col = 3, type = "b")
O <- rep(0,2); P2 <- rbind(xy0[2,], Txy[2,])
arrows(O,0,P2[,1],P2[,2], col = "dark gray")

xy0 <- .8*rbind(c(1,0), c(.5,.6), c(.7,1), c(1,1), c(.9,0), c(1,0)) - 0.2
plot(xy0, col=2, type="b", main="rot2( <polygon>, pi/4 * 1:7)", asp=1,
     xlim=c(-1,1), ylim=c(-1,1), lwd=2, axes = FALSE, xlab="", ylab="")
abline(h=0, v=0, col="thistle"); text(1.05, -.05, "x"); text(-.05,1.05, "y")
for(phi in pi/4 * 0:7)
do.call("arrows",c(list(0,0),rot2(xy0[2,], phi), length=.1, col="gray40"))
for(phi in pi/4 * 1:7)
polygon(rot2(xy0, phi = phi), col = 1+phi/(pi/4), border=2, type = "b")
```
Description

Compute generalized ‘rot13’ character translations or “rotations”

In the distant past, considered as poor man’s encryption, such rotations are way too poor nowadays and provided mainly for didactical reasons.

Usage

`rotn(ch, n = 13)`

Arguments

- `ch`: a character vector; often a string (of length 1).
- `n`: an integer in `{1, ..., 26}`; the default is particularly useful.

Details

Note that the default `n = 13` makes `rotn` into a function that is its own inverse.

Written after having searched for it and found seqinr::rotQSHI which was generalized and rendered more transparently to my eyes.

Value

A character as `ch`, but with each character (which belongs to `letters` or `LETTERS` “rotated” by `n` (positions in the alphabet).

Author(s)

Martin Maechler

See Also

- `rot2`, a completely different rotation (namely in the plane aka $R^2$).

Examples

```r
rotn(c("ABC", "a", "b", "c"), 1)
rotn(c("ABC", "a", "b", "c"), 2)
rotn(c("ABC", "a", "b", "c"), 26) # rotation by 26 does not change much

(ch <- paste("Hello", c("World!", "you too")))
rotn(ch)
rotn( rotn(ch ) ) # rotn(*, 13) is its own inverse
```
Round to Integer Keeping the Sum Fixed

Description

Given a real numbers $y_i$ with the particular property that $\sum_i y_i$ is integer, find integer numbers $x_i$ which are close to $y_i$ ($|x_i - y_i| < 1 \forall i$), and have identical “marginal” sum, $\sum(x) = \sum(y)$.

As I found later, the problem is known as “Apportionment Problem” and it is quite an old problem with several solution methods proposed historically, but only in 1982, Balinski and Young proved that there is no method that fulfills three natural desiderata.

Note that the (first) three methods currently available here were all (re?)-invented by M.Maechler, without any knowledge of the litterature. At the time of writing, I have not even checked to which (if any) of the historical methods they match.

Usage

```
roundfixS(x, method = c("offset-round", "round+fix", "1greedy"))
```

Arguments

- `x` a numeric vector which **must** sum to an integer
- `method` character string specifying the algorithm to be used.

Details

Without hindsight, it may be surprising that all three methods give identical results (in all situations and simulations considered), notably that the idea of ‘mass shifting’ employed in the iterative "1greedy" algorithm seems equivalent to the much simpler idea used in "offset-round".

I am pretty sure that these algorithms solve the $L_p$ optimization problem, $\min_x ||y - x||_p$, typically for all $p \in [1, \infty]$ simultaneously, but have not bothered to find a formal proof.

Value

a numeric vector, say $r$, of the same length as $x$, but with integer values and fulfilling $\sum(r) = \sum(x)$.

Author(s)

Martin Maechler, November 2007

References

Michel Balinski and H. Peyton Young (1982) *Fair Representation: Meeting the Ideal of One Man, One Vote*:

https://en.wikipedia.org/wiki/Apportionment_paradox

https://www.ams.org/samplings/feature-column/fcarc-apportionii3
See Also

round etc

Examples

```r
## trivial example
kk <- c(0,1,7)
stopifnot(identical(kk, roundfixS(kk))) # failed at some point

x <- c(-1.4, -1, 0.244, 0.493, 1.222, 1.222, 2, 2, 2.2, 2.444, 3.625, 3.95)
sum(x) # an integer
r <- roundfixS(x)
stopifnot(all.equal(sum(r), sum(x)))
m <- cbind(x=x, `r2i(x)` = r, resid = x - r, `|res|` = abs(x-r))
rbind(m, c(colSums(m[,1:2]), 0, sum(abs(m[,"|res|"]))))

chk <- function(y) {
cat("sum(y) =", format(S <- sum(y)),"\n")
r2 <- roundfixS(y, method="offset")
r2. <- roundfixS(y, method="round")
r2_ <- roundfixS(y, method="lg")
stopifnot(all.equal(sum(r2 ), S),
    all.equal(sum(r2.), S),
    all.equal(sum(r2_), S))
    all(r2 == r2. & & r2. == r2_) # TRUE if all give the same result
}

makeIntSum <- function(y) {
    n <- length(y)
y[n] <- ceiling(y[n]) - (sum(y[-n]) %% 1)
y
} 
set.seed(11)
y <- makeIntSum(rnorm(100))
chk(y)

## nastier example:
set.seed(7)
y <- makeIntSum(rpois(100, 10) + c(runif(75, min= 0, max=.2), runif(25, min=.5, max=.9)))

chk(y)

## Not run:
for(i in 1:1000)
    stopifnot(chk(makeIntSum(rpois(100, 10) +
                      c(runif(75, min= 0, max=.2),
                      runif(25, min=.5, max=.9))))

