

Package ‘sfsmisc’

March 19, 2012

Version 1.0-20

Date 2012-03-19

Author Martin Maechler and many others.

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

Title Utilities from Seminar fuer Statistik ETH Zurich

Description Useful utilities [‘goodies’] from Seminar fuer Statistik
ETH Zurich, quite a few related to graphics; many ported from S-plus times.

Depends R (>= 2.10.0), stats, methods, utils

Suggests datasets, tcltk, cluster, MASS, Matrix, nlme, lokern

Enhances mgcv, rpart, nor1mix, polycor, sm

Encoding latin1

License GPL (>= 2)

Repository CRAN

Date/Publication 2012-03-19 11:12:41

R topics documented:

AsciiToInt	3
axTexpr	5
capture.and.write	6
col01scale	7
compresid2way	8
cum.Vert.funkt	9
D1D2	10
D2ss	12
Deprecated	14
diagDA	15

diagX	17
digitsBase	17
Duplicated	19
eaxis	20
ecdf.ksCI	22
ellipsePoints	23
empty.dimnames	24
errbar	25
f.robftest	26
factorize	27
hatMat	28
hist.bxp	29
integrate.xy	31
inv.seq	32
iterate.lin.recursion	33
KSd	34
last	35
linesHyperb.lm	36
lseq	37
mat2tex	38
mpl	39
mult.fig	40
n.code	41
n.plot	42
nearcor	43
nr.sign.chg	46
p.arrows	46
p.datum	47
p.dnorm	48
p.hboxp	49
p.profileTraces	50
p.res.2fact	51
p.res.2x	52
p.scales	53
p.tachoPlot	54
p.ts	55
paste.vec	57
plotDS	57
plotStep	59
pmax.sa	60
polyn.eval	61
posdefify	62
potatoes	64
pretty10exp	65
primes	66
printTable2	67
prt.DEBUG	68
ps.end	69

ps.latex	70
quadrant	72
QUnif	73
repChar	74
rot2	75
roundfixS	76
rrange	78
seqXtend	79
signi	80
sourceAttach	81
str_data	82
Sys.cpuinfo	83
Sys.ps	85
TA.plot	86
tapplySimpl	88
tkdensity	89
u.assign0	90
u.boxplot.x	91
u.date	91
u.datumdecode	92
u.Datumvonheute	93
u.log	94
u.sys	95
unif	95
uniqueL	96
vcat	97
wrapFormula	98
xy.grid	99
xy.unique.x	100

Index**102**

AsciiToInt

*Character to and from Integer Codes Conversion***Description**

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 `AsciiToint`, producing (one byte) characters from integer codes.

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(AsciiToInt(LETTERS), 65:90))
## Not run:
## may only work in ISO-latin1 locale (not in UTF-8):
strcodes(c(a= "ABC", ch="1234", place = "Zürich"))
## gives
$a
[1] 65 66 67

$ch
[1] 49 50 51 52

$place
[1] 90 252 114 105 99 104

## End(Not run)

```

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 E<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](#), calling [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](#); [axis](#), [axTicks](#).

Examples

```
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= .1+ c(5,5,4,1))
plot(x,y, axes= FALSE, frame=TRUE)
ax <- axTicks(1); axis(1, at=ax, label= axTexpr(1, ax))
## horizontal labels on y-axis:
ay <- axTicks(2); axis(2, at=ay, label= axTexpr(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= axExpr(1, ax))

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

exp <- par("xexp") #-> powers of 10 *inside* 'usr'
exp[3] <- 1 # such that only 10^1 are labeled
ax <- axTicks(1, exp = exp)
xu <- 10 ^ par("usr")[1:2]
e10 <- c(-1,1) + round(log10(exp[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 = axExpr(1, ax, drop.1=TRUE)) # 'default'
axis(1, at = v, label = FALSE, tcl = 2/3 * par("tcl"))

```

capture.and.write

Capture output and Write / Print First and Last Parts

Description

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

Usage

```
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

Value

return value of [writeLines](#)

Author(s)

Martin Maechler

col01scale

Matrix Scaling Utilities

Description

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

```
## 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

```
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
coef	coefficients: Intercept and effects of the factors

Author(s)

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

References

F. Mosteller and J. W. Tukey (1977) *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley, Reading, Mass., p. 176.

John W. Tukey (1977) *Exploratory Data Analysis*. Addison-Wesley, Reading, Mass., p. 381.

See Also

[interaction.plot](#)

Examples

```
## From Venables and Ripley (2002) p.165.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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"))

## Fisher's 1926 data on potatoe yield
data(potatoes)
pot.aov <- aov(yield ~ nitrogen+potash+pos, data=potatoes)
compresid2way(pot.aov, pch=as.character(potatoes$pos))

compresid2way(yield~nitrogen+potash, data=subset(potatoes, pos == 2))

## 2 x 3 design :
data(warpbreaks)
summary(fm1 <- aov(breaks ~ wool + tension, data = warpbreaks))
compresid2way(fm1)
```

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

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

Arguments

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

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)
```

Description

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

$$Y_i = f(x_i) + E_i, \quad i = 1, \dots, n,$$

Usage

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

Arguments

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

<code>x</code>	the abscissae values at which the derivative(s) are evaluated.
<code>D1</code>	if <code>deriv</code> contains 1, estimated values of $f'(x_i)$ where x_i are the values from <code>xout</code> .
<code>D2</code>	if <code>deriv</code> contains 2, estimated values of $f''(x_i)$.
<code>spar</code>	the smoothing parameter used in the (final) <code>smooth.spline</code> call.
<code>df</code>	the equivalent degrees of freedom in that <code>smooth.spline</code> 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

```
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")])
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'() : " * frac(d,dx) * sin(x) == cos(x)))
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(offs))), lwd=1,
      col = 1:(1+length(offs)), cex = 0.8, bg = NA)
par(op)

```

D2ss

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

Description

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

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

D2ss first uses `smooth.spline` for the first derivative $f'()$ 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

```
D1tr(y, x = 1)
```

```
D1ss(x, y, xout = x, spar.offset = 0.1384, spl.spar=NULL)
```

```
D2ss(x, y, xout = x, spar.offset = 0.1384, spl.spar=NULL)
```

Arguments

<code>x,y</code>	numeric vectors of same length, supposedly from a model $y \sim f(x)$. For <code>D1tr()</code> , <code>x</code> can have length one and then gets the meaning of $h = \Delta x$.
<code>xout</code>	abscissa values at which to evaluate the derivatives.
<code>spar.offset</code>	numeric fudge added to the smoothing parameter(s), see <code>spl.par</code> below. Note that the current default is there for historical reasons only, and we often would recommend to use <code>spar.offset = 0</code> instead.
<code>spl.spar</code>	direct smoothing parameter(s) for <code>smooth.spline</code> . If it is <code>NULL</code> (as per default), the smoothing parameter used will be <code>spar.offset + sp\$spar</code> , where <code>sp\$spar</code> is the GCV estimated smoothing parameter for <i>both</i> 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. `spar.offset` is really just a *fudge* offset added to the smoothing parameters. Note that in R's implementation of `smooth.spline`, `spar` is really on the log λ scale.

Value

`D1tr()` and `D1ss()` return a numeric vector of the length of `y` or `xout`, respectively.

`D2ss()` returns a list with components

<code>x</code>	the abscissae values (= <code>xout</code>) at which the derivative(s) are evaluated.
<code>y</code>	estimated values of $f''(x_i)$.
<code>spl.spar</code>	numeric vector of length 2, contain the <code>spar</code> arguments to the two <code>smooth.spline</code> calls.
<code>spar.offset</code>	as specified on input (maybe <code>rep()</code> eated to length 2).

Author(s)

Martin Maechler, in 1992 (for S).

See Also

[D1D2](#) which directly uses the 2nd derivative of the smoothing spline; [smooth.spline](#).

Examples

```
## 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")])
xx <- seq(0,10, len=201)
plot(xx, -sin(xx), 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(off 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)

```

 Deprecated

 Deprecated 'sfsmisc' Functions

Description

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

```
p.pllines(x,y,group,lty=c(1,3,2,4),...)
```

```
list2mat(x, check = TRUE)
```

Arguments

<code>x,y</code>	panel function (or plot or general) arguments: numeric vectors of same length.
<code>group</code>	grouping vector of same length as <code>x</code> or <code>y</code> .
<code>lty</code>	line types to use for the groups (recycled).
<code>...</code>	further arguments passed to (plot) methods.
<code>check</code>	logical specifying if it should be checked that list components have same length.

Details

`p.pllines` is deprecated because basic R graphics (but not S-PLUS) provide its functionality directly: Use `plot(x,y, lty = group, type = 'l', ...)`.

`list2mat(x)` is usually the same as `sapply(x, c)` (where the latter does not construct column names where `x` has no names).

`rnls(formula, data, ...)` has been available, improved in package **robustbase** for almost a year and been replaced by a stub here. All `*.rnls` methods have been completely removed.

`diagDA`*Diagonal Discriminant Analysis*

Description

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

Usage

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

diagDA(ls, cll, ts, pool = TRUE)
```

