Package ‘sfsmisc’

April 25, 2024

Title Utilities from 'Seminar fuer Statistik' ETH Zurich

Version 1.1-18

VersionNote Last CRAN: 1.1-17 on 2024-02-01

Date 2024-04-25

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

Description Useful utilities ['goodies'] from Seminar fuer Statistik ETH Zurich, some of which were ported from S-plus in the 1990s.
For graphics, have pretty (Log-scale) axes eaxis(), an enhanced Tukey-Anscombe plot, combining histogram and boxplot, 2d-residual plots, a 'tachoPlot()', pretty arrows, etc.
For robustness, have a robust F test and robust range().
For system support, notably on Linux, provides 'Sys.*( )' functions with more access to system and CPU information.
Finally, miscellaneous utilities such as simple efficient prime numbers, integer codes, Duplicated(), toLatex.numeric() and is.whole().

Depends R (>= 3.3.0)

Imports grDevices, utils, stats, tools

Suggests datasets, tcltk, cluster, lattice, MASS, Matrix, nlme, lokern, Rmpfr, gmp

Enhances mgcv, rpart, nor1mix, polycor, sm, tikzDevice, e1071, Hmisc, pastecs, polynom, robustbase

EnhancesNote 2nd line: packages mentioned in Rd xrefs

Encoding UTF-8

ByteCompile yes

License GPL (>= 2)

URL https://github.com/mmaechler/sfsmisc

BugReports https://github.com/mmaechler/sfsmisc/issues

NeedsCompilation no
Author  Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Werner Stahel [ctb] (Functions: comprisid2way(), f.robftest(), last(), p.scales(), p.dnorm()), Andreas Ruckstuhl [ctb] (Functions: p.arrows(), p.profileTraces(), p.res.2x()), Christian Keller [ctb] (Functions: histBxp(), p.tachoPlot()), Kjetil Halvorsen [ctb] (Functions: KSd(), ecdf.ksCI()), Alain Hauser [ctb] (Functions: cairoSwd(), is.whole(), toLatex.numeric(*)), Christoph Buser [ctb] (to function Duplicated()), Lorenz Gygax [ctb] (to function p.res.2fact()), Bill Venables [ctb] (Functions: empty.dimnames(), primes()), Tony Plate [ctb] (to inv.seq()), Isabelle Flückiger [ctb], Marcel Wolbers [ctb], Markus Keller [ctb], Sandrine Dudoit [ctb], Jane Fridlyand [ctb], Greg Snow [ctb] (to loessDemo()), Henrik Aa. Nielsen [ctb] (to loessDemo()), Vincent Carey [ctb], Ben Bolker [ctb], Philippe Grosjean [ctb], Frédéric Ibanez [ctb], Caterina Savi [ctb], Charles Geyer [ctb], Jens Oehlschlägel [ctb]

Repository  CRAN

Date/Publication  2024-04-25 15:50:02 UTC

R topics documented:

AsciiToInt .................................................. 5
axTexpr .................................................... 6
cairoSwd ................................................... 8
capture.and.write ......................................... 9
col01scale .................................................. 10
comprisid2way .............................................. 11
cum.Vert.funkt ............................................ 12
D1D2 ........................................................ 13
D2ss .......................................................... 15
Deprecated .................................................. 17
diagDA ....................................................... 17
diagX ........................................................ 19
digitsBase ................................................... 20
Duplicated .................................................. 22
eaxis .......................................................... 23
<table>
<thead>
<tr>
<th>Topics Documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecdf.ksCI</td>
</tr>
<tr>
<td>ellipsePoints</td>
</tr>
<tr>
<td>empty.dinames</td>
</tr>
<tr>
<td>errbar</td>
</tr>
<tr>
<td>f.robftest</td>
</tr>
<tr>
<td>factorize</td>
</tr>
<tr>
<td>funEnv</td>
</tr>
<tr>
<td>hatMat</td>
</tr>
<tr>
<td>helppdf</td>
</tr>
<tr>
<td>histBxp</td>
</tr>
<tr>
<td>integrate.xy</td>
</tr>
<tr>
<td>inv.seq</td>
</tr>
<tr>
<td>is.whole</td>
</tr>
<tr>
<td>iterate.lin.recursion</td>
</tr>
<tr>
<td>KSd</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>linesHyperb.Lm</td>
</tr>
<tr>
<td>list_named</td>
</tr>
<tr>
<td>loessDemo</td>
</tr>
<tr>
<td>lseq</td>
</tr>
<tr>
<td>mat2tex</td>
</tr>
<tr>
<td>missingCh</td>
</tr>
<tr>
<td>mpl</td>
</tr>
<tr>
<td>mult.fig</td>
</tr>
<tr>
<td>n.code</td>
</tr>
<tr>
<td>n.plot</td>
</tr>
<tr>
<td>nearcor</td>
</tr>
<tr>
<td>nr.sign.chg</td>
</tr>
<tr>
<td>p.arrows</td>
</tr>
<tr>
<td>p.datum</td>
</tr>
<tr>
<td>p.dnorm</td>
</tr>
<tr>
<td>p.hboxp</td>
</tr>
<tr>
<td>p.profileTraces</td>
</tr>
<tr>
<td>p.res.2fact</td>
</tr>
<tr>
<td>p.res.2x</td>
</tr>
<tr>
<td>p.scales</td>
</tr>
<tr>
<td>p.tachoPlot</td>
</tr>
<tr>
<td>p.ts</td>
</tr>
<tr>
<td>paste.vec</td>
</tr>
<tr>
<td>pkgDesc</td>
</tr>
<tr>
<td>pkgLibs</td>
</tr>
<tr>
<td>plotDS</td>
</tr>
<tr>
<td>plotStep</td>
</tr>
<tr>
<td>polyn.eval</td>
</tr>
<tr>
<td>posdefify</td>
</tr>
<tr>
<td>potatoes</td>
</tr>
<tr>
<td>pretty10exp</td>
</tr>
<tr>
<td>primes</td>
</tr>
</tbody>
</table>
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. Note that it (and hence strcodes()) depends on the locale, see Sys.getlocale().

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Martin Maechler, partly in 1991 for S-plus
Examples

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

## may only work in ISO-latin1 locale (not in UTF-8):
```
try( strcodes(c(a = "ABC", ch="1234", place = "Zürich")))
```

## in "latin-1" gives {otherwise should give NA instead of 252}:
```
## Not run:
$a
[1] 65 66 67
$ch
[1] 49 50 51 52
$place
[1] 90 252 114 105 99 104
## End(Not run)
```

myloc <- Sys.getlocale()

```r
if(.Platform $ OS.type == "unix") withAutoprint({ # should work' here
  try( Sys.setlocale(locale = "de_CH") ) # "try": just in case
  strcodes(c(a = "ABC", ch="1234", place = "Zürich"))) # no NA hopefully
  AsciiToInt(chars8bit())) # -> 1:255 {if setting latin1 succeeded above}
```

```r
chars8bit(97:140)
try( Sys.setlocale(locale = "de_CH.utf-8") ) # "try": just in case
chars8bit(97:140) # typically looks different than above
```

```r
## Resetting to original locale . works "mostly”:
  lapply(strsplit(strsplit(myloc, ";")[[1]], "="),
    function(cc) try(Sys.setlocale(cc[1], cc[2]))) -> .scratch
  Sys.getlocale() == myloc # TRUE if we have succeeded to reset it
```

---

### 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**

```r
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× should be dropped from the resulting expressions.

Details

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

Value

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

Author(s)

Martin Maechler

See Also

`pretty10exp`, `eaxis`, `axis`, `axTicks`.

Examples

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

## For horizontal y-axis labels, need more space:
op <- par(mar = .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= axTexpr(1, aX))

## Now an "engineer's version" ( more ticks; only label "10 ^ k" ) :
axp <- par("xaxp") #-> powers of 10 *inside* 'usr'
axp[3] <- 1 # such that only 10^*. are labeled
ax <- axTicks(1, axp = axp)
xu <- 10 ^ par("usr")[1:2]
e10 <- c(-1,1) + round(log10(axp[1:2])) ## exponents of 10 *outside* 'usr'
v <- c(outer(1:9, e10[1], e10[2]), function(x,E) x * 10 ^ E))
v <- v[xu[1] <= v & v <= xu[2]]
```
cairoSwd

Cairo PDF Graphics Device useful for Sweave

Description

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

Usage

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

Arguments

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

Note

Sweave devices need to have an argument list as above.

Usage in a Sweave chunk:

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

Author(s)

Alain Hauser

See Also

pdf, Cairo_pdf, Sweave.
**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 (*RweaveLatex*).

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

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

**Usage**

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

**Arguments**

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

**Value**

return value of `capture.output(EXPR)`.

**Author(s)**

Martin Maechler, ETH Zurich

**See Also**

`head`, `tail`
Examples

```r
x <- seq(0, 10, by = .1)
## for matrix, dataframe, .. first lines include a header line:
capture.and.write( cbind(x, log1p(exp(x))), first = 5)

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

---

### 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**

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

**Arguments**

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

**Value**

a matrix with the same attributes as the input `mat`.

**Author(s)**

Martin Maechler

**See Also**

The standard R function `scale()`.

**Examples**

```r
## See the simple function definitions:
colcenter ## simply one line
col01scale# almost as simple
```
**compresid2way**  

*Plot Components + Residuals for Two Factors*

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

**Value**

Invisibly, a list with components

- `compy` data frame containing the component effects of the two factors, and combined effects plus residual
- `coef` coefficients: Intercept and effects of the factors
Author(s)

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

References


See Also

interaction.plot

Examples

```r
N <- c(0,1,0,1,1,0,0,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1,1,0,0,1,1,0,1,1,0,0,1,1,0,1,1,0,1,1,0)
P <- c(1,0,0,0,1,1,1,1,0,0,0,0,1,0,1,1,1,0,0,1,0,1,0,1,1,1,0,0,1,1,0,1,1,0)
K <- c(1,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0)
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)
sample(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 = "", ...)
D1D2

Arguments

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

Value

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

Author(s)

Martin Maechler et al.

See Also

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

Examples

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

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

---

D1D2

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

Description

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

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

Usage

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

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

Details

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

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

Value

A list with several components,

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

Author(s)

Martin Maechler, in 1992 (for S).

See Also

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

Examples

```r
set.seed(8840)
x <- runif(100, 0,10)
y <- sin(x) + rnorm(100)/4

op <- par(mfrow = c(2,1))
```
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 \( \text{smooth.spline} \) for the first derivative \( f'(t) \) and then applies the same to the predicted values \( \hat{f}'(t_i) \) (where \( t_i \) are the values of \( xout \)) to find \( \hat{f}''(t_i) \).