## End(Not run)
```
rrange

Robust Range using Boxplot 'Quartiles'

Description
Compute a robust range, i.e. the usual range() as long as there are no outliers, using the “whisker boundaries” of boxplot, i.e., boxplot.stats.

Usage
rrange(x, range=1, coef = 1.5, na.rm = TRUE)

Arguments
x numeric vector the robust range of which shall be computed.
range number for S compatibility; 1.5 * range is equivalent to coef.
coef numeric multiplication factor defining the outlier boundary, see ‘Details’ below.
na.rm logical indicating how NA values should be handled; they are simply dropped when na.rm = TRUE as by default.

Details
The robust range is really just what boxplot.stats(x, coef=coef) returns as the whisker boundaries. This is the most extreme values x[j] still inside median plus/minus coef * IQR.

Value
numeric vector c(m,M) with m ≤ M which is (not strictly) inside range(x) = c(min(x),max(x)).

Author(s)
Martin Maechler, 1990.

See Also
range, fivenum, boxplot and boxplot.stats.
A more sophisticated robust range for (strongly) asymmetric data can be derived from the skewness adjusted boxplot statistics adjboxStats which is a generalization of boxplot.stats.

Examples
stopifnot(rrange(c(1:10,1000)) == c(1,10))
seqXtend  

Sequence Covering the Range of \(X\), including \(X\)

Description

Produce a sequence of unique values (sorted increasingly), containing the initial set of values \(x\). This can be useful for setting prediction e.g. ranges in nonparametric regression.

Usage

\[
\text{seqXtend}(x, \text{length.}, \text{method} = \text{c("simple", "aim", "interpolate")}, \\
\text{from} = \text{NULL}, \text{to} = \text{NULL})
\]

Arguments

- \(x\) numeric vector.
- \text{length.} integer specifying approximately the desired \text{length()} of the result.
- \text{method} string specifying the method to be used. The default, "simple" uses \text{seq(*, length.out = length.)} where "aim" aims a bit better towards the desired final length, and "interpolate" interpolates evenly inside each interval \([x_i, x_{i+1}]\) in a way to make all the new intervals of approximately the same length.
- \text{from, to} numbers to be passed to (the default method for) \text{seq()}, defaulting to the minimal and maximal \(x\) value, respectively.

Value

numeric vector of increasing values, of approximate length \text{length.}, (unless \text{length.} < \text{length(unique(x))} in which case, the result is simply \text{sort(unique(x))}, containing the original values of \(x\).

From, \(r <- \text{seqXtend}(x, *)\), the original values are at indices \(\text{ix <- match(x, r)}\), i.e., identical\((x, r[\text{ix}])\).

Note

\text{method = "interpolate" typically gives the best results. Calling roundfixS, it also need more computational resources than the other methods.}

Author(s)

Martin Maechler

See Also

\text{seq; plotDS} can make particularly good use of seqXtend()
Examples

```r
a <- c(1, 2, 10, 12)
seqXtend(a, 12)# --> simply 1:12
seqXtend(a, 12, "interp")# ditto
seqXtend(a, 12, "aim")# really worse
stopifnot(all.equal(seqXtend(a, 12, "interp"), 1:12))
```

```
## for a "general" x, however, "aim" aims better than default
x <- c(1.2, 2.4, 4.6, 9.9)
length(print(seqXtend(x, 12)))  # 14
length(print(seqXtend(x, 12, "aim")))  # 12
length(print(seqXtend(x, 12, "int")))  # 12
```

```
## "interpolate" is really nice:
x <- seqXtend(x, 100, "interp")
plot(xt, main="seqXtend(x, 100, \"interp\")")
points(match(x, xt), x, col = 2, pch = 20)
# .... you don't even see that it's not equidistant
# whereas the cheap method shows ...
x <- seqXtend(x, 100)
plot(xt, col="blue")
points(match(x, xt), x, col = 2, pch = 20)
```

```
## with "Date" objects
Drng <- as.Date(c("2007-11-10", "2012-07-12"))
(px <- pretty(Drng, n = 16)) # say, for the main labels
## say, a finer grid, for ticks -- should be almost equidistant
n3 <- 3*length(px)
summary(as.numeric(diff(seqXtend(px, n3))))  # wildly varying
summary(as.numeric(diff(seqXtend(px, n3, "aim"))))  # (ditto)
summary(as.numeric(diff(seqXtend(px, n3, "int"))))  # around 30
```

---

### `sessionInfoX` 
**Extended Information About the Current R Session**

#### Description
Collect (and print) information about the current R session and environment, using `sessionInfoX()` and more mostly low-level and platform dependent information.

#### Usage
```
sessionInfoX(pkgs = NULL, list.libp = FALSE, extraR.env = TRUE)
```

```
## S3 method for class 'sessionInfoX'
print(x, locale = TRUE, RLIBS = TRUE, Renv = TRUE, ...)```
Arguments

pkgs an optional character vector of R packages, whose `packageDescription()`s are wanted.

list.libP a logical indicating if for all `.libPaths` entries, the files should be listed via `list.files`.

extraR.env logical indicating if all environment variables should be recorded which start with "R_" or "_R_".

x typically the result of `sessionInfoX()`.

locale logical, passed to `print.sessionInfo()` indicating if the locale information should be printed.

RLIBS logical indicating if the information about R_LIBS should be printed.

Renv logical indicating if the information about R environment variables should be printed.

... passed to `print` methods.

Value

an object of S3 class "sessionInfoX", a `list` with components (there may be more, experimental and not yet listed here):

sInfo simply the value of `sessionInfo()`.

sysInf the value of `Sys.info()`.

capabilities the value of `capabilities()`.

extSoft for R 3.2.0 and newer, the value of `extSoftVersion()`.

LAPACK for R 3.0.3 and newer, the value of `la_version()`.

pcre for R 3.1.3 and newer, the value of `pcre_config()`.

pkgDescr If pkgs was non-empty, a named `list` of `packageDescription()`s for each entry in pkgs.

libPath the value of `.libPaths()`.

RLIBS a character vector of entries from `Sys.getenv("R_LIBS")`, typically very similar to the libPaths component.

n.RLIBS simply a `normalizePath()`ed version of RLIBS.

R.env a named character vector with the “important” R environment variables R_ENVIRON, R_PROFILE, R_CHECK_ENVIRON.

xR.env if extraR.env was true, a named character vector of “all R related” environment variables, as specified in extraR.env’s description above.

Author(s)

Martin Maechler, December 2015

See Also

`sessionInfo`, `.libPaths`, `R.version`, `Sys.getenv`. 
Examples

```r
six0 <- sessionInfo()
sixN <- sessionInfo("nlme", list.libP = TRUE)
sixN # -> print() method for "sessionInfo"
   names(sixN)
   str(sixN, max = 1)# outline of lower-level structure
   str(sixN$pkgDescr) # list with one component "nlme"
```

---

**signi**

**Rounding to Significant Digits**

Description

Rounds to significant digits similarly to `signif`.

Usage

```
signi(x, digits = 6)
```

Arguments

- `x` numeric vector to be rounded.
- `digits` number of significant digits required.