Arguments

<code>x, ls</code>	learning set data matrix, with rows corresponding to cases (e.g., mRNA samples) and columns to predictor variables (e.g., genes).
<code>cll</code>	class labels for learning set, must be consecutive integers.
<code>object</code>	object of class <code>dDA</code> .
<code>ts, newdata</code>	test set (prediction) data matrix, with rows corresponding to cases and columns to predictor variables.
<code>pool</code>	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 (<code>pool= FALSE</code>), the covariance matrices may vary across classes and the discriminant rule is quadratic in the data.
<code>...</code>	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 `diagDA()`:

`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

S. Dudoit, J. Fridlyand, and T. P. Speed. (2000) Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. (Statistics, UC Berkeley, June 2000, Tech Report \#576)

See Also

[lda](#) and [qda](#) from the MASS package.

Examples

```
## two artificial examples by Andreas Greutert:
d1 <- data.frame(x = c(1, 5, 5, 5, 10, 25, 25, 25, 25, 29),
                 y = c(4, 1, 2, 4, 4, 4, 6:8, 7))

n.plot(d1)
library(cluster)
(c11P <- pam(d1,k=4)$cluster) # 4 surprising clusters
with(d1, points(x+0.5, y, col = c11P, pch =c11P))

i1 <- c(1,3,5,6)
tr1 <- d1[-i1,]
c11. <- c(1,2,1,2,1,3)
c11 <- c(2,2,1,1,1,3)
plot(tr1, cex=2, col = c11, pch = 20+c11)
(dd.<- diagDA(tr1, c11., ts = d1[ i1,]))# ok
(dd <- diagDA(tr1, c11 , ts = d1[ i1,]))# ok, too!
points(d1[ i1,], pch = 10, cex=3, col = dd)

## use new fit + predict instead :
(r1 <- dDA(tr1, c11))
(r1.<- dDA(tr1, c11.))
stopifnot(dd == predict(r1, new = d1[ i1,]),
          dd.== predict(r1., new = d1[ i1,]))

plot(tr1, cex=2, col = c11, bg = c11, pch = 20+c11,
      xlim=c(1,30), ylim= c(0,10))
xy <- cbind(x= runif(500, min=1,max=30), y = runif(500, min=0, max=10))
points(xy, cex= 0.5, col = predict(r1, new = xy))
abline(v=c( mean(c(5,25)), mean(c(25,29))))

## example where one variable xj has Var(xj) = 0:
x4 <- matrix(c(2:4,7, 6,8,5,6, 7,2,3,1, 7,7,7,7), ncol=4)
y <- c(2,2, 1,1)
m4.1 <- dDA(x4, y, pool = FALSE)
m4.2 <- dDA(x4, y, pool = TRUE)
xx <- matrix(c(3,7,5,7), ncol=4)
predict(m4.1, xx)## gave integer(0) previously
predict(m4.2, xx)
```

diagX	<i>The “Other” Diagonal Matrix</i>
-------	------------------------------------

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.

See Also

[diag](#).

Examples

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

digitsBase	<i>Digit/Bit Representation of Integers in any Base</i>
------------	---------------------------------------------------------

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(log(max(x),base)))
## S3 method for class 'basedInt'
as.integer(x, ...)
## S3 method for class 'basedInt'
print(x, ...)

as.intBase(x, base = 2)

```

Arguments

x	For <code>digitsBase()</code> : non-negative integer (vector) whose base base digits are wanted. For <code>as.intBase()</code> : a list of numeric vectors, a character vector, or an integer matrix as returned by <code>digitsBase()</code> , 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 <code>print</code> .

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.

Note

`digits` and `digits.v` are now deprecated and will be removed from 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(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

## Inverse of digitsBase() : as.integer method for the "basedInt" class
as.integer(M)
## or also as.intBase() working from strings:
(cb <- apply(digitsBase(0:33, 4), 2, paste, collapse = ""))
##-> "000" "001" ..... "200" "201"
all(0:33 == as.intBase(cb, base = 4))
```

Duplicated

*Counting-Generalization of duplicated()***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

```
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

```
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 expressions](#) of the form $k \times 10^k$ for labeling a log-scaled axis and when otherwise exponential formatting would be used.

Usage

```
eaxis(side, at = ,
      labels = NULL, log = NULL,
      f.smalltcl = 3/5, at.small = NULL, small.mult = NULL,
      draw.between.ticks = TRUE,
      outer.at = TRUE, drop.1 = TRUE, las = 1,
      nintLog = max(10, par("lab")[2 - is.x]), max.at = Inf, ...)
```

Arguments

side	integer in 1:4, specifying side of <code>axis</code> .
at	numeric vector of (“normalized”) tick locations; by default <code>axTicks(side, ..)</code> , i.e., the same as <code>axis()</code> would use.
labels	NULL (default), logical , character or expression, as in <code>axis()</code> ; in addition, if NA, <code>labels = TRUE</code> is passed to <code>axis()</code> , i.e. <code>pretty10exp</code> is <i>not</i> 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 normalized, labeled ticks.
at.small	locations of <i>small</i> ticks; the default, NULL, uses <code>small.mult</code> and constructs “smart” locations.
small.mult	positive integer (or NULL), used when <code>at.small</code> is NULL to indicate which multiples of <code>at</code> (typically <code>axTicks()</code>) should be used as “small ticks”. The default NULL will use 9 in the log case and a number in 2:5 otherwise.

<code>draw.between.ticks</code>	(only if <code>log</code> is true): logical indicating that possible (non-small) ticks between the labeled (via <code>at</code>) ones should be drawn as well (and possibly also used for <code>at.small</code> construction).
<code>outer.at</code>	logical specifying that <code>at.small</code> should also be constructed outside the <code>at</code> range, but still inside the corresponding <code>par("usr")</code> .
<code>drop.1</code>	logical specifying if "1 *" should be dropped from labels, passed to <code>pretty10exp()</code> .
<code>nintLog</code>	only used in R > 2.13.x, when <code>log</code> is true: approximate (lower bound on) number of intervals for log scaling.
<code>max.at</code>	maximal number of <code>at</code> values to be used effectively. If you don't specify at yourself carefully, it is recommended to set this to something like 25.
<code>las, ...</code>	arguments passed to (the first) <code>axis</code> call. Note that the default <code>las = 1</code> differs from <code>axis</code> 's default <code>las = 0</code> .

Author(s)

Martin Maechler

See Also

[axis](#), [axTicks](#), [axTexpr](#), [pretty10exp](#).

Examples

```
x <- lseq(1e-10, 0.1, length = 201)
plot(x, pt(x, df=3), type = "l", xaxt = "n", log = "x")
eaxis(1)
## 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, 0.1, length = 1000)
plot(x, sin(1/x)*x, type = "l", xaxt = "n", log = "x")
eaxis(1)

## 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(1)

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

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

```

eaxis(2)
## more labels than default:
op <- par(lab=c(10,5,7))
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis(1) # 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(1)[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)

## From R version 2.14.0, on, this looks better

```

ecdf.ksCI

Plot the Empirical Distribution Function Together With 95% Confidence Curves.

Description

Plots the empirical distribution function for univariate data, together with upper and lower simultaneous 95% confidence curves.

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.
...	... arguments given 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

ReferencesBickel and Doksum, see [KSd](#).**See Also**[ecdf](#) and [plot.stepfun](#) in package [stepfun](#).**Examples**

```
ecdf.ksCI( rchisq(50,3) )
```

 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

<code>a, b</code>	length of half axes in (x,y) direction. Note that (a, b) is equivalent to (b, a) <i>unless</i> <code>keep.ab.order=TRUE</code> .
<code>alpha</code>	angle (in degrees) giving the orientation of the ellipse, i.e., the original (x,y)-axis ellipse is rotated by <code>angle</code> .
<code>loc</code>	center (LOCation) of the ellipse.
<code>n</code>	number of points to generate.
<code>keep.ab.order</code>	logical indicating if (a, b) should be considered <i>ordered</i> . When <code>FALSE</code> , as per default, the orientation of the ellipse is solely determined by <code>alpha</code> . Note that <code>keep.ab.order = TRUE</code> seems a more natural default, but <code>FALSE</code> is there for back-compatibility.

Value

A numeric matrix of dimension $n \times 2$, 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:
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 :
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
if(dev.interactive()) {
  ## 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

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

Arguments

`a` an [array](#), especially 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