Usage

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

Arguments

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

Value

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

`D2ss()` returns a list with components

- **x**: the abscissae values (= `xout`) at which the derivative(s) are evaluated.
- **y**: estimated values of \( f''(x_i) \).
- **spl.spar**: numeric vector of length 2, contain the `spar` arguments to the two `smooth.spline` calls.
- **spar.offset**: as specified on input (maybe repeated 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

```r
## First Derivative --- spar.off = 0 ok "asymptotically" (?) 
set.seed(330)
mult.fig(12)
for(i in 1:12) {
  x <- runif(500, 0,10); y <- sin(x) + rnorm(500)/4
  f1 <- D1ss(x=x, y=y, spar.off=0.0)
  plot(x,f1, ylim = range(c(-1,1,f1)))
  curve(cos(x), col=3, add= TRUE)
}
set.seed(8840)
x <- runif(100, 0,10)
y <- sin(x) + rnorm(100)/4
op <- par(mfrow = c(2,1))
plot(x,y)
lines(ss <- smooth.spline(x,y), col = 4)
str(ss[c("df", "spar")])
xx <- seq(0,10, len=201)
plot(xx, -sin(xx), type = 'l', ylim = c(-1.5,1.5))
title(expression("Estimating f"^"\prime\prime" : " * 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 '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

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

#### Arguments

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

#### Details

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

### diagDA

#### Diagonal Discriminant Analysis

#### Description

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

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

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

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

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

object  
object of class dDA.

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

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

...  
further arguments passed to and from methods.

Value

dDA() returns an object of class dDA for which there are print and predict methods. The latter returns the same as 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


See Also

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

```r
## 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)
(cl1P <- pam(d1,k=4)$cluster) # 4 surprising clusters
with(d1, points(x+0.5, y, col = cl1P, pch =cl1P))

i1 <- c(1,3,5,6)
t1 <- d1[-i1,]
cl1. <- c(1,2,1,2,1,3)
cl1 <- c(2,2,1,1,1,3)
plot(t1, cex=2, col = cl1, pch = 20+cl1)
(dd. <- diagDA(t1, cl1., ts = d1[ i1,]))# ok
(dd <- diagDA(t1, cl1 ) , ts = d1[ i1,]))# ok, too!
points(d1[ i1,], pch = 10, cex=3, col = dd)

## use new fit + predict instead :
(r1 <- dDA(t1, cl1))
(r1. <- dDA(t1, cl1.))
stopifnot(dd == predict(r1, new = d1[ i1,]),
dd.== predict(r1., new = d1[ i1,]))
plot(t1, cex=2, col = cl1, bg = cl1, pch = 20+cl1,
     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)
x <- matrix(c(3,7,5,7), ncol=4)
predict(m4.1, x)x## gave integer(0) previously
predict(m4.2, x)
```

---

### diagX

#### The “Other” Diagonal Matrix

**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**

\texttt{diag}.

**Examples**

\begin{verbatim}
  diagX(4)
  for(m in 1:5)
      stopifnot(identical(diagX(m), diag(m)[, m:1, drop = FALSE]))
\end{verbatim}

---

### digitsBase

**Digit/Bit Representation of Integers in any Base**

**Description**

Integer number representations in other Bases.

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

**Usage**

\begin{verbatim}
  digitsBase(x, base = 2, ndigits = 1 + floor(1e-9 + log(max(x,1), base)))
  ## S3 method for class 'basedInt'
  as.integer(x, ...)
  ## S3 method for class 'basedInt'
  print(x, ...)

  as.intBase(x, base = 2)
  bi2int(xlist, base)
\end{verbatim}
digitsBase

Arguments

x For digitsBase(): non-negative integer (vector) whose base base digits are wanted.
For as.intBase():
a list of numeric vectors, a character vector, or an integer matrix as returned by
digitsBase(), representing digits in base base.
base integer, at least 2 specifying the base for representation.
ndigits number of bits/digits to use.
... potential further arguments passed to methods, notably print.
xlist a list of integer vectors with entries typically in 0:(base-1), such as resulting from
digitsBase().

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.
biz2int() is the low-level workhorse of as.intBase().

Note

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

Author(s)

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

Examples

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

## This may be handy for just one number (and default decimal):
digits <- function(n, base = 10) as.vector(digitsBase(n, base = base))
digits(128882734) # 1 2 8 9 8 2 7 3 4
digits(128, base = 8) # 2 0 0

## one way of pretty printing (base <= 10!)
b2ch <- function(db)
  noquote(gsub("^0+(.{$}.)$"," \1",
     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])
Duplicated <- hexdig[1 + M]; dim(cM) <- dim(M)

b2ch(cM) #-> 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 ... 6A

## IP (Internet Protocol) numbers coding: <n>.<n>.<n>.<n> <--> longinteger

ip_ntoa <- function(n)
  apply(digitsBase(n, base = 256), 2, paste, collapse=".")

ip_ntoa(2130706430 + (0:9)) # "126.255.255.254" ... "127.0.0.7"

## and the inverse:

ip_aton <- function(a)
  bi2int(lapply(strsplit(a, ".", fixed=TRUE), as.integer), 256)

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

stopifnot( n == ip_aton(ip_ntoa(n)),
  ip == ip_ntoa(ip_aton(ip)))

## 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.
eaxis

Description

An extended \texttt{axis()} function which labels more prettily, in particular for log-scale axes.

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

Usage

\begin{verbatim}
eaxis(side, at = if(log) axTicks(side, axp=axp, log=log, nintLog=nintLog)
  else axTicks(side, axp=axp, log=log),
  labels = NULL, log = NULL,
  use.expr = log || format.info(as.numeric(at), digits=7)[3] > 0,
  f.smalltcl = 3/5, at.small = NULL, small.mult = NULL, equidist.at.tol = 0.002,
  small.args = list(),
  draw.between.ticks = TRUE, between.max = 4,
  outer.at = TRUE, drop.1 = TRUE, sub10 = FALSE, las = 1,
  nintLog = max(12, par("lab")[2 - is.x]),
  axp = NULL, n.axp = NULL, max.at = Inf,
  lab.type = "plotmath", lab.sep = "cdot",
  ...)
\end{verbatim}
Arguments

side  integer in 1:4, specifying side of axis.
at   numeric vector of ("normalsized") tick locations; by default axTicks(side, ..), i.e., the same as axis() would use.
labels NULL (default), logical, character or expression, as in axis(); in addition, if NA, labels = TRUE is passed to axis(), i.e. pretty10exp is not used. Use FALSE to suppress any labeling.
log   logical or NULL specifying if log-scale should be used; the default depends on the current plot’s axis.
use.expr logical specifying if pretty10exp(.) should be used for constructing labels when they are NULL. The default is typically good enough, but you may occasionally force use.expr = TRUE.
f.smalltcl factor specifying the lengths of the small ticks in proportion to the normalsized, labeled ticks.
at.small locations of small ticks; the default, NULL, uses small.mult and constructs “smart” locations.
small.mult positive integer (or NULL), used when at.small is NULL to indicate which multiples of at (typically axTicks()) should be used as “small ticks”. The default NULL will use 9 in the log case and a number in 2:5 otherwise.
equidist.at.tol a small positive number, a tolerance to be used for checking equidistant at values. Used to be hardwired at .001 which was seen to be too small; increase it when necessary.
small.args optional list of further arguments to the (second) axis() call which draws the small ticks.
draw.between.ticks (only if log is true): logical indicating that possible (non-small) ticks between the labeled (via at) ones should be drawn as well (and possibly also used for at.small construction), see also between.max.
between.max (only if log and draw.between.ticks are true): integer indicating ticks should be drawn (approximately) between the labeled ones.
outer.at logical specifying that at.small should also be constructed outside the at range, but still inside the corresponding par("usr").
drop.1 logical specifying if 1× should be dropped from labels, passed to pretty10exp().
sub10 logical, integer (of length 1 or 2) or "10", indicating if some 10^k should be simplified to "traditional" formats, see pretty10exp.
nintLog only used in R > 2.13.x, when log is true: approximate (lower bound on) number of intervals for log scaling.
axp to be passed to axTicks() if at is not specified.
n.axp to be set to axp[3] when axp and at are not specified, in order to tweak the number of (non-small) tick marks produced from axTicks(..), notably when log is true, set n.axp to 1, 2, or 3:
1: will produce tick marks at 10^j for integer j,
**eaxis**

2: gives marks $k10^j$ with $k \in \{1, 5\}$,  
3: gives marks $k10^j$ with $k \in \{1, 2, 5\}$  
see 'xaxp' on the `par` help page.

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

**las, ...**  
arguments passed to (the first) `axis` call. Note that the default `las = 1` differs from `axis`’s default `las = 0`.

**lab.type**  
string, passed to `pretty10exp` to choose between default `plotmath` or LaTeX label format.

**lab.sep**  
separator between mantissa and exponent for LaTeX labels, see `pretty10exp`.

**Author(s)**  
Martin Maechler

**See Also**  
`axis`, `axTicks`, `axTexpr`, `pretty10exp`.

**Examples**

```r
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(3, n.axp = 1)# -> xaxp[3] = 1: only 10^j (main) ticks

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

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

## more labels than default:
op <- par(lab=c(10,5,7))
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis() # maybe (depending on your canvas), there are too many,
## in that case, maybe use
plot(x, sin(x)/x, type = "l", xaxt = "n")
eaxis(1, axTicks(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)

### Answering R-help "How do I show real values on a log10 histogram", 26 Mar 2013
## the data:
set.seed(1); summary(x <- rlnorm(100, m = 2, sdl = 3))
## the plot (w/o x-axis):
r <- hist(log10(x), xaxt = "n", xlab = "x [log scale]")
## the nice axis:
axt <- axTicks(1)
eaxis(1, at = axt, labels = pretty10exp(10^axt, drop.1=TRUE))
## Additionally demo'ing 'sub10' options:
plot(r, xaxt="n")
eaxis(1, at = axt, labels = pretty10exp(10^axt, drop.1=TRUE, sub10 = 2))
## or
plot(r, xaxt="n")
eaxis(1, at = axt, labels = pretty10exp(10^axt, drop.1=TRUE, sub10 = "10"))
## or
plot(r, xaxt="n")
eaxis(1, at = axt, labels = pretty10exp(10^axt, drop.1=TRUE, sub10 = c(-2, 2)))
```

---

**ecdf.ksCI**

*Plot Empirical Distribution Function With 95% Confidence Curves*

**Description**

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

**Usage**

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

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

**Value**

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

**Note**

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

**Author(s)**

Kjetil Halvorsen

**References**

Bickel and Doksum, see `Ksd`.

**See Also**

`ecdf` and `plot.stepfun` in standard R.

**Examples**

```r
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**

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

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

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

- **loc**
  center (LOCation) of the ellipse.