Value

numeric vector “close” to `x`, i.e. by at least `digits` significant digits.

Note

This is really just `round(x, digits = trunc(log10(abs(x))))` and hence mainly of didactical use. Rather use `signif()` otherwise.

Author(s)

Martin Maechler, in prehistoric times (i.e. before 1990).

See Also

`signif`, `round`.

Examples

```r
(x1 <- seq(-2, 4, by = 0.5))
   identical(x1, signi(x1))# since 0.5 is exact in binary arithmetic
(x2 <- pi - 3 + c(-5,-1,0,.1,.2,1,10,100))
   signi(x2, 3)
```
**sourceAttach**  

Source and Attach an R source file

**Description**

Source (via `sys.source()`) and attach (`attach()`) an R source file.

**Usage**

```r
sourceAttach(file, pos=2,
             name = paste(abbreviate(gsub("", dirname(file)),
                12, method="both.sides"),
                basename(file), sep=fsep),
             keep.source =getOption("keep.source.pkgs"),
             warn.conflicts = TRUE)
```

**Arguments**

- `file` file name
- `pos` passed to `attach()`
- `name` character, with a smart default, passed to `attach()`.
- `keep.source` logical, see `sys.source()`.
- `warn.conflicts` logical, see `attach`.

**Value**

the return value of `attach()`.

**Author(s)**

Martin Maechler, 29 Jul 2011

**See Also**

`attach`, `sys.source`, `source`

**Examples**

```r
sourceAttach(system.file("test-tools-1.R", package="Matrix", mustWork=TRUE))
search() # shows the new "data base" at position 2
## look what it contains:
ls.str(pos = 2)
```
str_data

Overview on All Datasets in an R Package

Description

Provide an overview over all datasets available by data() in a (list of) given R packages.

Usage

str_data(pkgs, filterFUN, ...)

Arguments

pkgs character vector of names of R packages.
filterFUN optionally a logical function for filtering the R objects.
... potentional further arguments to be passed to str; str(utils:::str.default) gives useful list.

Value

invisibly (see invisible) a list with named components matching the pkgs argument. Each of these components is a named list with one entry per data(.) argument name. Each entry is a character vector of the names of all objects, typically only one.

The side effect is, as with str(), to print everything (via cat) to the console.

Author(s)

Martin Maechler

See Also

str.data.

Examples

str_data("cluster")

str_data("datasets", max=0, give.attr = FALSE)

## Filtering (and return value)
dfl <- str_data("datasets", filterFUN=is.data.frame)
str(df.d <- dfl$datasets)
## dim() of all those data frames:
t(sapply(unlist(df.d), function(.) dim(get(.))))

### Data sets in all attached packages but "datasets" (and stubs):
s <- search()
(Apkgs <- sub("Package:", ",", s[grep("Package:", s)]))
str_data(Apkgs[!Apkgs %in% c("datasets", "stats", "base")])
Sys.cpuinfo

Provide Information about the Linux Hardware (CPU, Memory, etc)

Description

Return information about the Linux hardware, notably the CPU (the central processor unit) and memory of the computer R is running on. This is currently only available for Linux.

These functions exist on other unix-alike platforms, but produce an error when called.

Usage

Sys.procinfo(procfile)
Sys.cpuinfo()
Sys.meminfo()
Sys.memGB(kind = "MemTotal")
Sys.MIPS()

Arguments

procfile  name of file the lines of which give the CPU info “as on Linux”
kind      a character string specifying which kind of memory is desired.

Value

The Sys.*info() functions return a "simple.list", here basically a named character vector, (where the names have been filtered through make.names(*, unique=TRUE) which is of importance for multi-processor or multi-core CPUs, such that vector can easily be indexed.

Sys.memGB() returns available memory in giga bytes [GB];
Sys.MIPS() returns a number giving an approximation of the Million instructions Per Second that the CPU processes (using “bogomips”). This is a performance measure of the basic non-numeric processing capabilities. For single-core Linux systems, often about twice the basic clock rate in “MHz” (as available by Sys.cpuinfo()["cpu.MHz"]); now, with multicore systems, the result is often around (but smaller than) 2 * #{cores} * clock.rate.

Note

These currently do rely on the Linux ‘/proc/’ file system, and may not easily be portable to non-Linux environments.

On multi-processor machines, Sys.cpuinfo() contains each field for each processor (i.e., names(Sys.cpuinfo()) has duplicated entries).

Conceivably, the bogoMIPS source code is open and available and could be built into R.

Author(s)

Martin Maechler
Sys.ps

See Also

Sys.ps, etc.

Examples

(n.cores <- parallel::detectCores())
if(substr(R.version["os"], 1, 5) == "linux") { ##-- only on Linux
  Sys.cpuinfo() # which is often ugly; this looks much better:
  length(Sys.cpu2 <- local({I <- Sys.cpuinfo(); I[ !grep("^flags", names(I)) ] }))
  ## may still be too much, notably if n.cores > 2:
  (Sys3 <- Sys.cpu2[ !grep("[0-9]+$", names(Sys.cpu2)) ])
  Sys.MIPS() ## just the 'bogomips' from above:
  Sys.MIPS() / as.numeric(Sys.cpuinfo()["cpu.MHz"])) ## -- 2 * #{cores} (no longer)

  ## Available Memory -- can be crucial:
  Sys.memGB() # default "MemTotal"
  if(Sys.memGB("MemFree") > 16)
    message("Be happy! You have more than 16 Gigabytes of free memory")
}

Sys.ps

Return Process Status (Unix 'ps') Information

Description

These functions return process id and status information, typically about the running R process.

Usage

Sys.ps(process= Sys.getpid(),
   fields = c("pid", "pcpu", "time", "vsz", "comm"),
   usefile = length(fields) > 10,
   ps.cmd  = Sys.ps.cmd(),
   verbose = getOption("verbose"),
   warn.multi = verbose || any(fields != "ALL"))

Sys.sizes(process = Sys.getpid(), ps.cmd = Sys.ps.cmd())

Arguments

  process   the process id, an integer.
  fields    character strings of "ALL", specifying which process status fields are desired.
  usefile   logical; if true, system writes to a temporary file and that is scanned subsequently.
  ps.cmd    character string, giving the “ps” command name to be used.
  verbose   logical ...
  warn.multi logical ...
Details

Use `man ps` on your respective Unix system, to see what fields are supported exactly. Unix dialects do differ here, and, SunOS-Solaris even has more than one `ps` command...