```
empty.dimnames(diag(5)) # looks much nicer

(a <- matrix(-9:10, 4,5))
empty.dimnames(a) # nicer, right?
```

errbar

Scatter Plot with Error Bars

Description

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

Usage

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

Arguments

<code>x</code>	vector of x values.
<code>y</code>	vector of y values.
<code>yplus</code>	vector of y values: the tops of the error bars.
<code>yminus</code>	vector of y values: the bottoms of the error bars.
<code>cap</code>	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.
<code>ylim</code>	(numeric of length 2): the y-axis extents with a sensible default.
<code>xlab, ylab</code>	axis labels for the plot, as in plot.default .
<code>...</code>	Graphical parameters (see par) may also be supplied as arguments to this function.

Author(s)

Originally Charles Geyer, U.Chicago, early 1991; then Martin Mächler.

See Also

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

Examples

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

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

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

Arguments

<code>object</code>	result of <code>rlm()</code> .
<code>var</code>	variables. Either their names or their indices; the default, <code>-1</code> means all <i>but</i> 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

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

Author(s)

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

References

FIXME — Need some here !

See Also

[rlm](#), [summary.aov](#), etc.

Examples

```
if(require("MASS")) {  
  ## same data as example(rlm)  
  data(stackloss)  
  summary(rsl <- rlm(stack.loss ~ ., stackloss))  
  f.robftest(rsl)  
} else " forget it "
```

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.

See Also

[primes](#).

For factorization of moderately or really large numbers, see the **gmp** package, and its [factorize\(\)](#).

Examples

```
factorize(47)
factorize(seq(101, 120, by=2))
```

hatMat

Hat Matrix of a Smoother

Description

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

Usage

```
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 $n <- \text{length}(x)$ also in the case where some x values are duplicated (aka *ties*).

Author(s)

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

References

Hastie and Tibshirani (1990). *Generalized Additive Models*. Chapman & Hall.

See Also

[smooth.spline](#), etc. Note the demo, `demo("hatmat-ex")`.

Examples

```
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\n", sum(diag(Hk))))
image(Hk)
Matrix::printSpMatrix(as(round(Hk, 2), "sparseMatrix"))

##---> see demo("hatmat-ex") for more (and larger) examples
```

hist.bxp

Plot a Histogram and a Boxplot

Description

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

Usage

```
hist.bxp(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

<code>x</code>	numeric vector of data for histogram. Missing values (NAs) are allowed.
<code>nclass</code>	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 <code>x</code> .
<code>breaks</code>	vector of the break points for the bars of the histogram. The count in the <i>i</i> -th bar is <code>sum(breaks[i] < x <= breaks[i+1])</code> except that if <code>include.lowest</code> is TRUE (the default), the first bar also includes points equal to <code>breaks[1]</code> . If omitted, evenly-spaced break points are determined from <code>nclass</code> and the extremes of the data.
<code>probability</code>	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.
<code>include.lowest</code>	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 <code>breaks</code> argument).
<code>xlab</code>	character or expression for x axis labeling.
<code>...</code>	additional arguments to <code>barplot</code> . The <code>hist</code> function uses the function <code>barplot</code> to do the actual plotting; consequently, arguments to the <code>barplot</code> function that control shading, etc., can also be given to <code>hist</code> . See the <code>barplot</code> documentation for arguments <code>angle</code> , <code>density</code> , <code>col</code> , and <code>inside</code> . Do not use the space or <code>histo</code> arguments.
<code>width</code>	width of the box relative to the height of the histogram. DEFAULT is 0.2.
<code>boxcol</code>	color of filled box. The default is 3.
<code>medcol</code>	the color of the median line. The special value, NA, indicates the current plotting color (<code>par("col")</code>). The default is 2. If <code>boxcol=0</code> and <code>medcol</code> is not explicitly specified this is set to the current plotting color (<code>par("col")</code>).
<code>medlwd</code>	median line width. The special value NA, is used to indicate the current line width (<code>par("lwd")</code>). The default is 5.
<code>whisklty</code>	whisker line type. The special value NA indicates the current line type (<code>par("lty")</code>). The default is 2 (dotted line).
<code>staplelty</code>	staple (whisker end cap) line type. The special value NA indicates the current line type (<code>par("lty")</code>). The default is 1 (solid line). Graphical parameters (see <code>par</code>) may also be supplied as arguments to this function. In addition, the high-level graphics arguments described under <code>par</code> and the arguments to <code>title</code> 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.

Author(s)

S-Plus: Markus Keller, Christian Keller; port to R: Martin Mächler.

See Also

[hist](#), [barplot](#), [boxplot](#), [rug](#) and [scat1d](#) in the **Hmisc** package.

Examples

```
lab <- "50 samples from a t distribution with 5 d.f."
mult.fig(2*3, main = "Hist() + Rug() and Hist.bxp(*)")
for(i in 1:3) {
  my.sample <- rt(50, 5)
  hist(my.sample, main=lab); rug(my.sample)# for 50 obs., this is ok, too..
  hist.bxp(my.sample, main=lab)
}
```

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 $\sqrt{.Machine\$double.eps}$(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.

Author(s)

Martin Maechler, May 1994 (for S).

See Also

[integrate](#) for numerical integration of *functions*.

Examples

```
x <- 1:4
integrate.xy(x, exp(x))
print(exp(4) - exp(1), digits = 10) # the true integral

for(n in c(10, 20,50,100, 200)) {
  x <- seq(1,4, len = n)
  cat(formatC(n,wid=4), formatC(integrate.xy(x, exp(x)), dig = 9),"\n")
}
```

inv.seq

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))))
```

iterate.lin.recursion *Generate Sequence Iterating a Linear Recursion*

Description

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

Usage

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

Arguments

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

```
## 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)))
```

KSd

Approximate Critical Values for Kolmogorov-Smirnov's D

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%).

Author(s)

Kjetil Halvorsen and Martin Maechler

References

Peter J. Bickel and Kjell A. Doksum (1977), *Mathematical Statistics: Basic Ideas and Selected Topics*. Holden Day. Section 9.6 and table IX.

See Also

Is used from [ecdf.ksCI](#).

Examples

```
KSd(90)
KSd(1:9)# now works

op <- par(mfrow=c(2,1))
plot(KSd, 10, 150)# nice
abline(v = c(75,85), col = "gray")
plot(KSd, 79, 81, n = 1001)# *very* tiny discontinuity at 80
par(op)
```

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

<code>x</code>	any vector.
<code>length.out</code>	integer indicating how many element are desired. If positive, return the <code>length.out</code> last elements of <code>x</code> ; if negative, the last <code>length.out</code> elements are <i>dropped</i> .
<code>na.rm</code>	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 new (R 1.9.0) 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

```
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

```
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 <code>lm(.)</code> .
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 <code>Inf</code> ; 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 `linesHyperb.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

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

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", nam.center = "1", col.center = "c",
        append = TRUE, digits = 3, title)
```

Arguments

<code>x</code>	a matrix
<code>file</code>	names the file to which LaTeX commands should be written
<code>nam.center</code>	character specifying row names should be center; default "1".
<code>col.center</code>	character (vector) specifying how the columns should be centered; must have values from <code>c("l", "c", "r")</code> ; defaults to "c".
<code>append</code>	logical; if FALSE, will destroy the file <code>file</code> before writing commands to it; otherwise (by default), simply adds commands at the end of file <code>file</code> .
<code>digits</code>	integer; setting of <code>options(digits=..)</code> for purpose of number representation.
<code>title</code>	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 <vjcarey@sphunix.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

```
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|}
##> \multicolumn{ 4 }{c}{ $\pi$, e$, etc. } \\ \hline
##> \ & a & b & c \\ \hline \hline
##> $\pi$ & 3.14 & 1.57 & 0.785 \\ \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"))

```

mpl

Simple Matrix Plots

Description

Do simple matrix plots, providing an easy interface to [matplotlib](#) by using a default x variable.

Usage

```

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

```

Arguments

mat	numeric matrix.
...	further arguments passed to matplotlib , e.g., type, xlab, etc.

Details

`p.m(m)` use the first column of `m` as x variable, whereas `mpl(m)` uses the integers `1, 2, ..., nrow(m)` as coordinates and `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 `matplotlib()` directly is more appropriate.

Author(s)

Martin Maechler

See Also

`matplot`, `plot.mts`(*, `plot.type = "single"`).

Examples