- **n**
  number of points to generate.

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

**Value**

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

**Author(s)**

Martin Maechler, March 2002.

**See Also**

the ellipse package and ellipsoidhull and ellipsoidPoints in the cluster package.

**Examples**

```r
## 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(TRUE)) {
    ## Movie : rotating ellipse :
    nTurns <- 4 # #{full 360 deg turns}
    for(al in 1:(nTurns*360)) {
        ep <- ellipsePoints(3,6, alpha=al, loc = c(5,2))
        plot(ep, type="l", xlim=c(-1,11), ylim=c(-4,8),
             asp=1, axes = FALSE, xlab="", ylab="")
    }
}
```
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, i.e., as special case a matrix.

Value

Returns a with its dimnames replaced by empty character strings.

Author(s)

Bill Venables / Martin Maechler, Sept 1993.

See Also

unname removes the dimnames.

Examples

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

erbar(x, y, yplus, yminus, cap = 0.015,
ylim = range(y, yplus, yminus),
xlab = deparse(substitute(x)),
ylab = deparse(substitute(y)), ...)
Robust F-Test: Wald test for multiple coefficients of rlm() Object

Description

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

Usage

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

Arguments

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

Details

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

Value

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

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

Author(s)

Werner Stahel, July 2000; updates by Martin Maechler.

References

FIXME — Need some here!

See Also

`rlm`, `summary.aov`, etc.
Examples

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

##  Prime Factorization of Integers

**Description**

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

**Usage**

```r
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**

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

List-like Environment of Functions (and More)

Description

Construct a “list”, really an environment typically of functions and optionally other R objects, where the functions and formulas all share the same environment. Consequently, the functions may call each other.

On technical level, this is just a simple wrapper around list2env().

Usage

funEnv(..., envir = NULL, parent = parent.frame(),
hash = (...length() > 100), size = max(29L, ...length()))

Arguments

... an arbitrary named “list” of R objects, typically including several functions.
envir an environment or NULL.
parent (for the case envir = NULL): a parent frame aka enclosing environment, see new.env and list2env.
hash, size (for the case envir = NULL): hash a logical indicating if the created environment should use hashing, and (size) the hash size, see list2env.

Value

an environment, say E, containing the objects from ... (plus those in envir), and all function objects’ environment() is E.

Author(s)

Martin Maechler

See Also

list2env, environment

Examples

ee <- funEnv(f = function(x) g(2*(x+1)),
g = function(y) hh(y+1),
hh = function(u) u^2,
info = “Some Information (not a function)”
ls(ee) # here the same as names(ee)
# Check that it works: i.e., that “f sees g” and “g sees hh”:
stopifnot(all.equal(ee$f(pi), (2*pi+3)^2))
ee$f(0:4) # [1] 9 25 49 81 121
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 \( \text{trace} = \text{FALSE} \) as per default) or a number, \( tr(H) \), the trace of \( H \), i.e., \( \sum_i H_{ii} \).

Note that \( \text{dim}(H) = c(n, n) \) where \( n \leftarrow \text{length}(x) \) also in the case where some \( x \) values are duplicated (aka ties).

Author(s)

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

References


See Also

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

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

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

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

## ksmooth() :
Hk <- hatMat(x1, pr = pksm, bandwidth = 2)
cat(sprintf("df = %.2f\n", sum(diag(Hk))))
image(Hk)
Matrix::printSpMatrix(as(round(Hk, 2), "sparseMatrix"))
```

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

---

**helppdf**

**help() Type="pdf" and View It**

**Description**

Utility to view PDF-rendered help pages; particularly useful in case they contain mathematical formulas or otherwise sophisticated formats.

**Usage**

```r
helppdf(topic, viewer = getOption("pdfviewer"), quiet = !interactive(), ...)
```

**Arguments**

- `topic` the topic, passed to `help()`.
- `viewer` a pdf viewer; the default is typically what you want interactively.
- `quiet` logical indicating that nothing should be printed to the console and the result should be returned as `invisible()`.
- `...` further optional arguments passed to `help()`.
Value
Returns the full path of the pdf file produced.

Author(s)
Martin Maechler

See Also
help, system.

Examples
if(interactive()) {
  ## Both calls work :
  helppdf(Normal)
  helppdf("NegBinomial")
} else if(.Platform$OS.type != "windows") { # batch mode (Windows often too slow for this)
  od <- setwd(tempdir())
  ff <- helppdf(Normal, viewer=NULL)
  stopifnot(file.exists(ff)) ; print(ff)
  setwd(od)# revert to previous dir.
}

histBxp

Plot a Histogram and a Boxplot

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

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

Arguments
x
  numeric vector of data for histogram. Missing values (NAs) are allowed.
nclass
  recommendation for the number of classes (i.e., bars) the histogram should have.
  The default is a number proportional to the logarithm of the length of x.
breaks
  vector of the break points for the bars of the histogram. The count in the i-th bar is sum(breaks[i] < x <= breaks[i+1]) except that if include.lowest is TRUE (the default), the first bar also includes points equal to breaks[1]. If omitted, evenly-spaced break points are determined from nclass and the extremes of the data.
probability logical flag: if TRUE, the histogram will be scaled as a probability density; the sum of the bar heights times bar widths will equal 1. If FALSE, the heights of the bars will be counts.

include.lowest If TRUE (the default), the lowest bar will include data points equal to the lowest break, otherwise it will act like the other bars (see the description of the breaks argument).

xlab character or expression for x axis labeling.

... additional arguments to barplot. The hist function uses the function barplot to do the actual plotting; consequently, arguments to the barplot function that control shading, etc., can also be given to hist. See the barplot documentation for arguments angle, density, col, and inside. Do not use the space or histo arguments.

width width of the box relative to the height of the histogram. DEFAULT is 0.2.

boxcol color of filled box. The default is 3.

medcol the color of the median line. The special value, NA, indicates the current plotting color (par("col")). The default is 2. If boxcol=0 and medcol is not explicitly specified this is set to the current plotting color (par("col")).

medlwd median line width. The special value NA, is used to indicate the current line width (par("lwd")). The default is 5.

whisklty whisker line type. The special value NA indicates the current line type (par("lty")). The default is 2 (dotted line).

staplelty staple (whisker end cap) line type. The special value NA indicates the current line type (par("lty")). The default is 1 (solid line).

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

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

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

Author(s) S-Plus: Markus Keller, Christian Keller; port to R in 1990’s: Martin Mächler.

See Also hist, barplot, boxplot, rug and scat1d in the Hmisc package.
Examples

```r
lab <- "50 samples from a t distribution with 5 d.f."
mult.fig(2*3, main = "Hist() + Rug() and histBxp(*)")
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..
  histBxp(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**

```r
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 \(\text{min}(x)\) and \(\text{max}(x)\) respectively.
- `use.spline` logical; if TRUE use an interpolating spline.
- `xtol` tolerance factor, typically around \(\sqrt{\text{.Machine$double.eps}}\) ......(fixme)....

**Details**

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

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

**Value**

the approximate integral.

**Author(s)**

Martin Maechler, May 1994 (for S).

**See Also**

`integrate` for numerical integration of *functions.*
inv.seq

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 \( \text{mapply}() \) bug in this example:

\[
\text{ii} \leftarrow c(1:3, 6:9, 11:16)
\]

\[
\text{stopifnot}(\text{identical(ii, eval(\text{inv.seq(ii)})))}
\]

---

### is.whole

Test Whether a Vector or Array Consists of Whole Numbers

**Description**

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

**Usage**

\[
is\text{.whole}(x, \text{tolerance} = \sqrt{\text{.Machine}\$\text{double.\text{eps}}})
\]

**Arguments**

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

**Value**

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

**Author(s)**

Alain Hauser <alain@huschhus.ch>

**See Also**

`is.integer`

**Examples**

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

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

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

---  

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

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

**Note**  
Depending on the zeroes of the characteristic polynomial of `coeff`, there are three cases, of convergence, oscillation and divergence.

**Author(s)**  
Martin Maechler

**See Also**  
`seq` can be regarded as a trivial special case.
Examples

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

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

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

---

**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**

```r
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**

last

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

x
length.out
na.rm

any vector.
integer indicating how many element are desired. If positive, return the length.out last elements of x; if negative, the last length.out elements are dropped.
logical indicating if the last non-missing value (if any) shall be returned. By default (it is FALSE and) the last elements (whatever its values) are returned.

Value

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

Note

This function may eventually be deprecated for the standard R function tail().

Useful for the turnogram() function in package pastecs.

Author(s)

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

first, turnogram

Examples

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

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

---

**linesHyperb.lm**

Plot Confidence or Prediction Hyperbolas around a Regression Line

Description

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

Usage

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

Arguments

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

Note

With `predict.lm(*, interval=)` is available, this function `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")

Description

A version of `list(...)`, but with “automatically” named list components.

Usage

`list_(...)`

Arguments

`...` components to make up the resulting `list`. Their variable names (or unevaluated expressions in the call) will become the `names(.)` of the result.

Details

The names are extracted from `sys.call()`, and the function is written to be `fast` (rather than easy to ready for the uninitiated ;-)

Value

a `list` with the components in the arguments with `names` taken from their call to `list_(..)`.

Author(s)

Martin Maechler

See Also

`list`, `names`
Examples