Value

Note, that `Sys.sizes()` currently returns two integers which are “common” to Solaris and Linux.

Author(s)

Martin Maechler

See Also

`Sys.info`, `Sys.getpid`, `proc.time`.

Examples

```r
(pid <- Sys.getpid()) ## process ID of current process
Sys.sizes(pid)

## The default process statistics about the running R process
try( Sys.ps() )
```

---

**TA.plot**  
*Tukey-Anscombe Plot (Residual vs. Fitted) of a Linear Model*

Description

From a linear (or glm) model fitted, produce the so-called Tukey-Anscombe plot. Useful (optional) additions include: 0-line, lowess smooth, 2sigma lines, and automatic labeling of observations.

Usage

```r
TA.plot(lm.res,
  fit= fitted(lm.res), res= residuals(lm.res, type="pearson"),
  labels= NULL, main= mk.main(), xlab = "Fitted values",
  draw.smooth= n >= 10, show.call = TRUE, show.2sigma= TRUE,
  lo.iter = NULL, lo.cex= NULL,
  par0line = list(lty = 2, col = "gray"),
  parSmooth = list(lwd = 1.5, lty = 4, col = 2),
  parSigma = list(lwd = 1.2, lty = 3, col = 4),
  verbose = FALSE,
  ...
)
```
Arguments

- `lm.res`: Result of `lm()`, `aov()`, `glm()` or a similar object.
- `fit`: fitted values; you probably want the default here.
- `res`: residuals to use. Default: **Weighted** ("Pearson") residuals if weights have been used for the model fit.
- `labels`: strings to use as plotting symbols for each point. Default(NULL): extract observations’ names or use its sequence number. Use, e.g., "*" to get simple * symbols.
- `main`: main title to plot. Default: sophisticated, resulting in something like "Tukey-Anscombe Plot of : y \sim x" constructed from lm.res $ call.
- `xlab`: x-axis label for plot.
- `draw.smooth`: logical; if TRUE, draw a lowess smoother (with automatic smoothing fraction).
- `show.call`: logical; if TRUE, write the "call"ing syntax with which the fit was done.
- `show.2sigma`: logical; if TRUE, draw horizontal lines at ±2σ where σ is mad(resid).
- `lo.iter`: positive integer, giving the number of lowess robustness iterations. The default depends on the model and is 0 for non Gaussian glm’s.
- `lo.cex`: character expansion ("cex") for lowess and other marginal texts.
- `par0line`: a list of arguments (with reasonable defaults) to be passed to `abline(.)` when drawing the x-axis, i.e., the $ y = 0 $ line.
- `parSmooth, parSigma`: each a list of arguments (with reasonable default) for drawing the smooth curve (if draw.smooth is true), or the horizontal sigma boundaries (if show.2sigma is true) respectively.
- `verbose`: logical indicating if some construction details should be reported (print()ed).
- `...`: further graphical parameters are passed to `n.plot(.)`.

Side Effects

The above mentioned plot is produced on the current graphic device.

Author(s)

Martin Maechler, Seminar fuer Statistik, ETH Zurich, Switzerland; <maechler@stat.math.ethz.ch>

See Also

- `plot.lm` which also does a QQ normal plot and more.

Examples

```r
data(stackloss)
TA.plot(lm(stack.loss ~ stack.x))

example(airquality)
summary(lmO <- lm(Ozone ~ ., data= airquality))
```
tapplySimpl

More simplification in tapply() result

description

For the case of more than two categories or indices (in INDEX), traditional \texttt{tapply(\*, simplify = TRUE)}
still returns a list when an array may seem more useful and natural. This is provided by \texttt{tapplySimpl()} if the function \texttt{FUN()} is defined such as to return a vector of the same length in all cases.

usage

\texttt{tapplySimpl(X, INDEX, FUN, ...)}

arguments

\begin{itemize}
\item \texttt{X} an atomic object, typically a vector. All these arguments are as in \texttt{tapply()} and are passed to \texttt{tapply(\*)}.
\item \texttt{INDEX} list of (typically more than one) factors, each of same length as \texttt{X}.
\item \texttt{FUN} the function to be applied. For the result to be simplifiable, \texttt{FUN()} must return a vector of always the same length.
\item \texttt{...} optional arguments to \texttt{FUN}.
\end{itemize}

value

If the above conditions are satisfied, the list returned from \texttt{r <- tapplySimpl(X, INDEX, FUN, ...)} is simplified into an array of rank $1+\#\{\text{indices}\}$, i.e., \texttt{1+length(INDEX)}; otherwise, \texttt{tapplySimpl()} returns the list \texttt{r}, i.e., the same as \texttt{tapply()}.
tkdensity

Author(s)
Martin Maechler, 14 Jun 1993 (for S-plus).

See Also
tapply(*, simplify=TRUE).

Examples

```r
## Using tapply() would give a list (with dim() of a matrix);
## here we get 3-array:

data(esoph)
with(esoph, {
    mima <- tapply(cases/ncontrols, list(agegp, alcgp), range)
    stopifnot(dim(mima) == c(2, nlevels(agegp), nlevels(alcgp)))
})
aperm(mima)
```

---

**tkdensity**

*GUI Density Estimation using Tcl/Tk*

**Description**

This is graphical user interface (GUI) to `density`, allowing for dynamic bandwidth choice and a simple kind of zooming, relying on `library(tcltk)`.

**Usage**

```r
tkdensity(y, n = 1024, log.bw = TRUE, showvalue = TRUE, 
         xlim = NULL, do.rug = size < 1000, kernels = NULL, 
         from.f = if (log.bw) -2 else 1/1000, 
         to.f = if (log.bw) +2.2 else 2, 
         col = 2)
```

**Arguments**

- `y` numeric; the data the density of which we want.
- `n` integer; the number of abscissa values for `density` evaluation (and plotting).
- `log.bw` logical; if true (default), the gui scrollbar is on a log bandwidth scale, otherwise, simple interval.
- `showvalue` logical; if true, the value of the current (log) bandwidth is shown on top of the scrollbar.
- `xlim` initial xlim for plotting, see `plot.default`.
- `do.rug` logical indicating if `rug(y)` should be added to each plot. This is too slow for really large sample sizes.
toLatex.numeric

kernels character vector of kernel names as allowable for the kernels argument of the standard density function.

from.f, to.f numeric giving the left and right limit of the bandwidth scrollbar.

col color to be used for the density curve.