```
data(animals, package = "cluster")
mpl(animals, type = "l")
```

mult.fig

Plot Setup for MULTiple FIGures, incl. Main Title

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 `par` calls.

Usage

```
mult.fig(nr.plots, mfrow, mfcol,
         marP = rep(0, 4), mgp = c(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,
         quiet = .Device == "postscript",
         cex.main = par("cex.main"), line.main = cex.main - 1/2,
         col.main = par("col.main"), font.main = par("font.main"), ...)
```

Arguments

<code>nr.plots</code>	integer; the number of plot figures you'll want to draw.
<code>mfrow</code>	<i>instead of nr.plots</i> : integer(2) vector giving the rectangular figure layout for <code>par(mfrow= .)</code> .
<code>mfcol</code>	<i>instead of nr.plots</i> : integer(2) vector giving the rectangular figure layout for <code>par(mfcol= .)</code> .
<code>marP</code>	numeric(4) vector of figure margins to <i>add</i> ("Plus") to default <code>mar</code> , see below.
<code>mgp</code>	argument for <code>par(mgp= .)</code> with a smaller default than usual.
<code>mar</code>	argument for <code>par(mar= .)</code> with a smaller default than usual, using the <code>marP</code> argument, see above.
<code>oma</code>	argument for <code>par(oma= .)</code> , by default for adding space for the main title if necessary.
<code>main</code>	character. The main title to be used for the whole graphic.
<code>tit.wid</code>	numeric specifying the vertical width to be used for the main title; note that this is only used for the default value of <code>oma</code> (s. above).
<code>cex.main</code>	numeric; the character size to be used for the main title.

`line.main` numeric; the line at which the title.
`col.main, font.main` color and font for main title; see [par\(*\)](#) for details.
`quiet` logical; if TRUE, do *not* write the reminder about resetting [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.

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 [plot](#)(*, type="n") and [text](#). To have names or numbers instead of points in a plot is useful for identification, e.g., in a residual plot, see also [TA.plot](#).

Usage

```
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"), ...)
```

Arguments

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

Value

invisibly, a character vector with the labels used.

Author(s)

Martin Maechler, since 1992

See Also

[plot.default](#), [text](#).

Examples

```
n.plot(1:20, cumsum(rnorm(20)))
data(cars)
with(cars, n.plot(speed, dist, cex = 0.8, col = "forest green"))
```

nearcor

Find the Nearest Proper Correlation Matrix

Description

This function smooths an improper correlation matrix as it can result from [cor](#) with `use="pairwise.complete.obs"` or [hetcor](#).

Usage

```
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 constraints 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`); [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))
  x2 <- ordered(as.integer(x2 > 0))
  x3 <- ordered(as.integer(x3 > 1))
  x4 <- ordered(as.integer(x4 > -1))

  y1 <- ordered(as.integer(y1 > 0))
  y2 <- ordered(as.integer(y2 > 0))
  y3 <- ordered(as.integer(y3 > 1))
  y4 <- ordered(as.integer(y4 > -1))

  odat <- data.frame(x1, x2, x3, x4, y1, y2, y3, y4)

  xcor <- cor(dat)
  pcor <- cor(data.matrix(odat)) # cor() no longer works for factors
  hcor <- hetcor(odat, ML=TRUE, std.err=FALSE)$correlations
  ncor <- nearcor(hcor)$cor

  try(factanal(covmat=xcor, factors=2, n.obs=n))
  try(factanal(covmat=pcor, factors=2, n.obs=n))
  try(factanal(covmat=hcor, factors=2, n.obs=n))
  try(factanal(covmat=ncor, factors=2, n.obs=n))
}

```

}

`nr.sign.chg`*Number of Sign Changes in Sequence*

Description

Compute the number of sign changes in the sequence `y`.

Usage

```
nr.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 <code>segments()</code> .

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[s], y[s], x[s+1], y[s+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.date

Usage

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

Arguments

outer	logical; passed to <code>mtext</code> .
cex	non-negative; passed to <code>mtext</code> .
...	any arguments to <code>u.Datumvonheute</code> .

See Also

[u.date](#), [date](#).

Examples

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

p.dnorm

*Plot Parametric Density Functions***Description**

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

Usage

```
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 χ^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 ± 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

```
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)
```

p.hboxp

Add a Horizontal Boxplot to the Current Plot

Description

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

Usage

```
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

```
## ==> See code in 'hist.bxp' (.) and example(hist.bxp) !  
##
```

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

```
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`-internal `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:::plot.profile.nls` methods).

Examples

```
require(stats)
data(Puromycin)
Treat <- Puromycin[Puromycin$state == "treated", ]
fm <- nls(rate ~ T1*conc/(T2+conc), data=Treat,
         start = list(T1=207,T2=0.06))
(pr <- profile(fm)) # quite a few things..
op <- par(mfcol=1:2)
plot(pr) # -> 2 'standard' plots
par(op)
## ours:
p.profileTraces(pr)
```

p.res.2fact *Plot Numeric (e.g. Residuals) vs 2 Factors Using Boxplots*

Description

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

Usage

```
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 <code>boxplot(*, notch)</code> .
xlab,ylab	axis labels, see <code>plot.default</code> , per default the actual argument expressions.
main	main title passed to <code>plot</code> , 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

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

## Real data
data(warpbreaks)
(fm1 <- lm(breaks ~ wool*tension, data = warpbreaks))
with(warpbreaks,
     p.res.2fact(wool, tension, residuals(fm1)))
```

p.res.2x

*Stahel's Residual Plot against 2 X's***Description**

Plot Residuals (e.g., of a multiple linear regression)

Usage

```
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), ...)
```

Arguments

x,y	numeric vectors of the same length specifying 2 covariates.
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., main = "".
xlim, ylim	the basic x- and y- axis extents, see plot.default . Note that these will be slightly extended such that segments are not cut off.
...	further arguments passed to plot.

Details

.....

Author(s)

Andreas Ruckstuhl in June 1991 and Martin Maechler, in 1992, '94, 2003-4.

References

Stahel, W. (1996)

See Also

[p.res.2fact](#), [plot.lm](#), [TA.plot](#).

Examples

```
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)
mult.fig(2, marP=c(0,-1,-2,0), main="p.res.2x(*,*, residuals(lm.SR))")
with(LifeCycleSavings,
     { p.res.2x(pop15, ddpi, residuals(lm.SR), scol=c("red", "blue"))
       p.res.2x(pop75, dpi, residuals(lm.SR), scol=2:1)
     })
```

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 = relsize * 2.54 * min(pin), relsize = 0.05)
```

Arguments

unit	length of unit (or x and y units) of symbol coordinates in cm.
relsize	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 <code>boxplot</code> or <code>rrange</code> 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 <code>plot.default</code> .
...	further arguments to <code>plot</code> . Graphical parameters (see <code>par</code>) 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 <keller@stat.math.ethz.ch>, 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")
```

p.ts

plot.ts with multi-plots and Auto-Title – on 1 page

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 %% 400)), 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

<code>x</code>	timeseries (possibly multivariate) or numeric vector.
<code>nrplots</code>	number of sub-plots. Default: in $\{1..8\}$, approximately $n/400$ if possible.
<code>overlap</code>	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 DateTimeClasses).
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)
```

paste.vec

Utility for 'Showing' S vectors

Description

A simple utility for displaying simple S vectors; can be used as debugging utility.

Usage

```
paste.vec(name, digits = options()$digits)
```

Arguments

name string with an 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

```
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

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

Arguments

<code>x</code> , <code>yd</code> , <code>ys</code>	numeric vectors all of the same length, representing (x_i, y_i) and fitted (smooth) values \hat{y}_i . <code>x</code> will be sorted increasingly if necessary, and <code>yd</code> and <code>ys</code> accordingly. Alternatively, <code>ys</code> can be an x-y list (as resulting from <code>xy.coords</code>) containing fitted values on a finer grid than the observations <code>x</code> . In that case, the observational values <code>x[]</code> must be part of the larger set; <code>seqXtend()</code> may be applied to construct such a set of abscissa values.
<code>xlab</code> , <code>ylab</code>	x- and y- axis labels, as in <code>plot.default</code> .
<code>ylim</code>	limits of y-axis to be used; defaults to a <i>robust</i> range of the values.
<code>xpd</code>	see <code>par(xpd=.)</code> ; by default do allow to draw outside the plot region.
<code>do.seg</code>	logical indicating if residual segments should be drawn, at <code>x[i]</code> , from <code>yd[i]</code> to <code>ys[i]</code> (approximately, see <code>seg.p</code>).
<code>seg.p</code>	segment percentage of segments to be drawn, from <code>yd</code> to <code>seg.p*ys + (1-seg.p)*yd</code> .
<code>segP</code>	list with named components <code>lty</code> , <code>lwd</code> , <code>col</code> specifying line type, width and color for the residual segments, used only when <code>do.seg</code> is true.
<code>linP</code>	list with named components <code>lty</code> , <code>lwd</code> , <code>col</code> specifying line type, width and color for “smooth curve lines”.
<code>...</code>	further arguments passed to <code>plot</code> .

Note

Non-existing components in the lists `segP` or `linP` will result in the `par` defaults to be used.
`plotDS()` used to be called `pl.ds` up to November 2007.

Author(s)

Martin Maechler, since 1990

See Also

`seqXtend()` to construct more smooth `ys` “objects”.

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$Time, seq(0, 25, length = 201))))
ysm <- c(predict(fm1, newdata = list(Time = xsm)))
plotDS(Th4$Time, Th4$conc, ys = list(x=xsm, y=ysm),
       sub = "Theophylline data - Subject 4 only",
       segP = list(lwd=2), las = 1)
```

plotStep

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., " <i>continue à droite, limite à gauche</i> ". 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 <code>par()</code> .
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 <code>plot(..)</code> .