```r
a <- 1:4; lett <- letters[1:9]; CH <- "Suisse"
all.equal(list(a, lett),
  list_(a, lett)) # "names for current but not for target"
str(list(a, lett, CH)) # [[1]], [[2]], .. (no names)
str(list_(a, lett, CH))# $a $lett ..
stopifnot(identical(
  list(a, lett, CH),
  unname(L <- list_(a, lett, CH))),
  is.list(L), names(L) == c("a", "lett", "CH"),
  identical(lett, L$lett)## etc
)
```

## The function is currently defined as
function (...) `names<-`(`list(...)`, vapply(sys.call()[-1L], as.character, ""))

---

**loessDemo**

*Graphical Interactive Demo of loess()*

**Description**

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

**Usage**

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

**Arguments**

- **x, y**
  - numeric vectors of the same length; the demo is about *loess*(y ~ x).
- **span**
  - the smoothing parameter $\alpha$.
- **degree**
  - the degree of the polynomials to be used; must be in 0, 1, 2.
- **family**
  - if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey's biweight function. Can be abbreviated.
- **nearest**
  - logical indicating how $x_0$ should be determined, the value at which $\hat{f}(x_0)$ is computed. If nearest is true, the closest data value is taken.
- **nout**
  - the number of points at which to evaluate, i.e. determining $u_i, i = 1, 2, \ldots, nout$, at which $f(u_i)$ is computed.
- **xlim**
  - x-range; to extend or determine (iff strictlim is true) the x-range for plotting.
- **ylim**
  - y-range; to extend or determine (iff strictlim is true) the y-range for plotting.
loessDemo

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

verbose logical ......

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

pch plotting character, see points.

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

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

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

Author(s)

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

See Also

loess.

Examples

if(dev.interactive()) {

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

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

}
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 \texttt{R} matrix (like object) into a \LaTeX\ table, using \texttt{\begin\{tabular\} \ldots\n
Usage
mat2tex(x, file = "mat.tex", envir = "tabular",
         nam.center = "1", col.center = "c",
         append = TRUE, digits = 3, title)
Arguments

- **x**
  - a matrix

- **file**
  - names the file to which LaTeX commands should be written

- **envir**
  - a string, the LaTeX environment name; default is "tabular"; useful maybe "array", or other versions of tabular environments.

- **nam.center**
  - character specifying row names should be center; default "l".

- **col.center**
  - character (vector) specifying how the columns should be centered; must have values from c("l", "c", "r"); defaults to "c".

- **append**
  - logical; if FALSE, will destroy the file file before writing commands to it; otherwise (by default), simply adds commands at the end of file file.

- **digits**
  - integer; setting of \f[options(digits=..)]\f for purpose of number representation.

- **title**
  - a string, possibly using LaTeX commands, which will span the columns of the LaTeX matrix

Value

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

Author(s)

For S: Vincent Carey <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

- \f[latex\f in package \f[Hmisc\f is more flexible (but may surprise by its auto-printing ..).]

Examples

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

```r
## The last command produces the file "mat<xyz>.tex" containing
##> \begin{tabular} {| l|| c| c| c|}
##> \multicolumn{ 4 }{c}{ $\pi$, $e$, etc. } \ \hline
##> \ & a & b & c \ \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"))

---

### missingCh

**Has a Formal Argument been Set or is it Missing?**

**Description**

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

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

**Usage**

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

**Arguments**

- **x**
  - a **character** string.

- **envir**
  - a (function evaluation) **environment**, in which the variable named `x` is to be "missing".

**Value**

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

**Author(s)**

Martin Maechler

**See Also**

`missing`

**Examples**

```r
tst1 <- function(a, b, dd, ...) ## does not work an with argument named 'c' !
  c(b = missingCh("b"), dd = missingCh("dd"))
tst1(2)#-> both 'b' and 'dd' are missing
tst1(3,3)
  ## b    dd
  ## FALSE TRUE -- as 'b' is not missing but 'dd' is.
Tst <- function(a,b,cc,dd,EEE, ...)
  vapply(c("a","b","cc","dd","EEE"), missingCh, NA, envir=environment())
```
Tst()
## TRUE ... TRUE -- as all are missing()
Tst(1,3)
## a b cc dd EEE
## FALSE TRUE FALSE TRUE TRUE
## ...... ......
## as 'a' and 'cc' where not missing()

## Formal testing:
stopifnot(tst1(), !tst1(,3), Tst(),
Tst(1,3, b=2, E="bar") == c(0,0,1,0,0))
## maybe surprising that this ^^ becomes 'dd' and only 'cc' is missing

---

### mpl

**Simple Matrix Plots**

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

`p.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 `matplot()` directly is more appropriate.

**Author(s)**

Martin Maechler

**See Also**

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

```r
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

```r
mult.fig(nr.plots, mfrow, mfcol, marP = rep(0, 4),
         mgp = c(if(par("las") != 0) 2. else 1.5, 0.6, 0),
         mar = marP + 0.1 + c(4,4,2,1), oma = c(0,0, tit.wid, 0),
         main = NULL,
         tit.wid = if (is.null(main)) 0 else 1 + 1.5*cex.main,
         cex.main = par("cex.main"), line.main = cex.main - 1/2,
         col.main = par("col.main"), font.main = par("font.main"), ...)
```

Arguments

- `nr.plots` integer; the number of plot figures you’ll want to draw.
- `mfrow, mfcol` instead of `nr.plots`: integer(2) vectors giving the rectangular figure layout for `par(mfrow = *)`, or `par(mfcol=*)`, respectively. The default is to use `mfrow = n2mfrow(nr.plots)`.
- `marP` numeric(4) vector of figure margins to *add* ("Plus") to default `mar`, see below.
- `mgp` argument for `par(mgp= .)` with a smaller default than usual.
- `mar` argument for `par(mar= .)` with a smaller default than usual, using the `marP` argument, see above.
- `oma` argument for `par oma= .), by default for adding space for the main title if necessary.
- `main` character. The main title to be used for the whole graphic.
- `tit.wid` numeric specifying the vertical width to be used for the main title; note that this is only used for the default value of `oma` (s. above).
- `cex.main` numeric; the character size to be used for the main title.
- `line.main` numeric; the margin line at which the title is written (via `mtext(main, side=3, outer=TRUE, line = line.main, ....))`.
- `col.main, font.main` color and font for main title, passed to `mtext()`, see also `par(*)`.
- `...` further arguments to `mtext` for the main title.
Value

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

- `new.par` - the current par settings.
- `old.par` - the par before the call.

Author(s)

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

See Also

- `par`, `layout`.

Examples

```r
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))))
```

## Manual setting of `tit.wid` in case subsequent code also manages `par()`:
```r
mult.fig(4, tit.wid = 2)$old.par -> opar
plot(lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings))
par(opar) # reset
```

### 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

```r
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

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

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

```r
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

- `x, y` coordinates at which to plot. If `y` is missing, `x` is used for both, if it’s a `data.frame`, `list`, 2-column matrix etc – via `xy.coords`; formula do not work.
- `nam` the labels to plot at each `(x, y)`. Per default, these taken from the data `x` and `y`; case numbers `1:n` are taken if no names are available.
- `abbr` logical indicating if the `nam` labels should be abbreviated – with a sensible default.
- `xlab, ylab` labels for the x- and y- axis, the latter being empty by default.
nearcor

log character specifying if log scaled axes should be used, see plot.default.
cex plotting character expansion, see par.
col color to use for text().
... 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 "smoothes" an improper correlation matrix as it can result from cor with use="pairwise.complete.obs" or hetcor.
It is deprecated now, in favor of nearPD() from package Matrix.

Usage
nearcor(R, eig.tol = 1e-6, conv.tol = 1e-07, posd.tol = 1e-8,
maxits = 100, verbose = FALSE)

Arguments
R a square symmetric approximate correlation matrix
eig.tol defines relative positiveness of eigenvalues compared to largest, default=1e-6.
conv.tol convergence tolerance for algorithm, default=1.0e-7
posd.tol tolerance for enforcing positive definiteness, default=1.0e-8
maxits maximum number of iterations
verbose logical specifying if convergence monitoring should be verbose.
Details

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

Value

A list, with components

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

Author(s)

Jens Oehlschlägel

References

See those in posdefify.

See Also

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

hetcor, eigen; posdefify for a simpler algorithm.

Examples

```r
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)
```
nr.sign.chg

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 segments().

Author(s)

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

arrows.

Examples

eexample(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")

Description

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

Usage

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

Arguments

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

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 \( \chi^2 \)-density function \texttt{dchisq}.
shape number, the shape parameter for the Gamma distribution.
h0.col color specification for the line \( y = 0 \).
ms.lines logical, used for the normal only: should lines be drawn at the mean and \( \pm 1 \) standard deviation.
ms.col color for the ms lines if ms.lines is TRUE.
... further parameter passed to \texttt{curve()}, e.g., add = TRUE for adding to current plot.

Author(s)

Werner Stahel et al.

See Also

the underlying density functions, \texttt{dnorm}, \texttt{dchisq}, \texttt{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)
**Add a Horizontal Boxplot to the Current Plot**

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

....

**Author(s)**

Martin Maechler building on code from Markus and Christian Keller.

**See Also**

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

**Examples**

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

Plot a profile.nls Object With Profile Traces

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

Usage
\begin{verbatim}
p.profileTraces(x, cex = 1, 
    subtitle = paste("t-Profiles and traces of ", 
        deparse(attr(x,"summary")$formula)))
\end{verbatim}

Arguments
\begin{itemize}
\item \texttt{x} \hspace{1cm} an object of class "profile.nls", typically resulting from \texttt{profile(nls(.))}, see \texttt{profile.nls}.
\item \texttt{cex} \hspace{1cm} character expansion, see \texttt{par(cex =)}.
\item \texttt{subtitle} \hspace{1cm} a subtitle to set for the plot. The default now includes the \texttt{nls()} formula used.
\end{itemize}

Note
the \texttt{stats}-internal \texttt{stats:::plot.profile.nls} plot method just does "the diagonals".

Author(s)
Andreas Ruckstuhl, \texttt{R} port by Isabelle Flückiger and Marcel Wolbers

See Also
\texttt{profile}, and \texttt{nls} (which has unexported \texttt{profile} and \texttt{stats:::plot.profile.nls} methods).

Examples
\begin{verbatim}
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)
\end{verbatim}
**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Author(s)**

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

**See Also**

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

**Examples**

```r
I <- 8; J <- 3; K <- 20
xx <- factor(rep(rep(1: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")
mtextr("p.res.2fact(xx, yy, zz, restr= 4, ..)",
       line=1, adj=1, outer=TRUE, cex=1)
```

## Real data

data(warpbreaks)

```r
(fm) <- lm(breaks ~ wool*tension, data = warpbreaks)
```
## call via formula method of p.res.2x():
p.res.2x(~ ., fm1) # is shorter than, but equivalent to
## p.res.2x(~ wool + tension, fm1) ## or the direct
## with(warpbreaks, p.res.2fact(wool, tension, residuals(fm1)))
## whereas this is "transposed":
p.res.2x(~ tension+wool, fm1)

### Description
Plot Residuals, e.g., of a multiple linear regression, against two (predictor) variables, using positively and negatively oriented line segments for positive and negative residuals.

This is a (S3) **generic** function with a **default** and a **formula** method.

### Usage

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

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

### Arguments

- `x, y` numeric vectors of the same length specifying 2 covariates. For the formula method, `x` is a **formula**.
- `z` numeric vector of same length as `x` and `y`, typically residuals.
- `restricted` positive value which truncates the size. The corresponding symbols are marked by stars.
- `size` the symbols are scaled so that `size` is the size of the largest symbol in cm.
- `slwd, scol` line width and color(s) for the residual **segments**. If `scol` has length 2 as per default, the two colors are used for positive and negative `z` values, respectively.
- `xlab, ylab, main` axis labels, and title see **title**, each with a sensible default. To suppress, use, e.g., `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**, or `p.res.2x.default()`, respectively.
- `data` (for the **formula** method:) a data frame or a fitted "lm" object.
Details

Each residual $zz[i]$ is visualized as line segment centered at $(xx_i, yy_i)$, $i = 1, \ldots, n$, where the lengths of the segments are proportional to the absolute values $\|zz_i\|$. Positive residuals’ line segments have slope +1, and negative ones slope −1, and `scol` is used to use different colors for negative and positive segments.

The formula interface calls `p.res.2fact()` when both x and y are factors.

Author(s)


References


See Also

`p.res.2fact`, `plot.lm`, `TA.plot`.

Examples