Details

library(tcltk) must be working, i.e., Tcl/Tk must have been installed on your platform, and must have been visible during R’s configuration and/or installation.

You can not only choose the bandwidth (the most important parameter), but also the kernel, and you can zoom in and out (in x-range only).

Value

none.

(How could this be done? tcltk widgets run as separate processes!)

Author(s)

Martin Maechler, building on demo(tkdensity).

Examples

if (dev.interactive(TRUE)) ## does really not make sense otherwise
if(try(require("tcltk"))) { ## sometimes (rarely) there, but broken

  data(faithful)
  tkdensity(faithful $ eruptions)

  set.seed(7)
  if(require("nor1mix"))
    tkdensity(rnorMix(1000, MW.nn9), kernels = c("gaussian", "epanechnikov"))
}


---

toLatex.numeric \hspace{1cm} \textit{\LaTeX \ or \ Sweave \ friendly \ Formatting \ of \ Numbers}

Description

Formats real numbers, possibly in scientific notation, with a given number of digits after the decimal point. Output can be used in \LaTeX\ math mode, e.g., for printing numbers in a table, where each number has to be printed with the same number of digits after the decimal point, even if the last digits are zeros.

Usage

## S3 method for class 'numeric'
toLatex(object, digits = format.info(object)[2],
    scientific = format.info(object)[3] > 0, times = "\cdot", ...)

Arguments

- **object**: a numeric vector.
- **digits**: number of digits after the decimal point (for the mantissa if scientific). The default behaves the same as R's `format()`.  
- **scientific**: logical indicating if scientific notation should be used. The default behaves the same as R's `format()`. 
- **times**: character string indicating the LaTeX symbol to be used for the ‘times’ sign. 
- **...**: unused; for compatibility with `toLatex`.

Value

A character vector of the same length as `object`, containing the formatted numbers.

Note

We use `digits` for `round`, i.e., round after the decimal point on purpose, rather than significant digit rounding as used by `print()` or `format()`.

Author(s)

Alain Hauser

See Also

- `pretty10exp` which gives expressions similar to our `scientific=TRUE`. `toLatex` with other methods.

Examples

```r
xx <- pi * 10^(-9:9)

format(xx)
formatC(xx)

toLatex(xx)   #-> scientific = TRUE is chosen
toLatex(xx, scientific=FALSE)

sapply(xx, toLatex)
sapply(xx, toLatex, digits = 2)
```
u.assign0

'Portable' assign / get functions (R / S-plus) for 'Frame 0'

Description

R does not have S' concept of frame = 0, aka 'session frame'. These two function were an attempt to provide a portable way for working with frame 0, particularly when porting code from S.

They have been deprecated since August 2013.

Usage

u.assign0(x, value, immediate = FALSE)
u.get0(x)

Arguments

x character string giving the name of the object.
value any R object which is to be assigned.
immediate logical, for S compatibility. No use in R.

Note

Really don't use these anymore...

Author(s)

Martin Maechler

See Also

g.get, assign.

u.boxplot.x

Utility Returning x-Coordinates of Boxplot

Description

Return the x-coordinates in an 'n-way' side-by-side boxplot. This is an auxiliary function and exists mainly for backcompatibility with S-plus.

Usage

u.boxplot.x(n, j = 1:n, fullrange = 100)
Arguments

- **n**: number of boxplots.
- **j**: indices of boxplots.
- **fullrange**: x-coords as 'uniform' in \([0, \text{fullrange}]\); \((f=100, \text{corresponds to Splus 3.x (x = 1,2)})\).

Value

A numeric vector of length \(n\), with values inside \((0, M)\) where \(M = \text{fullrange}\).

Author(s)

Martin Maechler

See Also

boxplot.

Examples

```r
u.boxplot.x(7) # == 8.93 22.62 36.3 ... 91.07
```

---

**u.date**

**Return Date[-Time] String in 'European' Format**

Description

Return one string of the form "day/month/year", plus "hour:minutes", optionally.

Usage

```r
u.date(short=FALSE)
```

Arguments

- **short**: logical; if TRUE, no time is given.

Value

String with current date (and time).

Author(s)

Martin Maechler, ca. 1992

See Also

u.Datumvonheute.
**Examples**

```r
u.date()
u.date(short = TRUE)
```

---

**u.datumdecode**  
*Convert “Numeric” Dates*

**Description**

Daten der Form 8710230920 aufspalten in Jahr, Monat, Tag, Std, Min

**Usage**

```r
u.datumdecode(d, YMDHMnames = c("Jahr", "Monat", "Tag", "Std", "Min"))
```

**Arguments**

- `d` numeric dates in the form YYMMDDHHMM.
- `YMDHMnames` (column) names to be used for the result.

**Value**

a numeric matrix (or vector) with 5 columns containing the year, month, etc.

**Note**

MM: This is a wrong concept, and also suffers from the “millenium bug” (by using only 2 digits for the year).

**Author(s)**

?? (someone at SfS ETH)

**See Also**

R’s proper date-time coding: `DateTimeClasses`; `u.date` etc.

**Examples**

```r
u.datumdecode(8710230920)
##     Jahr Monat   Tag   Std  Min
## 87 10 23   9   20

u.datumdecode(c(8710230900, 9710230920, 0210230920))
##     Jahr Monat   Tag   Std  Min
## [1,] 87 10 23   9   00
## [2,] 97 10 23   9   20
## [3,] 2  10 23   9   20
```
u.Datumvonheute

*Datum und Uhrzeit (auf deutsch)*

**Description**

Return current date and time as a string, possibly including day of the week in *German*.

**Usage**

```
u.Datumvonheute(W.tag=2, Zeit=FALSE)
```

- C.Monatsname
- C.Wochentag
- C.Wochentagkurz
- C.weekday

**Arguments**

- `W.tag` logical or integer specifying you want weekday (‘Wochentag’). 0 or FALSE gives no, 1 or TRUE gives a short and 2 the long version of the day of the week.
- `Zeit` logical or integer specifying if time (“Zeit”) is desired. 0 or FALSE gives no, 1 or TRUE gives a hours only and 2 hours and minutes.

**Value**

A string with the current date/time, in the form specified by the arguments.

The C.* are character vector “constants”, the German ones actually used by u.Datumvonheute.