Value

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

Side Effects

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

Author(s)

Martin Maechler, Seminar for Statistics, ETH Zurich, <maechler@stat.math.ethz.ch>, 1991 ff.

See Also

The `plot` methods `plot.ecdf` and `plot.stepfun` in R which are conceptually nicer.
`segments(..., method = "constant")`.

Examples

```
##-- Draw an Empirical CDF (and see the default title ..)
plotStep(rnorm(15))

plotStep(runif(25), cad.lag=FALSE)
plotStep(runif(25), cad.lag=FALSE, add=TRUE, lty = 2)

ui <- sort(runif(20))
plotStep(ui, ni <- cumsum(rpois(19, lambda=1.5) - 1.5), cad.lag = FALSE)
plotStep(ui, ni, verticals = TRUE, right.points = TRUE)

plotStep(rnorm(201), pch = '.') #- smaller points
```

pmax.sa

Parallel Maxima / Minima (Scalar, Array)

Description

These are versions of `pmax` and `pmin` which return (more-dimensional) arrays.

Usage

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

Arguments

<code>scalar</code>	numeric scalar.
<code>arr</code>	any numeric R object, typically array.

Value

an array with the same `dim` and `dimnames` as the input `arr`.

Author(s)

Martin Maechler

See Also

[pmin](#).

Examples

```
m <- cbind(a=1:5, b=3:7, c=-2:2)
pmax  (3, m) # < dropping all attributes
pmax.sa(3, m) # matrix with dimnames

## no "added value" for simple case:
x <- rpois(20, 7)
stopifnot(identical(pmin  (2.1, x),
  pmin.sa(2.1, x)),
  identical(pmax  (2.1, 3),
  pmax.sa(2.1, 3)))
```

polyn.eval

Evaluate Polynomials

Description

Evaluate one or several univariate polynomials at several locations, i.e. compute $\text{coef}[1] + \text{coef}[2]*x + \dots + \text{coef}[p+1]*x^p$ (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 $\text{length}(x) * \text{nrow}(\text{coef})$.

`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.

Value

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

Author(s)

Martin Maechler, ages ago.

See Also

For much more sophisticated handling of polynomials, use the `polynom` package, e.g. [predict.polynomial](#).

Examples

```
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)
```

 posdefify

Find a Close Positive Definite Matrix

Description

From a matrix `m`, construct a "close" positive definite one.

Usage

```
posdefify(m, method = c("someEVadd", "alleVadd"),
          symmetric = TRUE, eigen.m = eigen(m, symmetric= symmetric),
          eps.ev = 1e-07)
```

Arguments

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

Details

We form the eigen decomposition

$$m = V\Lambda V'$$

where Λ is the diagonal matrix of eigenvalues, $\Lambda_{j,j} = \lambda_j$, with *decreasing* eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.

When the smallest eigenvalue λ_n are less than `Eps <- eps.ev * abs(lambda[1])`, i.e., negative or “almost zero”, some or all eigenvalues are replaced by *positive* (\geq 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 Eps).

Value

a matrix of the same dimensions and the “same” diagonal (i.e. `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 `nearPD()` function in the **Matrix** package.

Author(s)

Martin Maechler, July 2004

References

Section 4.4.2 of Gill, P.-E., Murray, W. and Wright, M.-H. (1981) *Practical Optimization*, Academic Press.

Cheng, Sheung Hun and Higham, Nick (1998) A Modified Cholesky Algorithm Based on a Symmetric Indefinite Factorization; *SIAM J. Matrix Anal. Appl.*, **19**, 1097–1110.

Knol DL, ten Berge JMF (1989) Least-squares approximation of an improper correlation matrix by a proper one. *Psychometrika* **54**, 53–61.

Higham (2002) Computing the nearest correlation matrix - a problem from finance; *IMA Journal of Numerical Analysis* **22**, 329–343.

Lucas (2001) Computing nearest covariance and correlation matrices. A thesis submitted to the University of Manchester for the degree of Master of Science in the Faculty of Science and Engineering.

See Also

`eigen` on which the current methods rely. `nearPD()` in the **Matrix** package.

Examples

```
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))
```

 potatoes

Fisher's Potato Crop Data

Description

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

Usage

```
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

Bennett, J. H. (1972) *Collected Papers of R. A. Fisher* vol.~II, 1925-31; The University of Adelaide.

References

T.Eden and R. A. Fisher (1929) Studies in Crop Variation. VI. Experiments on the Response of the Potato to Potash and Nitrogen. *J. Agricultural Science* **19**, 201–213. Accessible from Bennett (1972), see above.

Examples

```
data(potatoes)
## See the experimental design:
with(potatoes, {
  cat("4 blocks of experiments;",
      "each does every (nitrogen,potash) combination (aka 'treat'ment) 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

```
pretty10exp(x, drop.1 = FALSE, digits.fuzz = 7)
```

Arguments

x	numeric vector (e.g. axis tick locations)
drop.1	logical indicating if $1 \times$ should be dropped from the resulting expressions.
digits.fuzz	number of digits to be considered for integer equality; do not change lightly!

Value

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

Author(s)

Martin Maechler

See Also

[axTexpr](#) which builds on pretty10exp(); further [axis](#), [axTicks](#).

Examples

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

ax <- 10^(-6:0) - 2e-16
pretty10exp(ax, drop.1=TRUE)
## 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 [integers](#) to be quite fast.

Usage

```
primes(n)
```

Arguments

n a (typically positive integer) number.

Details

As the function only uses [max](#)(n), n can also be a *vector* of numbers.

Value

numeric vector of all prime numbers $\leq n$.

Author(s)

Bill Venables (≤ 2001); Martin Maechler gained another 40% speed, working with [logicals](#) and [integers](#).

See Also

[factorize](#); [next](#) in ‘‘base’’ R’s ([stats](#) package).

Examples

```
(p1 <- primes(100))
system.time(p1k <- primes(1000)) # still lightning ..
stopifnot(length(p1k) == 168)
```

`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

```
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

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

```
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"])
```

prt.DEBUG

Utility Printing in DEBUG mode

Description

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`.

The global `DEBUG` has been a cheap precursor to the now standard R's `options(verbose= .)` setting.

Usage

```
prt.DEBUG(..., LEVEL = 1)
```

Arguments

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

Note

This is mainly kept for historical reasons (and old code fragments), but sometimes I still consider renaming it and have it work using `getOption("verbose")` alone.

Author(s)

Martin Maechler, originally for S-PLUS.

ps.end	<i>Close PostScript or Acrobat Graphics Device opened by 'ps.do' / 'pdf.do'</i>
--------	---------------------------------------------------------------------------------

Description

Closes the PostScript or PDF file ([postscript.pdf](#)), opened 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 <code>command</code>) 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, <code>getOption("eps_view")</code> is set to <code>gv -watch -geometry -0+0 -magstep -2 -media BBox -noantialias</code> which assumes <code>gv</code> (aka <i>ghostview</i>) 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()
) {
  ps.do("ex.ps")
  data(sunspots)
  plot(sunspots)
```

```

ps.end()

pdf.latex("ex-sun.pdf")
plot(sunspots)
pdf.end(call. = FALSE) # basically the same as dev.off()
}
ps.latex("ex2.eps")
plot(sunspots)
ps.end(call.gv = FALSE) # basically the same as dev.off()

```

ps.latex

PostScript/PDF Preview Device with Optional 'LaTeX' Touch

Description

All functions start a pseudo PostScript or Acrobat preview device, using `postscript` or `pdf`, and further registering the file name for subsequent calls to `pdf.end()` or `ps.end()`.