```r
xx <- rep(1:10,7)
yy <- rep(1:7, rep(10,7))
zz <- rnorm(70)
p.res.2x(xx,yy,zz, restricted = 2, main = "i.i.d. N(0,1) random residuals")

eexample(lm.influence, echo = FALSE)

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

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

par(op) # revert par() settings above
```

Description

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

\[ p.scales(\text{unit} = \text{relsysize} \times 2.54 \times \min(\text{pin}), \text{relsysize} = 0.05) \]

Arguments

- \text{unit} length of unit (or x and y units) of symbol coordinates in cm.
- \text{relsysize} same, as a proportion of the plotting area.

Value

A numeric 2x2 matrix, with rows named \text{x} and \text{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

\texttt{par("usr")}, of also ("pin") on which this is based.

Examples

\begin{verbatim}
p.scales()
\end{verbatim}

p.tachoPlot

\textit{Draw Symbol on a Plot}

Description

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

Usage

\begin{verbatim}
p.tachoPlot(x, y, z, \text{angle} = c(\pi/4, 3\pi/4), \text{size},
\text{method} = c("robust", "sensitive", "rank"),
\text{legend} = \text{TRUE}, \text{show.method} = \text{legend},
\text{ylab} = \text{deparse(\text{substitute}(y))}, \text{ylim, xlim, ...})
\end{verbatim}
**Arguments**

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

**Details**

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

**Side Effects**

A plot is created on the current graphics device.

**Author(s)**

Christian Keller, June 1995

**See Also**

- symbols

**Examples**

data(state)
data(USArrests)
plot(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!)

```r
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**

```r
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**

- `x` timeseries (possibly multivariate) or numeric vector.
- `nrplots` number of sub-plots. Default: in {1..8}, approximately n/400 if possible.
- `overlap` by how much should subsequent plots overlap. Defaults to about 1/16 of sub-length on each side.
- `date.x` a time “vector” of the same length as x and coercable to class "POSIXct" (see 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 `multi.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

\begin{verbatim}
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)
\end{verbatim}

---

\section*{Description}

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

\section*{Usage}

\begin{verbatim}
paste.vec(name, digits = options()$digits)
\end{verbatim}

\section*{Arguments}

\begin{itemize}
  \item{name} string with an variable name which must exist in the current environment (R session).
  \item{digits} how many decimal digits to be used; passed to format.
\end{itemize}

\section*{Value}

a string of the form "NAME = x1 x2 ..."
Author(s)

Martin Maechler, about 1992.

Examples

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

pkgDesc

Version of packageDescription() as Simple Vector

Description

a simple “version”, or wrapper for packageDescription(), returning a named character vector, including "file", and still has a useful print() method.

Usage

```
pkgDesc (pkg, lib.loc = NULL, fields = NULL, ...)
pkgBuilt(pkg, lib.loc = NULL, ...)
```

Arguments

- `pkg` a character string, name of an installed R package.
- `lib.loc` library location to find the package in; the default NULL uses the full .libPaths().
- `fields` a character vector (or NULL) specifying fields to be returned.
- `...` further optional arguments passed to packageDescription().

Value

a named character vector, with names, the fields, identical to the names of the list returned by packageDescription, plus its “file” attribute. Additionally the resulting vector is of class “Dlist” which activates a useful print() method.

Note

The file is always returned; not the least that the author wants to see it quite often as his .libPaths() is non-trivial and typically longer than 4 entries.

Author(s)

Martin Maechler, Jan. 2021

See Also

packageDescription, .libPaths.
pkgLibs

Examples

```r
str(pd <- pkgDesc("sfsmisc"))
pd[c("Date", "Packaged", "Built", "file")]

pkgBuilt("sfsmisc")

## Show "Built" (and "file") for all packages whose namespaces are loaded:
lns <- loadedNamespaces()
mlNs <- sapply(lns, pkgBuilt)
t(mlNs) # typically prints nicely

pkgs <- c("grid", "lattice", "MASS", "Matrix", "nlme", "lme4", "sfsmisc")

pkgsOk <- basename(find.package(pkgs, quiet=TRUE))
mpkg <- sapply(pkgsOk, pkgBuilt)
stopifnot(is.matrix(mpkg), nrow(mpkg) == 2)

pkgsOk <- c("foobar", "barbar", pkgs, "kitty") 

pkgsOk <- basename(find.package(pkgs, quiet=TRUE))

mpkgOk <- c("grid", "lattice", "MASS", "Matrix", "nlme", "lme4", "sfsmisc")

pkgsOk <- c("foobar", "barbar", pkgs, "kitty")

pkgsOkOk <- basename(find.package(pkgsOk, quiet=TRUE))

mpkgOk <- sapply(pkgsOkOk, pkgBuilt)

stopifnot(is.matrix(mpkgOk), nrow(mpkgOk) == 2)

mpkgOk["Built",]
```

pkgLibs

R Package Compiled Code Library Dependencies (on Unix-alikes)

Description

List some system level information about the compiled code library, typically its dependencies, for R packages with compiled code; for Unix-alikes or more generally when cmd is installed locally.

Usage

```r
pkgLibs(pkg, 
  cmd = if(Sys.info()[["sysname"] == "Darwin") "otool -L" else "ldd"
```

Arguments

- **pkg** character vector of package names of installed R packages.
- **cmd** a character string with the name of an OS / system level program (to be called via system(cmd, ...)) which gives information about the shared library (of compiled code), also known as “DLL” (dynamically loadable library) or “so” ((dynamic) shared object) library. The default, "ldd" is a standard binary utility on Unix-alike platforms such as Linux. On macOS, "otool -L" is used by default.

Details

Note that there seems some language confusion as “DLL” on Windows is also used for “Dynamic-link Library” and Wikipedia warns about confusing the two concepts (“dynamically loaded ..” vs “dynamic-link ..”).
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.

Value

A named list with one entry per package in pkg, the names being the directory / folder names of the corresponding pkgs from pkg.

The exact structure of such entries is currently subject to change and you should not rely on its exact format for now.

Author(s)

Martin Maechler

References


man ldd from a terminal on a valid OS.

See Also

dyn.load(), library.dynam(), and getLoadedDLLs().

Also, .C, .Call which use such DLLs.

Examples

# for the example only using standard R packages :
myPkgs <- c("stats", "MASS", "rpart", "Matrix")
pl <- pkgLibs(myPkgs)
pl
stopifnot(exprs = {
is.list(pl)
length(pl) == length(myPkgs)
is.character(pkgD <- names(pl))
})
## Have seen this failing when a strange development version of "Matrix" was picked up:
try( stopifnot( dir.exists(pkgD) ) )
**plotDS**

Usage

```r
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

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

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

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

**Usage**

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

**Arguments**

- `ti` numeric vector = $X[1:N]$ or $t[0:n]$.
- `y` numeric vector $y[1:n]$; if omitted take $y = k/N$ for empirical CDF.
- `cad.lag` logical: Draw ‘cad.lag’, i.e., “continue à droite, limite à gauche”. Default = TRUE.
- `verticals` logical: Draw vertical lines? Default= ! cad.lag
- `left.points` logical: Draw left points? Default= cad.lag
### plotStep

- **right.points**  logical: Draw right points? Default= FALSE
- **end.points**  logical: Draw 2 end points? Default= FALSE
- **add**  logical: Add to existing plot? Default= FALSE
- **pch**  plotting character for points, see `par()`.
- **xlab,ylab**  labels of x- and y-axis
- **main**  main title; defaults to the call’ if you do not want a title, use `main = ""`.
- **...**  Any valid argument to `plot(..)`.

#### Value

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

#### Side Effects

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

#### 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

```r
#-- 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
```
polyn.eval  Evaluate Polynomials

Description
Evaluate one or several univariate polynomials at several locations, i.e. compute \( \text{coef}[1] + \text{coef}[2] \times \text{x} + \ldots + \text{coef}[\text{p}+1] \times \text{x}^\text{p} \) (in the simplest case where \text{x} is scalar and \text{coef} a vector).

Usage
polyn.eval(coef, x)

Arguments

- **coef**  
  “numeric” vector or matrix. If a vector, \text{x} can be an array and the result matches \text{x}.
  If \text{coef} is a matrix it specifies several polynomials of the same degree as rows, \text{x} must be a vector, \text{coef}[,k] is for \( \text{x}^{k-1} \) and the result is a matrix of dimension \text{length(x)} * nrow(coef).
  Note that \text{coef} can also be \text{complex} or big rational (\text{as.bigq(\ldots)} from \text{gmp}, or arbitrary precision (\"mpfr\") from \text{Rmpfr}, or similar number-like objects for which basic arithmetic is defined.

- **x**  
  “numeric” vector or array. Either \text{x} or \text{coef} must be a vector.

Details
The stable “Horner rule” is used for evaluation in any case.

When \text{length(coef)} == 1L, polyn.eval(coef, x) now returns a vector of \text{length(x)} whereas previously, it just gave the number \text{coef} independent of \text{x}.

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, see, e.g., predict.polynomial. For multivariate polynomials (and also for nice interface to the orthopolynom package), consider the mpoly package.
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
m a numeric (square) matrix.
method a string specifying the method to apply; can be abbreviated.
symmetric logical, simply passed to eigen (unless eigen.m is specified); currently, we do not see any reason for not using TRUE.
eigen.m the eigen value decomposition of m, can be specified in case it is already available.
eps.ev number specifying the tolerance to use, see Details below.
Details

We form the eigen decomposition

$$m = V \Lambda V'$$

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

When the smallest eigenvalue $\lambda_n$ are less than $Eps <- \text{eps.ev} * \text{abs(}\lambda[1])$, 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. We consider nearPD() to also be the successor of this package’s nearcor().