**Author(s)**

Caterina Savi, Martin Maechler

**See Also**

- [u.date](#) for a similar English version, and [p.datum](#) which plots. For English month names, etc
  - [month.name](#)

**Examples**

```
u.Datumvonheute()
u.Datumvonheute(W.tag=1, Zeit=TRUE)
u.Datumvonheute(W.tag= FALSE, Zeit=2)
```
**u.log**  
*(Anti)Symmetric Log High-Transform*

**Description**

Compute $\log()$ only for high values and keep low ones – antisymmetrically such that $u.log(x)$ is (once) continuously differentiable, it computes

$$f(x) = x \text{ for } |x| \leq c \text{ and } \text{sign}(x)c \cdot (1 + \log(|x|/c)) \text{ for } |x| \geq c.$$  

**Usage**

```r
u.log(x, c = 1)
```

**Arguments**

- `x` numeric vector to be transformed.
- `c` scalar, $> 0$

**Value**

numeric vector of same length as `x`.

**Author(s)**

Martin Maechler, 24 Jan 1995

**Examples**

```r
curve(u.log, -3, 10); abline(h=0, v=0, col = "gray20", lty = 3)
curve(1 + log(x), .01, add = TRUE, col = "brown") # simple log
curve(u.log(x, 2), add = TRUE, col=2)
curve(u.log(x, c= 0.4), add = TRUE, col=4)
```

---

**u.sys**  
'*Portable' System function (R/S-plus)*

**Description**

`u.sys()` is a convenient wrapper (of `system()`) to call to the underlying operating system. The main purpose has been to provide a function with identical UI both in S-PLUS and R. MM thinks you shouldn’t use this anymore, usually.

`Sys.ps.cmd()` returns the ‘ps’ (‘process status’) OS command name (as `character` string), and is typically usable on unix alikes only.
Usage

\texttt{u.sys(..., intern = TRUE)}
\texttt{Sys.ps.cmd()}

Arguments

\texttt{...} \hspace{1cm} any number of strings – which will be \texttt{paste()}d together and passed to \texttt{system}.
\texttt{intern} \hspace{1cm} logical – note that the default is \textit{reversed} from the one in \texttt{system()}.

Author(s)

Martin Maechler

See Also

\texttt{system}, \texttt{really!}; on non-Windows, \texttt{Sys.ps()} which makes use of \texttt{Sys.ps.cmd()}.

Examples

\texttt{u.sys # shows how simply the function is defined}
\texttt{## Not run:}
\texttt{function (...; intern = TRUE)}
\texttt{system(paste(...; sep = ""), intern = intern)}
\texttt{## End(Not run)}

\texttt{# All *running* processes of user [sometimes only R]:}
\texttt{try ( u.sys(Sys.ps.cmd(), "ur") )}

---

\textbf{unif} \hspace{1cm} \textit{Nice Uniform Points in Interval}

Description

Give regularly spaced points on interval $[-c, c]$ with mean 0 (exactly) and variance about 1 (very close for \textbf{even} $n$ and larger \texttt{round.dig}). Note that $c$ depends on $n$.

Usage

\texttt{unif(n, round.dig = 1 + trunc(log10(n)))}

Arguments

\texttt{n} \hspace{1cm} positive integer specifying the number of points desired.
\texttt{round.dig} \hspace{1cm} integer indicating to how many digits the result is rounded.
Value

numeric vector of length n, symmetric around 0, hence with exact mean 0, and variance approximately 1.

Note

It relies on the fact that \( Var(1, 2, ..., n) = n(n + 1)/12 \).

Author(s)

Martin Maechler, ca 1990

See Also

runif for producing uniform random numbers.

Examples

(u <- unif(8))
  var(u)
  
(u. <- unif(8, 12))# more digits in result, hence precision for Var :
  var(u.)

Description

A Reversible Version of unique()

Usage

uniqueL(x, isuniq = !duplicated(x), need.sort = is.unsorted(x))

Arguments

x numeric vector, of length n, say.

isuniq logical vector of the same length as x. For the reversion to work this should select at least all unique values of x.

need.sort logical indicating if x is not yet sorted. Note that this argument exists only for speedup possibility when it is known, and that it must be set correctly.
Value

list of two components,

ix integer vector of indices
xU vector of values from x

such that both x[isuniq] == xU and xU[ix] == x.

Author(s)

Martin Maechler

See Also

duplicated from the sfsmisc package in addition to the standard unique and duplicated.

Examples

x0 <- c(1:3,2:7,8:4)
str(r0 <- unique(x0))
with(r0, xU[ix]) ## == x0 !

vcat Paste Utilities – Concatenate Strings

Description

Concatenate vector elements or anything using paste(*, collapse = ). These are simple short abbreviations I have been using in my own codes in many places.

Usage

vcat(vec, sep = " ")
ccat(...)

Arguments

vec, ... any vector and other arguments to be pasted to together.
sep the separator to use, see the Details section.

Details

The functions are really just defined as
vcat := function(vec, sep = " ") paste(vec, collapse = sep)
ccat := function(...) paste(..., collapse = "", sep = "")
Value

a character string (of length 1) with the concatenated arguments.

Author(s)

Martin Maechler, early 1990’s.

See Also

paste, as.character, format, cat() is really for printing.

Examples

ch <- "is"
cat("This ", ch, ", it: ", 100, "%")
vv <- c(1, pi, 20.4)
vcat(vv)
vcat(vv, sep = ", ")

Description

The main motivation for this function has been the easy construction of a “full GAM formula” from something as simple as Y ~ ..
The potential use is slightly more general.

Usage

wrapFormula(f, data, wrapString = "s(*)")

Arguments

f the initial formula; typically something like Y ~ ..
data data.frame to which the formula applies; see, formula or also gam or lm.
wrapString character string, containing "*", specifying the wrapping expression to use.

Value

a formula very similar to f; just replacing each additive term by its wrapped version.

Note

There are limits for this to work correctly; notably the right hand side of the formula f should not be nested or otherwise complicated, rather typically just .. as in the examples.
Author(s)


See Also

`formula`, `gam` from package `mgcv` (or also from package `gam`).

Examples

```r
myF <- wrapFormula(Fertility ~ ., data = swiss)
myF # Fertility ~ s(Agriculture) + s(....) + ...

if(require("mgcv")) {
  m1 <- gam(myF, data = swiss)
  print( summary(m1) )
  plot(m1, pages = 1); title(format(m1$call), line= 2.5)
}

## other wrappers:
wrapFormula(Fertility ~ ., data = swiss, wrap = "lo(*)")
wrapFormula(Fertility ~ ., data = swiss, wrap = "poly(*, 4)")
```

xy.grid

 Produce regular grid matrix.