Usage

```
pdf.do(file, paper = "default", width = -1, height = -1, onefile = FALSE,
       title = NULL, version = "1.4", ...)
```

```
pdf.latex(file, height = 5 + main.space * 1.25, width = 9.5,
          main.space=FALSE, lab.space = main.space,
          paper = "special", title = NULL,
          lab=c(10, 10, 7), mgp.lab=c(1.6, 0.7, 0), mar=c(4, 4, 0.9, 1.1), ...)
```

```
ps.do(file, width=-1, height=-1, onefile=FALSE, horizontal=FALSE,
      title = NULL, ...)
```

```
ps.latex(file, height = 5 + main.space * 1.25, width = 9.5,
         main.space=FALSE, lab.space = main.space,
         paper = "special", title = NULL,
         lab=c(10, 10, 7), mgp.lab=c(1.6, 0.7, 0), mar=c(4, 4, 0.9, 1.1), ...)
```

Arguments

file	character giving the PostScript/PDF file name to be written.
height	device height in <i>inches</i> , <code>height * 2.54</code> are <i>cm</i> . The default is 5 plus 1.25 iff <code>main.space</code> .
width	device width in <i>inches</i> ; for this and height, see <code>postscript</code> .
onefile, horizontal	logicals passed to <code>postscript(...)</code> or <code>pdf(...)</code> , most probably to be left alone.
title	PostScript/PDF (not plot!) title passed to <code>postscript()</code> or <code>pdf()</code> ; 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.
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 <i>not</i> subtracting from mar).
paper	character (or missing), 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\("papersize"\)](#).

Value

A list with components

old.par	containing the old par values
new.par	containing the newly set par values

Author(s)

Martin Maechler

See Also

[ps.end](#), [pdf](#), [postscript](#), [dev.print](#).

Examples

```
if(interactive()) {
  ps.latex("ps.latex-ex.ps", main= TRUE)
  data(sunspots)
  plot(sunspots,main=paste("Sunspots Data, n=",length(sunspots)),col="red")
  ps.end()
}
```

```

pdf.latex("pdf.latex-ex.pdf", main= TRUE)
  data(sunspots)
  plot(sunspots,main=paste("Sunspots Data, n=",length(sunspots)),col="red")
pdf.end()

ps.do("ps_do_ex.ps")
  example(plot.function)
ps.end()

pdf.do("pdf_do_ex.pdf", width=12, height=5)
  plot(sunspots, main="Monthly Sunspot numbers (in Zurich, then Tokyo)")
pdf.end()
}

```

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](#).

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))

```

Description

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

Usage

```
sHalton(n.max, n.min = 1, base = 2, leap = 1)
QUnif (n, min = 0, max = 1, n.min = 1, p, leap = 1)
```

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, leap = 1 and the maximal sequence number is n.max = n.min + (n-1)*leap.
base	integer ≥ 2 : The base with respect to which the Halton sequence is built.
min, 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.
leap	integer indicating (if > 1) if the series should be leaped, i.e., only every leapth entry should be taken.

Value

sHalton(n,m) returns a numeric vector of length $n-m+1$ of values in $[0, 1]$.

QUnif(n, min, max, n.min, p=p) generates $n-n.min+1$ p -dimensional points in $[min, max]^p$ 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

James Gentle (1998) *Random Number Generation and Monte Carlo Simulation*; sec.\ 6.3. Springer.
 Kocis, L. and Whiten, W.J. (1997) Computational Investigations of Low-Discrepancy Sequences. *ACM Transactions of Mathematical Software* **23**, 2, 266–294.

Examples

```

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 <- mult.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")

```

repChar

Make Simple String from Repeating a Character, e.g. Blank String

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
```

rot2

Rotate Planar Points by Angle

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., $\text{phi}=\pi$ 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

```
## 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,.8), 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=0.1, col="gray40"))
for(phi in pi/4 * 1:7)
  polygon(rot2(xy0, phi = phi), col = 1+phi/(pi/4), border=2, type = "b")
```

roundfixS

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, $\text{sum}(x) == \text{sum}(y)$.

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 $\text{sum}(r) == \text{sum}(x)$.

Author(s)

Martin Maechler, November 2007

References

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

http://en.wikipedia.org/wiki/Appportionment_paradox

<http://www.ams.org/samplings/feature-column/fcarc-apportionii3>

See Also

[round](#) etc

Examples

```
## 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="1g")
  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`.

Usage

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

Arguments

<code>x</code>	numeric vector the robust range of which shall be computed.
<code>range</code>	number for S compatibility; $1.5 * \text{range}$ is equivalent to <code>coef</code> .
<code>coef</code>	numeric multiplication factor defining the outlier boundary, see “Details” below.
<code>na.rm</code>	logical indicating how NA values should be handled; they are simply dropped when <code>na.rm = TRUE</code> 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 $m \leq M$ which is (not strictly) inside $\text{range}(x) = c(\min(x), \max(x))$.

Author(s)

Martin Maechler, 1990.

See Also

[range](#), [fivenum](#), [boxplot](#).

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

```
seqXtend(x, length., method = c("simple", "aim", "interpolate"),
         from = NULL, to = NULL)
```

Arguments

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

Value

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

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

Note

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

`seq`; `plotDS` can make particularly good use of `seqXtend()`

Examples

```
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:
xt <- 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 ...
xt2 <- seqXtend(x, 100)
plot(xt2, col="blue")
points(match(x,xt2), 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
```

Description

Rounds to significant digits similarly to [signif](#).

Usage

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

Arguments

<code>x</code>	numeric vector to be rounded.
<code>digits</code>	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

```
(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

```
sourceAttach(file, pos=2,
             name = paste(abbreviate(gsub(fsep,"", 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

```
sourceAttach(system.file("test-tools.R", package="Matrix"))
search() # shows the new "data base" at position 2
ls(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.
 ... potential 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)
df1 <- str_data("datasets", filterFUN=is.data.frame)
df.d <- df1$datasets
## work around bug in R < 2.14.1 (?) {for *some* setups only}:
str(df.d <- df.d[sapply(df.d, is.character)])
## 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")])
```

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.MIPS()

```

Arguments

procfile name of file the lines of which give the CPU info “as on Linux”

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.MIPS returns a number giving an approximation of the **M**illion **I**nstructions **P**er **S**econd that the CPU processes. This is a performance measure of the basic *non-numeric* processing capabilities and for Linux systems often about twice the basic clock rate in “MHz” as available by `Sys.cpuinfo()[“cpu MHz”]`.

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

See Also

[Sys.ps](#), etc.

Examples

```

if(substr(R.version[["os"]], 1,5) == "linux") { ##-- only on Linux

  Sys.cpuinfo() # which is often ugly
  local({I <- Sys.cpuinfo(); I[ "flags" != names(I) ] })
  Sys.MIPS()
  Sys.MIPS() / as.numeric(Sys.cpuinfo()[“cpu MHz”]) ## < often about 2
}

```

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.cmd()
```

```
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

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

 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

```
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),
        ...)
```

Arguments

lm.res	Result of <code>lm(..)</code> , <code>avov(..)</code> , <code>glm(..)</code> 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 <code>lm.res</code> \$ <code>call</code> .
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 $\pm 2\sigma$ where σ is <code>mad(resid)</code> .
lo.iter	positive integer, giving the number of lowess robustness iterations. The default depends on the model and is 0 for non Gaussian <code>glm</code> 's.
lo.cex	character expansion ("cex") for lowess and other marginal texts.
par0line	a list of arguments (with reasonable defaults) to be passed to <code>abline(.)</code> 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.