Author(s)

Martin Maechler, July 2004

References


See Also
eigen on which the current methods rely. nearPD() in the Matrix package. (Further, the deprecated nearcor() from this package.)
potatoes

Examples

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

---

**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**


**References**

Examples

```r
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**

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

**Arguments**

- `x` numeric vector (e.g. axis tick locations)
- `drop.1` logical indicating if $1 \times$ should be dropped from the resulting expressions.
- `sub10` logical, "$10^{1}\$", a non-negative integer number or an integer vector of length two, say $(k_1,k_2)$, indicating if some $10^{j}$ expressions for $j \in J$ should be formatted traditionally, notably e.g., $10^{0} \equiv 1$.
  - When a (non-negative) number, say $k$, $J = \{j; j \leq k\}$ are all simplified, when a length–2 vector, $J = \{j; k_1 \leq j \leq k_2\}$ are.
  - Special cases: `sub10 = TRUE` means to use 1 instead of $10^{0}$ and `sub10 = "10"` uses both 1 for $10^{0}$ and 10 for $10^{1}$; these are short forms of `sub10 = c(0,0)` and `sub10 = c(0,1)` respectively.
**pretty10exp**

- **digits**: number of digits for mantissa \((a)\) construction; the number of *significant* digits, see `signif`.
- **digits.fuzz**: the old deprecated name for `digits`.
- **lab.type**: a string indicating how the result should look like. By default, (plotmath-compatible) expressions are returned. Alternatively, `lab.type = "plotmath"` returns LaTeX formatted strings for labels. (The latter is useful, e.g., when using the tikzDevice package to generate LaTeX-processed figures.)
- **lab.sep**: character separator between mantissa and exponent for LaTeX labels; it will be prepended with a backslash, i.e., "cdot" will use "\cdot".

**Value**

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

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

**Note**

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

**Author(s)**

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

**See Also**

- `axTexpr` and `eaxis()` which build on `pretty10exp()`, notably the `eaxis()` example plots.
- The new `toLatex.numeric` method which gives very similar results with option `scientific = TRUE`.
- Further, `axis`, `axTicks`.

**Examples**

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

set.seed(17); lx <- rlnorm(10, m=8, s=6)
pretty10exp(lx, digits = 3)
pretty10exp(lx, digits = 3, sub10 = 2)
pretty10exp(lx, digits = 3, lab.type="latex")
pretty10exp(lx, digits = 3, lab.type="latex", lab.sep="times", sub10=2)
```
## use regular formatted numbers from 0.03 to 300:
pretty10exp(3*10^(-3:4), sub10 = c(-2,2))
pretty10exp(3*10^(-3:4), sub10 = c(-2,2), lab.type = "l")

ax <- 10^(-6:0) - 2e-16
pretty10exp(ax, drop.1=TRUE) # nice for plotting
pretty10exp(ax, drop.1=TRUE, sub10=TRUE)
pretty10exp(ax, drop.1=TRUE, sub10=c(-2,2))

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

---

### primes

**Find all Primes Less Than n**

---

**Description**

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

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

**Usage**

`primes(n, pSeq = NULL)`

**Arguments**

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

**Details**

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

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

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

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

**Value**

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

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

See Also

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

Examples

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

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

## Exploring p(n) := the n-th prime number -= n * pnn(n), where
## pnn(n) := log n + log log n
pnn <- function(n) { L <- log(n); L + log(L) }

n <- 6:(N <- length(PR <- primes(1e5)))
m.pn <- cbind(l.pn = ceiling(n*(pnn(n)-1)), pn = PR[n], u.pn = floor(n*pnn(n)))
matplot(n, m.pn, type="l", ylab = quote(p[n]), main = quote(p[n] ~~
"with lower/upper bounds" ~ n*(log(n) + log(log(n)) -(1~"or"~0)))
## (difference to the lower approximation) / n --> ~ 0.0426 (?) :
plot(n, PR[n]/n - (pnn(n)-1), type = 'l', cex = 1/8, log="x", xaxt="n")
eaxis(1); abline(h=0, col=adjustcolor(1, 0.5))
```

printTable2

Add and Print Marginals for 2-way Contingency Tables

Description

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

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

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

Arguments

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

Value

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

printTable2 is just producing output.

Author(s)

Martin Maechler, Feb.1993; then Dec 2003

See Also

table, ftable.

Examples

margin2table(diag(4),,TRUE)
m <- diag(3); colnames(m) <- letters[1:3]
margin2table(m)
margin2table(m / sum(m))
data(HairEyeColor)
margin2table(HairEyeColor[, "Male"])
printTable2(HairEyeColor[, "Male"])
printTable2(HairEyeColor[, "Female"])


Description

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

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

Usage

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

Arguments

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

Author(s)

Martin Maechler, originally for S-PLUS.

---

`ps.end`  
*Close PostScript or Acrobat Graphics Device opened by 'ps.do' / 'pdf.do'*

Description

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

Usage

```r
ps.end(call.gv= NULL, command = getOption("eps_view"),
   debug = getOption("verbose"))
```

```r
df.end(call.viewer= NULL, command = getOption("pdfviewer"),
   debug = getOption("verbose"))
```
Arguments

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

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

- **debug**
  logical; if TRUE print information during execution.

Details

Depends on Unix tools, such as `ps`.

Author(s)

Martin Maechler

See Also

`postscript`, `postscript pdf.do`, `ps.do`, ...

Examples

```r
if(interactive())
{
  myPS <- tempfile("ex", fileext = ".ps")
  ps.do(myPS)
  data(sunspots)
  plot(sunspots)
  ps.end()

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

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

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

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

```r
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 *inches*, height \* 2.54 are *cm*. The default is 5 plus 1.25 iff `main.space`.
- **width** device width in *inches*; for this and height, see `postscript`.
- **onefile**, **horizontal** logicals passed to `postscript()` or `pdf()`, most probably to be left alone.
- **title** PostScript/PDF (not plot!) title passed to `postscript()` or `pdf()`; by default use a title with \(\texttt{R}\) version and `file` in it.
- **version** a string describing the PDF version that will be required to view the output, see `pdf`; our (high) default ensures alpha-transparency.
- **quiet** logical specifying that some (informative/warning) messages should not be issued.
- **main.space** logical; if true, leave space for a main title (unusual for LaTeX figures!).
- **lab.space** logical; if true, leave space for x- and y- labels (by *not* subtracting from `mar`).
paper character (or missing), typically "a4" or "a4r" in non-America, see postscript. Only if this is "special" (or missing) are your choices of width and height completely honored (and this may lead to files that cannot print on A4) with resizing.

lab integer of length 3, lab[1:2] are desired number of tick marks on x- and y-axis, see par(lab=).

mgp.lab three decreasing numbers determining space for axis labeling, see par(mgp=), the default is here smaller than usual.

mar four numbers, indicating marginal space, see par(mar=), the default is here smaller than usual.

... arguments passed to ps.do() or pdf.do() from ps.latex / pdf.latex and to ps.options from ps.do/pdf.do.

Details

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

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

pdf.do() sets the default paper to "special" when both width and height are specified.

Value

A list with components

old.par containing the old par values

new.par containing the newly set par values

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)
```r
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**

```r
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**

```r
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))
```
Quasi Random Numbers via Halton Sequences

Description

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

Usage

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

Arguments

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

Value

- \text{sHalton}(n,m)\) returns a numeric vector of length \( n-m+1 \) of values in \([0, 1]\).
- \text{QUnif}(n, \text{min}, \text{max}, n.\text{min}, p=p)\) generates \( n-n.\text{min}+1 \) \( p \)-dimensional points in \([\text{min}, \text{max}]^p\) returning a numeric matrix with \( p \) columns.

Note

For \( \text{leap} \) Kocis and Whiten recommend values of \( L = 31, 61, 149, 409 \), and particularly the \( L = 409 \) for dimensions up to 400.

Author(s)

Martin Maechler
References


Examples

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")

---

read.org.table

*Read.table for an Emacs Org Table*

Description

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

Usage

read.org.table(file, header = TRUE, skip = 0,
  encoding = "native", fileEncoding = "," , text, quiet=FALSE, ...)
Arguments

- `file`: a file name, a file or other connection.
- `header`: logical indicating if the org table has header line (in the usual "|"-separated org table format).
- `skip`: integer number of initial lines to skip.
- `encoding`: to be used in the main `readLines(file, encoding=encoding)` call.
- `fileEncoding`: if `file` is a file name, i.e., a character string, and `fileEncoding` is not the empty string, `file(file, "rt", encoding = fileEncoding)` will be used.
- `text`: instead of `file`, a character or string (of a few lines, typically).
- `quiet`: logical to suppress the message which is signalled when no `nrows=*` has been specified and the automatic number of rows is smaller than 95% of the rows / non-header lines of the file.

... further arguments passed to `read.table`. You should not use `encoding` (but possibly `fileEncoding`) here, as we do not call `read.table` on `file` (but on a `textConnection`).

Value

- a `data.frame`

Note

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

References

- Org tutorial for tables, [https://orgmode.org/worg/org-tutorials/tables.html](https://orgmode.org/worg/org-tutorials/tables.html)

See Also

- CRAN package `ascii` can write org tables. `read.table`

Examples

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

relErr

Relative Error When Appropriate, Absolute Otherwise

Description

relErrV(): Compute the signed relative error componentwise ("vectorized") between the target and current vectors, using the absolute error, i.e., the difference in case the relative error is not well defined, i.e., when target is zero or infinite.

relErr(): simply the mean absolute value of the relative errors between target and current vectors; typically the “same” as all.equal.numeric(target, vector, tolerance=0, countEQ=TRUE). Currently useful only when both vectors are finite.

Usage

relErrV(target, current, eps0 = .Machine$double.xmin)
relErr (target, current)

Arguments

target numeric, possibly scalar.
current numeric vector of length() a multiple of length(target); if an array (incl matrix), dimensions are preserved; for vectors, names(target) are preserved.
eps0 non-negative number; values abs(target) < eps0 should be treated as zero (and hence absolute instead of relative error be computed). This may be crucial when target is an "mpfr"-number vector.

Value

relErrV(): a numeric vector of the same length (or array of the same dimension) as current.
relErr(): a single number.

Author(s)

Martin Maechler, originally as part of Matrix package’s ‘test-tools.R’.

See Also

all.equal.numeric() is similar in spirit but returns TRUE or string containing the mean relative or absolute error.
Examples