Description

Produce the grid used by `persp`, `contour`, etc, as an $N \times 2$ matrix. This is really outdated by `expand.grid()` nowadays.

Usage

```r
xy.grid(x, y)
```

Arguments

- `x, y` any vectors of same mode.

Value

A 2-column matrix of “points” for each combination of x and y, i.e. with `length(x) \times length(y)` rows.

Author(s)

See Also

`expand.grid` which didn’t exist when `xy.grid` was first devised.

Examples

```r
plot(xy.grid(1:7, 10*(0:4)))

x <- 1:3; y <- 10*(0:4)
xyg <- xy.grid(x,y)

## Compare with expand.grid() :
m2 <- as.matrix(expand.grid(y,x)[, 2:1])
dimnames(m2) <- NULL
stopifnot(identical(xyg, m2))
```

---

### xy.unique.x

**Uniqify (X,Y) Values using Weights**

#### Description

Given smoother data \((x_i, y_i)\) and maybe weights \(w_i\), with multiple \(x_i\), use the unique \(x\) values, replacing the \(y\)'s by their (weighted) mean and updating the weights accordingly.

#### Usage

```r
xy.unique.x(x, y, w, fun.mean = mean, ...)
```

#### Arguments

- **x, y** numeric vectors of same length. Alternatively, \(x\) can be a \('xy\' like structure, see `xy.coords`.  
- **w** numeric vector of non-negative weights – or missing which corresponds to all weights equal.  
- **fun.mean** the mean function to use.  
- **...** optional arguments all passed to `unique`.

#### Value

Numeric matrix with three columns, named \(x\), \(y\) and \(w\) with unique \(x\) values and corresponding \(y\) and weights \(w\).

#### Author(s)

Martin Maechler, 8 Mar 1993.

#### See Also

e.g., `smooth.spline` uses something like this internally.
Examples

```r
## simple example:
x <- c(1,1,2,4,3,1)
y <- 1:6
rbind(x, y)
xy.unique.x(x, y)
#  x  y  w
# 1  1  3 3
# 2  2  3 1
# 3  4  4 1
# 4  3  5 1
xy.unique.x(x, y, fromLast = TRUE)
```
Index

*Topic algebra
  nearcor, 51
  posdefify, 69
*Topic aplot
  eaxis, 22
  linesHyperb.lm, 40
  p.arrows, 54
  p.hboxx, 56
*Topic arithmetic
  primes, 74
*Topic arith
  digitsBase, 19
  inv.seq, 35
  is.whole, 36
  iterate.lin.recursion, 37
  nr.sign.chg, 53
  polyn.eval, 68
  roundfixs, 87
  sign1, 93
  u.log, 108
  unif, 109
*Topic array
  col01scale, 8
  diagX, 18
  empty.dimnames, 27
  mpl, 46
  nearcor, 51
  posdefify, 69
  xy.grid, 113
*Topic category
  tapplySimpl, 100
*Topic character
  repChar, 84
*Topic classif
  diagDA, 16
*Topic datagen
  QUnif, 81
*Topic datasets
  potatoes, 71

  str_data, 95
*Topic debugging
  prt.DEBUG, 76
*Topic device
  cairoSwd, 6
  ps.end, 77
  ps.latex, 78
*Topic distribution
  KSD, 38
*Topic documentation
  Deprecated, 16
  str_data, 95
*Topic dplot
  aXExpr, 5
  p.scales, 61
  pretty10exp, 72
  u.boxplot.x, 104
*Topic dynamic
  loessDemo, 41
  tkdensity, 101
*Topic environment
  u.assign0, 104
  u.sys, 108
*Topic file
  read.org.table, 82
  sourceAttach, 94
*Topic hplot
  compressid2way, 9
  cum.Vert.funkt, 11
  ecdf.ksCI, 25
  errbar, 28
  histBxp, 32
  loessDemo, 41
  mpl, 46
  mult.fig, 47
  n.plot, 50
  p.datum, 54
  p.dnorm, 55
  p.profileTraces, 57

116
INDEX

p.res.2fact, 58
p.res.2x, 59
p.tachoPlot, 61
p.ts, 63
plotDS, 65
plotStep, 67
tkdensity, 101
* Topic htest
  f.robftest, 29
* Topic interface
  mat2tex, 44
* Topic iplot
  ellipsePoints, 26
* Topic iteration
  tapplySimpl, 100
* Topic loess
  loessDemo, 41
* Topic manip
  AsciiToInt, 3
  Duplicated, 21
  last, 39
  lseq, 43
  rot2, 85
  rotn, 86
  roundfixS, 87
  seqXtend, 90
* Topic math
  factorize, 30
  integrate.xy, 34
  primes, 74
  QUnif, 81
  rot2, 85
* Topic misc
  sessionInfoX, 91
  tolatex.numeric, 102
* Topic models
  diagDA, 16
  TA.plot, 98
  wrapFormula, 112
* Topic multivariate
  QUnif, 81
* Topic naive Bayes classifier
  diagDA, 16
* Topic nonlinear
  p.profileTraces, 57
* Topic nonparametric
  plotStep, 67
* Topic print
  empty.dimnames, 27
  vcat, 111
* Topic programming
  missingCh, 45
* Topic regression
  hatMat, 31
  linesHyperb.lm, 40
  p.res.2fact, 58
  p.res.2x, 59
  TA.plot, 98
* Topic robust
  f.robftest, 29
  rrange, 89
* Topic smooth
  D1D2, 12
  D2ss, 14
  hatMat, 31
* Topic ts
  p.ts, 63
* Topic univar
  rrange, 89
* Topic utilities
  diagX, 18
  digitsBase, 19
  ellipsePoints, 26
  empty.dimnames, 27
  integrate.xy, 34
  inv.seq, 35
  mat2tex, 44
  n.code, 49
  p.dnorm, 55
  paste.vec, 64
  printTable2, 75
  quadrant, 80
  read.org.table, 82
  seqXtend, 90
  sourceAttach, 94
  str_data, 95
  Sys.cpuinfo, 96
  Sys.ps, 97
  u.assign0, 104
  u.boxplot.x, 104
  u.date, 105
  u.datumdecode, 106
  u.Datumvonheute, 107
  u.sys, 108
  unif, 109
  uniqueL, 110
INDEX

formula, 59, 60, 112, 113  
ftable, 76  
function, 95, 114

gam, 112, 113  
get, 104  
getOption, 79  
glm, 99

hatMat, 31  
head, 8  
hetcor, 51, 52  
hist, 33, 34  
histBxp, 32, 56

ichar (AsciToInt), 3  
integer, 3, 20, 49, 74  
integrate, 35  
integrate.xy, 34  
interaction.plot, 10  
inv.seq, 35  
invisible, 95  
is.integer, 36  
is.whole, 36  
is.prime, 75  
iternate.lin.recursion, 37