... 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

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

example(airquality)
summary(lm0 <- lm(Ozone ~ ., data= airquality))
TA.plot(lm0)
TA.plot(lm0, label = "0") # instead of case numbers

if(FALSE) {
  TA.plot(lm(cost ~ age+type+car.age, claims, weights=number, na.action=na.omit))
}

##--- for aov(.) : -----
data(Gun, package = "nlme")
TA.plot( aov(rounds ~ Method + Physique/Team, data = Gun))

##--- Not so clear what it means for GLM, but: -----
if(require(rpart)) { # for the two datasets only
  data(solder, package = "rpart")
  TA.plot(glm(skips ~ ., data = solder, family = poisson), cex= .6)

  data(kyphosis, package = "rpart")
  TA.plot(glm(Kyphosis ~ poly(Age,2) + Start, data=kyphosis, family = binomial),
  cex=.75) # smaller title and plotting characters
}
```

tapplySimpl	<i>More simplification in tapply() result</i>
-------------	-----------------------------------------------

Description

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

Usage

```
tapplySimpl(X, INDEX, FUN, ...)
```

Arguments

X	an atomic object, typically a vector. All these arguments are as in <code>tapply()</code> and are passed to <code>tapply(. .)</code> .
INDEX	list of (typically more than one) factors, each of same length as X.
FUN	the function to be applied. For the result to be simplifiable, <code>FUN()</code> must return a vector of always the same length.
...	optional arguments to FUN.

Value

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

Author(s)

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

See Also

`tapply(*, simplify=TRUE)`.

Examples

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

data(esoph)
with(esoph, {
  mima <- tapplySimpl(ncases/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

```
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

<code>y</code>	numeric; the data the density of which we want.
<code>n</code>	integer; the number of abscissa values for density evaluation (and plotting).
<code>log.bw</code>	logical; if true (default), the gui scrollbar is on a <i>log</i> bandwidth scale, otherwise, simple interval.
<code>showvalue</code>	logical; if true, the value of the current (log) bandwidth is shown on top of the scrollbar.
<code>xlim</code>	initial <code>xlim</code> for plotting, see plot.default .
<code>do.rug</code>	logical indicating if <code>rug(y)</code> should be added to each plot. This is too slow for really large sample sizes.
<code>kernels</code>	character vector of kernel names as allowable for the <code>kernels</code> argument of the standard density function.
<code>from.f</code> , <code>to.f</code>	numeric giving the left and right limit of the bandwidth scrollbar.
<code>col</code>	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())      ## 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.nm9), kernels = c("gaussian", "epanechnikov"))
}
```

u.assign0

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

Description

R doesn't 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. Its author, MM, no longer recommends them.

Usage

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

Arguments

x	character string giving the <i>name</i> 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

[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

<code>n</code>	number of boxplots.
<code>j</code>	indices of boxplots.
<code>fullrange</code>	x-coords as 'uniform' in $[0, fullrange]$; (f.=100, corresponds to Splus 3.x (x = 1,2)).

Value

a numeric vector of length `n`, with values inside $(0, M)$ where $M = fullrange$.

Author(s)

Martin Maechler

See Also

[boxplot.](#)

Examples

```
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

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

Arguments

short logical; if TRUE, no time is given.

Value

String with current date (and time).~Describe the value returned

Author(s)

Martin Maechler, ca. 1992

See Also

[u.Datumvonheute.](#)

Examples

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

u.datumdecode

Convert "Numeric" Dates

Description

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

Usage

```
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

```
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	<i>Datum und Uhrzeit (auf deutsch)</i>
-----------------	----------------------------------------

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

```
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

```
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

Convenient 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.

Usage

```
u.sys(..., intern = TRUE)
```

Arguments

... any number of strings – which will be `paste()`d together and passed to `system`.
intern logical – note that the default is *reversed* from the one in `system()`.

Author(s)

Martin Maechler

See Also

`system`, really!

Examples

```
u.sys # shows how simply the function is defined :
## Not run:
  function (... , intern = TRUE)
    system(paste(... , sep = "" ), intern = intern)

## End(Not run)
```

unif

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 **even** `n` and larger `round.dig`). Note that `c` depends on `n`.

Usage

```
unif(n, round.dig = 1 + trunc(log10(n)))
```

Arguments

`n` positive integer specifying the number of points desired.
`round.dig` 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 $Var(1, 2, \dots, n) = n(n + 1)/12$.

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.)
```

uniqueL

A Reversible Version of unique()

Description

A version of [unique](#) keeping enough information to reverse (or *invert*) to the original data.

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

wrapFormula

Enhance Formula by Wrapping each Term, e.g., by "s(.)"

Description

The main motivation for this function has been the easy construction of a “full GAM formula” from something as simple as $Y \sim ..$

The potential use is slightly more general.

Usage

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

Arguments

f the initial [formula](#); typically something like $Y \sim ..$

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.

Author(s)

Martin Maechler, May 2007.

See Also

[formula](#); [gam](#) from package **mgcv** (or also from package **gam**).

Examples

```
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

```
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 $\text{length}(x) * \text{length}(y)$ rows.

Author(s)

Martin Maechler, 26 Oct 1994.

See Also

[expand.grid](#) which didn't exist when `xy.grid` was first devised.

Examples

```
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

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

Arguments

x,y	numeric vectors of same length. Alternatively, x can be “xy” like structure.
w	numeric vector of non-negative weights – or missing which corresponds to all weights equal.
fun.mean	the mean function to use.

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

```
## simple example:  
xy.unique.x(c(1,1,1,2,4), 1:5)  
#  x y w  
# 1 1 2 3  
# 2 2 4 1  
# 3 4 5 1
```

Index

- *Topic **algebra**
 - nearcor, 43
 - posdefify, 62
- *Topic **aplot**
 - eaxis, 20
 - linesHyperb.lm, 36
 - p.arrows, 46
 - p.hboxp, 49
- *Topic **arithmetic**
 - primes, 66
- *Topic **arith**
 - digitsBase, 17
 - inv.seq, 32
 - iterate.lin.recursion, 33
 - nr.sign.chg, 46
 - polyn.eval, 61
 - roundfixS, 76
 - signi, 80
 - u.log, 94
 - unif, 95
- *Topic **array**
 - col01scale, 7
 - diagX, 17
 - empty.dimnames, 24
 - mpl, 39
 - nearcor, 43
 - pmax.sa, 60
 - posdefify, 62
 - xy.grid, 99
- *Topic **category**
 - tapplySimpl, 88
- *Topic **character**
 - repChar, 74
- *Topic **classif**
 - diagDA, 15
- *Topic **datagen**
 - QUnif, 73
- *Topic **datasets**
 - potatoes, 64
 - str_data, 82
- *Topic **debugging**
 - prt.DEBUG, 68
- *Topic **device**
 - ps.end, 69
 - ps.latex, 70
- *Topic **distribution**
 - KSd, 34
- *Topic **documentation**
 - Deprecated, 14
 - str_data, 82
- *Topic **dplot**
 - axExpr, 5
 - p.scales, 53
 - pretty10exp, 65
 - u.boxplot.x, 91
- *Topic **dynamic**
 - tkdensity, 89
- *Topic **environment**
 - u.assign0, 90
 - u.sys, 95
- *Topic **file**
 - sourceAttach, 81
- *Topic **hplot**
 - compresid2way, 8
 - cum.Vert.funkt, 9
 - ecdf.ksCI, 22
 - errbar, 25
 - hist.bxp, 29
 - mpl, 39
 - mult.fig, 40
 - n.plot, 42
 - p.datum, 47
 - p.dnorm, 48
 - p.profileTraces, 50
 - p.res.2fact, 51
 - p.res.2x, 52
 - p.tachoPlot, 54
 - p.ts, 55

- plotDS, 57
- plotStep, 59
- tkdensity, 89
- *Topic **htest**
 - f.robftest, 26
- *Topic **interface**
 - mat2tex, 38
- *Topic **iplot**
 - ellipsePoints, 23
- *Topic **iteration**
 - tapplySimpl, 88
- *Topic **manip**
 - AsciiToInt, 3
 - Duplicated, 19
 - last, 35
 - lseq, 37
 - rot2, 75
 - roundfixS, 76
 - seqXtend, 79
- *Topic **math**
 - factorize, 27
 - integrate.xy, 31
 - primes, 66
 - QUnif, 73
 - rot2, 75
- *Topic **models**
 - diagDA, 15
 - TA.plot, 86
 - wrapFormula, 98
- *Topic **multivariate**
 - QUnif, 73
- *Topic **nonlinear**
 - p.profileTraces, 50
- *Topic **nonparametric**
 - plotStep, 59
- *Topic **print**
 - empty.dimnames, 24
 - vcats, 97
- *Topic **regression**
 - hatMat, 28
 - linesHyperb.lm, 36
 - p.res.2fact, 51
 - p.res.2x, 52
 - TA.plot, 86
- *Topic **robust**
 - f.robftest, 26
 - rrange, 78
- *Topic **smooth**
 - D1D2, 10
 - D2ss, 12
 - hatMat, 28
- *Topic **ts**
 - p.ts, 55
- *Topic **univar**
 - rrange, 78
- *Topic **utilities**
 - diagX, 17
 - digitsBase, 17
 - ellipsePoints, 23
 - empty.dimnames, 24
 - integrate.xy, 31
 - inv.seq, 32
 - mat2tex, 38
 - n.code, 41
 - p.dnorm, 48
 - paste.vec, 57
 - pmax.sa, 60
 - printTable2, 67
 - quadrant, 72
 - seqXtend, 79
 - sourceAttach, 81
 - str_data, 82
 - Sys.cpuinfo, 83
 - Sys.ps, 85
 - u.assign0, 90
 - u.boxplot.x, 91
 - u.date, 91
 - u.datumdecode, 92
 - u.Datumvonheute, 93
 - u.sys, 95
 - unif, 95
 - uniqueL, 96
 - vcats, 97
 - xy.unique.x, 100
- :, 32
- abline, 86
- aov, 8, 86
- array, 25, 88
- arrows, 46, 47
- as.character, 98
- as.intBase (digitsBase), 17
- as.integer, 18
- as.integer.basedInt (digitsBase), 17
- AsciiToInt, 3
- assign, 90
- attach, 81, 82