```r
## relErrV() test example: showing how it works fine with {NA, Inf, 0}:
eps <- 1e-4*c(-9, -8, -6, -4, 0.5, 1, 5)
target <- c(-1:1, 0, NA, NA, Inf, -Inf, 0, Inf, 1, -3:3)
current <- c(-1:1,1e-7,NaN,NA, 0, Inf, Inf, 0, Inf, 1, Inf, -3:3+eps)
cbind(target, current, absE = current-target,
      relE = relErrV(target,current)) -> M ; M
stopifnot(exprs = {
  is.logical(isFr <- is.finite(rF <- M[,"relE"]))
  target==current | isFr == is.finite(aF <- M,"absE")
  identical(aF[!isFr] , rF[!isFr])
  identical(numeric(), relErrV(numeric(), integer())) # length 0 (used to fail)
})
```
Arguments

<table>
<thead>
<tr>
<th>char</th>
<th>single character (or arbitrary string).</th>
</tr>
</thead>
<tbody>
<tr>
<td>phi</td>
<td>non-negative integer.</td>
</tr>
</tbody>
</table>

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
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**

<table>
<thead>
<tr>
<th>xy</th>
<th>numeric 2-column matrix, or coercable to one.</th>
</tr>
</thead>
<tbody>
<tr>
<td>phi</td>
<td>numeric scalar, the angle in radians (i.e., phi=pi corresponds to 180 degrees) by which to rotate the points.</td>
</tr>
</tbody>
</table>
Value

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

Author(s)

Martin Maechler, Oct.1994

Examples

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

---

### rotn

Generalized Rot13 Character Translation (Rotation)

Description

Compute generalized ‘rot13’ character translations or “rotations”

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

Usage

`rotn(ch, n = 13)`

Arguments

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

Note that the default \( n = 13 \) makes \texttt{rotn} into a function that is its own inverse.

Written after having searched for it and found \texttt{seqinr::rot13()} which was generalized and rendered more transparently to my eyes.

Value

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

Author(s)

Martin Maechler

See Also

\texttt{rot2}, a completely different rotation (namely in the plane aka \( R^2 \)).

Examples

\begin{verbatim}
\texttt{rotn(c("ABC", "a","b","c"), 1)}
\texttt{rotn(c("ABC", "a","b","c"), 2)}
\texttt{rotn(c("ABC", "a","b","c"), 26)} # rotation by 26 does not change much

(ch <- paste("Hello", c("World!", "you too")))
\texttt{rotn(ch)}
\texttt{rotn(rotn(ch))} # rotn(*, 13) is its own inverse
\end{verbatim}

roundfixS

\textit{Round to Integer Keeping the Sum Fixed}

Description

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

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

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

Usage

\begin{verbatim}
\texttt{roundfixS(x, method = c("offset-round", "round+fix", "1greedy"))}
\end{verbatim}
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*:

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

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

See Also

- `round`

Examples

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

x <- c(-1.4, -1, 0.244, 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")
}
rrange <- roundfixS(y, method="lg")
stopifnot(all.equal(sum(r2), S),
  all.equal(sum(r2.), S),
  all.equal(sum(r2_), S))
  all(r2 == r2. & r2. == r2_) # TRUE if all give the same result
}

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

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

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

## End(Not run)

---

rrange

Robust Range using Boxplot 'Quartiles'

Description

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

Usage

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

Arguments

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

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

Value

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

Author(s)

Martin Maechler, 1990.

See Also

`range`, `fivenum`, `boxplot` and `boxplot.stats`.

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

Examples

```r
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

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

Arguments

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

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(*, 100, \"interpol\")")
points(match(x,xt), x, col = 2, pch = 20)
# .... you don't even see that it's not equidistant
# whereas the cheap method shows ...
x2 <- seqXtend(x, 100)
plot(x2, col="blue")
points(match(x,x2), 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)
s1 <- summary(as.numeric(diff(seqXtend(px, n3)))) # wildly varying
summary(as.numeric(diff(seqXtend(px, n3, "aim")))) # (ditto)
s2 <- summary(as.numeric(diff(seqXtend(px, n3, "int")))) # around 30
Description

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

`isRshared()` is a utility called from `sessionInfoX()`.

Usage

```r
sessionInfoX(pkgs = NULL, list.libP = FALSE, extraR.env = TRUE)
```

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

```r
isRshared(platform = .Platform)
```

Arguments

- `pkgs`: NULL (default), TRUE or a character vector of R package names, whose `packageDescription()`s are wanted. No packages by default, TRUE takes all currently loaded pkgs.
- `list.libP`: a logical indicating if for all `.libPaths` entries, the files should be listed via `list.files`.
- `extraR.env`: logical indicating if all environment variables should be recorded which start with "R_" or "_R_".
- `x`: typically the result of `sessionInfoX()`.
- `locale`: logical, passed to `print.sessionInfo()` indicating if the locale information should be printed.
- `RLIBS`: logical indicating if the information about R_LIBS should be printed.
- `Renv`: logical indicating if the information about R environment variables should be printed.
- `...`: passed to `print` methods.
- `platform`: a list “like” `.Platform`.

Value

For `isRshared()`, a logical indicating if R has been installed as “shared”, i.e., linked to ‘libR*’ shared library.

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

- `sInfo`: simply the value of `sessionInfo()`.
- `sysInf`: the value of `Sys.info()`.
capabilities the value of `capabilities()`.


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

grSoft for R 3.2.0 and newer, the value of `grSoftVersion()`.

tclVersion for R 3.2.0 and newer and when `tcltk` is loaded, the Tcl version (`tclVersion()`).

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

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

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

libPath the value of `.libPaths()`.

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

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

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

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

shared (not available on Windows, where it is conceptually always true:) logical indicating if the version of R is “shared”.

Author(s)

Martin Maechler, December 2015 ff.

See Also

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

Examples

```r
six0 <- sessionInfoX()
six0$shared # useful (for some, e.g., MM) on Unix alikes

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

Description

From base R’s `R.version.string`, produce a somewhat shorter version, with or without date, notably also for patched or development versions of R.

Main use is for plotting or construction of file of variable names.

Usage

```r
shortRversion(Rv = R.version, Rst = Rv$status, 
Rvstring = if (!is.null(s <- Rv$version.string)) s else R.version.string, 
   date = Rst != "", spaces = TRUE)
```

Arguments

- **Rv**
  - a list with some of the same components as `R.version`.
- **Rst**
  - a string specifying the status of R’s version. For released versions of R, this is ""; almost always use the default `Rv$status`.
- **Rvstring**
  - a string with a default that should work even for R versions previous to 1.0.0.
- **date**
  - logical specifying if the date of the R version should be included in the result; by default, this will be true only for non-released versions of R.
- **spaces**
  - logical indicating if the result may contain spaces (aka ‘blanks’); setting it to false, replaces the blanks by "-" and "_".

Value

a character string, typically a shortened version of `Rvstring`.

Author(s)

Martin Maechler

See Also

`R.version`, `R.version.string`

Examples

```r
shortRversion() ## (including the date, typically for an R Core developer) 
## but this is shorter: 
(Rver <- shortRversion(date=FALSE)) 
shortRversion(spaces=FALSE)# e.g. for a file of even directory name 
shortRversion(spaces=FALSE, date=FALSE)# even shorter, ditto 

## If you want even shorter { abbreviate() will remove spaces, too }:
```
Description
Rounds to significant digits similarly to `signif`.

Usage
`signi(x, digits = 6)`

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

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

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

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

See Also
`signif`, `round`.

Examples
```r
(x1 <- seq(-2, 4, by = 0.5))
identical(x1, signi(x1)) # since 0.5 is exact in binary arithmetic
(x2 <- pi - 3 + c(-5,-1,0,.1,.2,1,10,100))
signi(x2, 3)
```
Source and Attach an R source file

Description

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

Usage

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

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

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

Usage

```r
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

```r
str_data("cluster")

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

```r
## Filtering (and return value)
df1 <- str_data("datasets", filterFUN=is.data.frame)
str(df.d <- df1$datasets)
```

```r
## dim() of all those data frames:
t(sapply(unlist(df.d), function(.) dim(get(.))))
```

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

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

**Description**

Return information about the Linux hardware, notably the CPU (the central processor unit) and memory of the computer \( \mathcal{R} \) 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**

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

**Arguments**

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

**Value**

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

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

**Note**

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

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

Conceivably, the bogoMIPS source code is open and available and could be built into \( \mathcal{R} \).

**Author(s)**

Martin Maechler
See Also

`Sys.ps`, etc.

Examples

```r
(n.cores <- parallel::detectCores())
if(substr(R.version["os"], 1,5) == "linux") { ##-- only on Linux
  Sys.cpuinfo() # which is often ugly; this looks much better:
  length(Sys.cpu2 <- local({I <- Sys.cpuinfo(); I[!grepl("^flags", names(I)) ] }))
  ## may still be too much, notably if n.cores > 2:
  (Sys3 <- Sys.cpu2[!grepl("[.][0-9]+$", names(Sys.cpu2))])

  Sys.MIPS() ## just the 'bogomips' from above:
  Sys.MIPS() / as.numeric(Sys.cpuinfo()["cpu.MHz"]) # 2 * #cores ((no longer))

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

Description

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

Usage

```r
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"))
```

Arguments

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

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

Value

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

Author(s)

Martin Maechler

See Also

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

Examples

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

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

---

**TA.plot**

_Tukey-Anscombe Plot (Residual vs. Fitted) of a Linear Model_

Description

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

Usage

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

*lm.res*  
Result of *lm(.*), aov(.*), glm(.*)* or a similar object.

*fit*  
fitted values; you probably want the default here.

*res*  
residuals to use. Default: **Weighted** ("Pearson") residuals if weights have been used for the model fit.

*labels*  
strings to use as plotting symbols for each point. Default(NULL): extract observations’ names or use its sequence number. Use, e.g., "*" to get simple * symbols.

*main*  
main title to plot. Default: sophisticated, resulting in something like "Tukey-Anscombe Plot of: y ~ x" constructed from lm.res $ call.

*xlab*  
x-axis label for plot.

*draw.smooth*  
logical; if TRUE, draw a lowess smoother (with automatic smoothing fraction).

*show.call*  
logical; if TRUE, write the "calling" syntax with which the fit was done.

*show.2sigma*  
logical; if TRUE, draw horizontal lines at ±2σ where σ is mad(resid).

*lo.iter*  
positive integer, giving the number of lowess robustness iterations. The default depends on the model and is 0 for non Gaussian glm’s.

*lo.cex*  
character expansion ("cex") for lowess and other marginal texts.

*par0line*  
a list of arguments (with reasonable defaults) to be passed to abline(.) when drawing the x-axis, i.e., the y = 0 line.

*parSmooth, parSigma*  
each a list of arguments (with reasonable default) for drawing the smooth curve (if draw.smooth is true), or the horizontal sigma boundaries (if show.2sigma is true) respectively.