Ksd, 25, 38

La_version, 92  
last, 39  
lalex, 44  
layout, 48  
lda, 17  
length, 4, 90  
LETTERS, 86  
letters, 86  
lines, 40  
linesHyperb.lm, 40  
list, 4, 19, 23, 30, 48, 50, 51, 92, 95  
list.files, 92  
lm, 40, 60, 99, 112  
loess, 41, 42  
loessDemo, 41  
logical, 22, 45, 74, 95  
lsq, 43

maintainer, 83  
make.names, 96  
matrix2table (printTable2), 75  
mat2tex, 44  
match, 21  
matplot, 46, 47  
matrix, 20  
max, 74  
missing, 45, 46  
missingCh, 45  
month.name, 107  
mpl, 46  
mtext, 48, 55  
mult.fig, 47, 64

n.code, 49  
n.plot, 50, 99  
n2mfrow, 47  
NA, 89  
aiveBayes, 17  
names, 4  
nchar, 84  
nextcor, 51  
nextPD, 51, 52, 70  
nextprime, 75  
nls, 57  
normalizePath, 92  
nr.sign.chg, 53  
numeric, 43

options, 44, 76  
p.arrows, 54  
p.datum, 11, 54, 107  
p.dchisq (p.dnorm), 55  
p.dgamma (p.dnorm), 55  
p.dnorm, 55  
p.hboxp, 56  
p.m (mpl), 46  
p.profileTraces, 57  
p.res.2fact, 58, 60  
p.res.2x, 58, 59  
p.scales, 61  
p.tachoplot, 61  
p.ts, 63  
packageDescription, 92  
par, 23, 28, 33, 47, 48, 50, 57, 61, 62, 64–67, 79  
paste, 84, 109, 111, 112  
paste.vec, 64  
pcre_config, 92  
pdf, 6, 7, 77–80
pdf.do, 78, 79
pdf.do (ps.latex), 78
pdf.end, 78
pdf.end (ps.end), 77
pdf.latex, 77
pdf.latex (ps.latex), 78
persp, 113
plot, 50, 62, 64, 66–68
plot.default, 28, 42, 50, 58, 60, 62, 63, 65, 101
plot.ecdf, 11, 12, 68
plot.lm, 58, 60, 99
plot.mts, 47
plot.stepfun, 25, 26, 68
plot.ts, 63, 64
plotmath, 65, 90
plotStep, 11, 12, 67
pmax, 16
pmax.sa (Deprecated), 16
pmin, 16
pmin.sa (Deprecated), 16
points, 42
polygon, 42
polyline, 68
posdefify, 51, 52, 69
postscript, 77–80
potatoes, 71
predict, 17
predict.dDA (diagDA), 16
predict.lm, 41
predict.polynomial, 69
pretty1exp, 5, 6, 22, 23, 72, 103
primes, 30, 31, 74
print, 17, 19, 92, 99, 103
print.baseInt (digitsBase), 19
print.dDA (diagDA), 16
print.default, 76
print.margin2table (printTable2), 75
print.sessionInfo, 92
print.sessionInfoX (sessionInfoX), 91
printTable2, 75
proc.time, 98
profile, 57
profile.nls, 57
prt.DEBUG, 76
ps.do, 77–79
ps.do (ps.latex), 78
ps.end, 77, 80
ps.latex, 78
ps.options, 79
qda, 17
quadrant, 80
QUnif, 81
R.version, 92
range, 89
read.org.table, 82
read.table, 82, 83
repchar, 84
rle, 35
rlm, 29, 30
rot2, 85, 86
rotn, 86
round, 88, 93, 103
roundfixS, 87, 90
rrange, 89
rug, 34, 63, 101
runif, 110
RweaveLatex, 7
scale, 9
scan, 97
scat1d, 34
segments, 54, 60, 68
seq, 38, 43, 90
seqXtend, 65, 66, 90
sessionInfo, 91, 92
sessionInfoX, 91
shHalton (QUnif), 81
signi, 93
signif, 73, 93, 103
smooth.spline, 12–15, 32, 114
source, 94
sourceAttach, 94
split, 21
stepfun, 26
str, 95
str_data, 95
strcodes (AsciiToInt), 3
summary, 29
summary.aov, 30
summary.rlm, 29
Sweave, 7
symbols, 62
Sys.cpiinfo, 96
Sys.getenv, 92
Sys.getlocale, 4
Sys.getpid, 98
Sys.info, 92, 98
Sys.memGB(Sys.cpuinfo), 96
Sys.meminfo(Sys.cpuinfo), 96
Sys.MIPS(Sys.cpuinfo), 96
Sys.procinfo(Sys.cpuinfo), 96
Sys.ps, 97, 97
Sys.ps.cmd(u.sys), 108
Sys.setlocale, 4
Sys.sizes(Sys.ps), 97
sys.source, 94
system, 97, 109

TA.plot, 50, 58, 60, 98
table, 76
tail, 8, 40
tapply, 21, 100, 101
tapplySimpl, 100
text, 50
textConnection, 83
title, 10, 25, 60
tkdensity, 101
toLatex, 103
toLatexnumeric, 73, 102
TRUE, 76
turnogram, 40

u.assign0, 104
u.boxplot.x, 58, 104
u.date, 55, 105, 106, 107
u.datumdecode, 106
u.Datumvonheute, 55, 105, 107
u.get0(u.assign0), 104
u.log, 108
u.sys, 108
unif, 109
unique, 90, 110, 111, 114
uniqueL, 21, 110
unname, 28

vcat, 111

wrapFormula, 112

x11, 77
xy.coords, 50, 65, 80, 114
xy.grid, 113
xy.unique.x, 114