- axis, 5, 20, 21, 65
- axis.POSIXct, 56
- axTexpr, 5, 21, 65
- axTicks, 5, 20, 21, 65

- barplot, 30, 31
- bl.string (repChar), 74
- boxplot, 31, 49, 51, 78, 79, 91
- boxplot.stats, 78

- C.Monatsname (u.Datumvonheute), 93
- C.weekday (u.Datumvonheute), 93
- C.Wochentag (u.Datumvonheute), 93
- C.Wochentagkurz (u.Datumvonheute), 93
- call, 32
- capture.and.write, 6
- cat, 68, 83, 98
- ccat (vcat), 97
- character, 4, 42, 75, 83, 93, 98
- chars8bit (AsciiToInt), 3
- code2n (n.code), 41
- col01scale, 7
- colcenter (col01scale), 7
- compresid2way, 8
- contour, 99
- cor, 43
- cum.Vert.funkt, 9
- curve, 48

- D1D2, 10, 13
- D1ss (D2ss), 12
- D1tr (D2ss), 12
- D2ss, 11, 12
- data, 82, 83
- data.frame, 43, 98
- date, 47
- DateTimeClasses, 56, 93
- dchisq, 48
- dDA (diagDA), 15
- density, 89
- Deprecated, 14
- dev.off, 69
- dev.print, 71
- dgamma, 48
- diag, 17, 63
- diagDA, 15
- diagX, 17
- digitsBase, 17
- dim, 61
- dimnames, 61, 67
- dnorm, 48
- Duplicated, 19, 97
- duplicate, 19, 20, 84, 97

- eaxis, 20
- ecdf, 23
- ecdf.ksCI, 22, 35
- eigen, 44, 62, 63
- ellipsePoints, 23
- ellipsoidhull, 24
- ellipsoidPoints, 24
- empty.dimnames, 24
- errbar, 25, 26
- eval, 32
- expand.grid, 99
- expression, 20, 32

- f.robftest, 26
- factor, 19
- factorize, 27, 28, 66
- first, 36
- fitted.rnls (Deprecated), 14
- fivenum, 79
- format, 57, 98
- formula, 98, 99
- ftable, 67
- function, 83

- gam, 98, 99
- get, 90
- getOption, 71
- glm, 86

- hatMat, 28
- hetcor, 43, 44
- hist, 30, 31
- hist.bxp, 29, 49

- ichar (AsciiToInt), 3
- integer, 3, 18, 42, 66
- integrate, 32
- integrate.xy, 31
- interaction.plot, 9
- inv.seq, 32
- invisible, 83
- iterate.lin.recursion, 33

- KSd, 23, 34

- last, 35
- latex, 38
- layout, 41
- lda, 16
- length, 4, 79
- lines, 36
- linesHyperb.lm, 36
- list, 4, 27, 41, 43, 44, 83
- list2mat (Deprecated), 14
- lm, 36, 86, 98
- logical, 20, 66, 83
- lseq, 37

- make.names, 84
- margin2table (printTable2), 67
- mat2tex, 38
- match, 19, 20
- matplot, 39, 40
- matrix, 18
- max, 66
- month.name, 94
- mpl, 39
- mtext, 41, 47
- mult.fig, 40, 56

- n.code, 41
- n.plot, 42, 87
- NA, 78
- names, 4
- nchar, 75
- nearcor, 43
- nearPD, 44, 63
- next, 66
- nls, 50
- nr.sign.chg, 46
- numeric, 37

- options, 38, 68

- p.arrows, 46
- p.datum, 10, 47, 94
- p.dchisq (p.dnorm), 48
- p.dgamma (p.dnorm), 48
- p.dnorm, 48
- p.hboxp, 49
- p.m (mpl), 39
- p.pllines (Deprecated), 14
- p.profileTraces, 50
- p.res.2fact, 51, 52
- p.res.2x, 51, 52
- p.scales, 53
- p.tachoPlot, 54
- p.ts, 55
- par, 21, 25, 30, 40, 41, 43, 50, 53, 54, 56, 58, 59, 71
- paste, 75, 95, 97, 98
- paste.vec, 57
- pdf, 69–71
- pdf.do, 69, 71
- pdf.do (ps.latex), 70
- pdf.end, 70
- pdf.end (ps.end), 69
- pdf.latex, 69
- pdf.latex (ps.latex), 70
- persp, 99
- pl.ds (plotDS), 57
- plot, 42, 43, 54, 56, 58–60
- plot.default, 25, 43, 51, 52, 54, 56, 58, 89
- plot.ecdf, 9, 10, 60
- plot.lm, 51, 52, 87
- plot.mts, 40
- plot.stepfun, 22, 23, 60
- plot.ts, 55, 56
- plotDS, 57, 80
- plotmath, 20
- plotStep, 10, 59
- pmax, 60
- pmax.sa, 60
- pmin, 60, 61
- pmin.sa (pmax.sa), 60
- polyn.eval, 61
- posdefify, 44, 62
- postscript, 69–71
- potatoes, 64
- predict, 15
- predict.dDA (diagDA), 15
- predict.lm, 36, 37
- predict.polynomial, 62
- predict.rnls (Deprecated), 14
- pretty10exp, 5, 20, 21, 65
- primes, 27, 28, 66
- print, 15, 18
- print.basedInt (digitsBase), 17
- print.dDA (diagDA), 15
- print.default, 67
- print.margin2table (printTable2), 67
- print.rnls (Deprecated), 14

- printTable2, 67
- proc.time, 85
- profile, 50
- profile.nls, 50
- prt.DEBUG, 68
- ps.do, 69, 71
- ps.do(ps.latex), 70
- ps.end, 69, 71
- ps.latex, 70
- ps.options, 71

- qda, 16
- quadrant, 72
- QUnif, 73

- range, 78, 79
- repChar, 74
- residuals.rnls (Deprecated), 14
- rle, 32
- rlm, 26, 27
- rnls (Deprecated), 14
- rot2, 75
- round, 77, 81
- roundfixS, 76, 80
- rrange, 78
- rug, 31, 56, 89
- runif, 96

- scale, 7
- scan, 85
- scat1d, 31
- segments, 47, 52, 60
- seq, 34, 37, 79, 80
- seqXtend, 58, 79
- sHalton (QUnif), 73
- signi, 80
- signif, 81
- smooth.spline, 10–13, 29, 100
- source, 82
- sourceAttach, 81
- split, 19
- stepfun, 23
- str, 83
- str_data, 82
- strcodes (AsciiToInt), 3
- summary, 26
- summary.aov, 27
- summary.rlm, 26
- summary.rnls (Deprecated), 14

- symbols, 55
- Sys.cpuinfo, 83
- Sys.getpid, 85
- Sys.info, 85
- Sys.meminfo (Sys.cpuinfo), 83
- Sys.MIPS (Sys.cpuinfo), 83
- Sys.procinfo (Sys.cpuinfo), 83
- Sys.ps, 84, 85
- Sys.setlocale, 4
- Sys.sizes (Sys.ps), 85
- sys.source, 81, 82
- system, 85, 95

- TA.plot, 42, 51, 52, 86
- table, 67
- tail, 35
- tapply, 19, 88
- tapplySimpl, 88
- text, 42, 43
- title, 8, 22, 52
- tkdensity, 89
- TRUE, 68
- turnogram, 35, 36

- u.assign0, 90
- u.boxplot.x, 51, 91
- u.date, 47, 91, 93, 94
- u.datumdecode, 92
- u.Datumvonheute, 47, 92, 93
- u.get0 (u.assign0), 90
- u.log, 94
- u.sys, 95
- unif, 95
- unique, 79, 96, 97
- uniqueL, 20, 96
- unname, 25

- vcat, 97

- wrapFormula, 98
- writeLines, 6

- x11, 69
- xy.coords, 43, 58, 72
- xy.grid, 99
- xy.unique.x, 100