*verbose*  
logical indicating if some construction details should be reported (print(.))ed.

*...*  
further graphical parameters are passed to n.plot(.).

Side Effects

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

Author(s)

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

See Also

*plot.lm* which also does a QQ normal plot and more.

Examples

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

```r
example(airquality)
summary(lmO <- lm(Ozone ~ ., data= airquality))
```
TA.plot(lm0)
TA.plot(lmO, label = "O") # 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

More simplification in tapply() result

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 tapply() and are passed to tapply(.).
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, FUN() 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+length(INDEX); otherwise, tapplySimpl() returns the list r, i.e., the same as tapply().
tkdensity

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)

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

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

Details

library(tcltk) must be working, i.e., Tcl/Tk must have been installed on your platform, and must have been visible during R’s configuration and/or installation.
You can not only choose the bandwidth (the most important parameter), but also the kernel, and you can zoom in and out (in x-range only).

Value

none.
(How could this be done? tcltk widgets run as separate processes!)

Author(s)

Martin Maechler, building on demo(tkdensity).

Examples

if (dev.interactive(TRUE)) ## does really not make sense otherwise
if(try(require("tcltk"))) { ## sometimes (rarely) there, but broken
  data(faithful)
  tkdensity(faithful $ eruptions)
  set.seed(7)
  if(require("nor1mix"))
    tkdensity(rnorMix(1000, MW.nm9), kernels = c("gaussian", "epanechnikov"))
}

Description

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

Usage

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

LaTeX or Sweave friendly Formatting of Numbers

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

Usage

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

Arguments

object a numeric vector.
digits number of digits after the decimal point (for the mantissa if scientific). The default behaves the same as R's format() .
scientific logical indicating if scientific notation \( a \times 10^k \) should be used. The default behaves the same as R's format().
times character string indicating the LaTeX symbol to be used for the 'times' sign.
... unused; for compatibility with toLatex.

Value

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

Note

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

Author(s)

Alain Hauser

See Also

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

Examples

xx <- pi * 10^(-9:9)
format(xx)
formatC(xx)
toLatex(xx) #-> scientific = TRUE is chosen
toLatex(xx, scientific=FALSE)
sapply(xx, toLatex)
sapply(xx, toLatex, digits = 2)
u.assign0

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

Description

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

They have been deprecated since August 2013.

Usage

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

Arguments

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

Note

Really don’t use these anymore...

Author(s)

Martin Maechler

See Also

generate, assign.

u.boxplot.x

Utility Returning x-Coordinates of Boxplot

Description

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

Usage

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

n  number of boxplots.
j  indices of boxplots.
f.fullrange  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).

Author(s)

Martin Maechler, ca. 1992

See Also

u.Datumvonheute.
u.datumdecode

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
\texttt{u.Datumvonheute} \hspace{1cm} \textit{Datum und Uhrzeit (auf deutsch)}

\textbf{Description}

Return current date and time as a string, possibly including day of the week in \textit{German}.

\textbf{Usage}

\texttt{u.Datumvonheute(W.tag=2, Zeit=FALSE)}

\texttt{C.Monatsname}
\texttt{C.Wochentag}
\texttt{C.Wochentagkurz}
\texttt{C.weekday}

\textbf{Arguments}

\texttt{W.tag} \hspace{1cm} \text{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.}

\texttt{Zeit} \hspace{1cm} \text{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.}

\textbf{Value}

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

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

\textbf{Author(s)}

Caterina Savi, Martin Maechler

\textbf{See Also}

\texttt{u.date} for a similar English version, and \texttt{p.datum} which plots. For English month names, etc \texttt{month.name}.

\textbf{Examples}

\texttt{u.Datumvonheute()}
\texttt{u.Datumvonheute(W.tag=1, Zeit=TRUE)}
\texttt{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 \( u \cdot \log(x) \) 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

\[ u \cdot \log(x, c = 1) \]

Arguments

- **x**: numeric vector to be transformed.
- **c**: scalar, > 0

Details

Alternatively, the ‘IHS’ (inverse hyperbolic sine) transform has been proposed, first in more generality as \( S_U() \) curves by Johnson(1949); in its simplest form, \( f(x) = \text{arsinh}(x) = \log(x + \sqrt{x^2 + 1}) \), which is also antisymmetric, continuous and once differentiable as our \( u \cdot \log() \).

Value

numeric vector of same length as \( x \).

Author(s)

Martin Maechler, 24 Jan 1995

References


See Also

Werner Stahel’s sophisticated version of John Tukey’s “started log” (which was \( \log(x + c) \)), with concave extension to negative values and adaptive default choice of \( c \); \texttt{logst} in his CRAN package \texttt{relevance}, or \texttt{LogSt} in package \texttt{DescTools}. 
Examples

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

## Compare with IHS = inverse hyperbolic sine == asinh
ihs <- function(x) log(x+sqrt(x^2+1)) # == asinh(x) (aka "arsinh(x)" or "sinh^{-1}(x)"
X <- c(-Inf, Inf, NA, NaN)
stopifnot(all.equal(X, asinh(X))) # but not for ihs():
cbind(X, asinh=asinh(X), ihs=ihs(X)) # differs for -Inf
X <- runif(500, 0, 4); X[100+0:3] <- X
all.equal(ihs(X), asinh(X)) # all NA value mismatch: asinh() is correct (i.e. better!)
```

```r
curve(u.log, -2, 20, n=1000); abline(h=0, v=0, col = "gray20", lty = 3)
curve(ihs(X)+1-log(2), add=TRUE, col=adjustcolor(2, 1/2), lwd=2)
curve(ihs(X), add=TRUE, col=adjustcolor(4, 1/2), lwd=2)
## for x >= 0, u.log(x) is nicely between IHS(x) and shifted IHS
## a log10-scale version of asinh() (aka "IHS"): ihs10(x) := asinh(x/2) / ln(10)
ihs10 <- function(x) asinh(x/2)/log(10)
xyaxis <- function() abline(h=0, v=0, col = "gray20", lty = 3)
leg3 <- function(x = "right")
legend(x, legend = c(quote(ihs10(x) == asinh(x/2)/log(10)),
quote(log[10](1+x)), quote(log[10](x))),
col=c(1,2,5), bty="n", lwd=2)
curve(asinh(x/2)/log(10), -5, 20, n=1000, lwd=2); xyaxis()
curve(log10(1+x), col=2, lwd=2, add=TRUE)
curve(log10( x ), col=5, lwd=2, add=TRUE); leg3()
```

```r
## zoom out and x-log-scale
curve(asinh(x/2)/log(10), .1, 100, log="x", n=1000); xyaxis()
curve(log10(1+x), col=2, add=TRUE)
curve(log10( x ), col=5, add=TRUE); leg3("center")
```

```r
curve(abs(1 - ihs10(x) / log10(1+x)), .1, 5000, col=2, log = "xy", ylim = c(-1,1)*0.10,
main = "absolute difference", xaxt="n"); xyaxis(); eaxis(1, sub10=1)
```

```r
## Compare with Stahel's version of "started log"
## (here, for *vectors* only, and 'base', as DescTools::LogSt();
## by MM: "modularized" by providing a threshold-computer function separately:
logst_thrWS <- function(x, mult = 1) {
  lq <- quantile(x[x > 0], probs = c(0.25, 0.75), na.rm = TRUE, names = FALSE)
  ...}
```

```r
```
if (lq[1] == lq[2])
  lq[1] <- lq[2]/2
}

logst0L <- function(x, calib = x, threshold = thrFUN(calib, mult=mult),
  thrFUN = logst_thrWS, mult = 1, base = 10)
{
  res <- x # incl NA's
  notNA <- !is.na(sml <- (x < (th <- threshold)))
  i1 <- sml & notNA; res[i1] <- log(th, base) + ((x[i1] - th)/(th * log(base)))
  i2 <- !sml & notNA; res[i2] <- log(x[i2], base)
  attr(res, "threshold") <- th
  attr(res, "base") <- base
  res
}

logst0 <- function(x, calib = x, threshold = thrFUN(calib, mult=mult),
  thrFUN = logst_thrWS, mult = 1, base = 10)
{
  res <- x # incl NA's
  notNA <- !is.na(sml <- (x < (th <- threshold)))
  i1 <- sml & notNA; res[i1] <- log(th, base) + ((x[i1] - th)/(th * log(base)))
  i2 <- !sml & notNA; res[i2] <- log(x[i2], base)
  attr(res, "threshold") <- th
  attr(res, "base") <- base
  res
}

## u.log() is really using natural log() -- whereas logst() defaults to base=10
curve(u.log, -4, 10, n=1000); abline(h=0, v=0, col = "gray20", lty = 3); points(-1:1, -1:1, pch=3)
curve(log10(x) + 1, add=TRUE, col=adjcolor("midnightblue", 1/2), lwd=4, lty=6)
curve(log10(x), add=TRUE, col=adjcolor("skyblue3", 1/2), lwd=4, lty=7)
curve(logst0(x, threshold= 2 ), add=TRUE, col=adjcolor("orange",1/2), lwd=2)
curve(logst0(x, threshold= 1 ), add=TRUE, col=adjcolor(2, 1/2), lwd=2)
curve(logst0(x, threshold= 1/4), add=TRUE, col=adjcolor(3, 1/2), lwd=2, lty=2)
curve(logst0(x, threshold= 1/8), add=TRUE, col=adjcolor(4, 1/2), lwd=2, lty=2)

---

**Description**

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

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

Usage

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

Sys.ps.cmd()

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!; on non-Windows, Sys.ps() which makes use of Sys.ps.cmd().

Examples

u.sys # shows how simply the function is defined :

## Not run:
function (...,
intern = TRUE)
system(paste(...,
sep = ""),intern = intern)

## End(Not run)

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

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.

\(\text{round.dig}\) 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 \( \text{Var}(1, 2, ..., n) = n(n + 1)/12 \).

Author(s)

Martin Maechler, ca 1990

See Also

\texttt{runif} for producing uniform random numbers.

Examples

\begin{verbatim}
(u <- unif(8))
var(u)

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

\texttt{uniqueL}

A Reversible Version of unique()

Description

A version of \texttt{unique} keeping enough information to reverse (or invert) to the original data.

Usage

\texttt{uniqueL(x, isuniq = !duplicated(x), need.sort = is.unsorted(x))}

Arguments

\begin{verbatim}
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 \textit{must be set correctly}.
\end{verbatim}
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

```r
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

```r
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

```r
vcat := function(vec, sep = " ") paste(vec, collapse = sep)
ccat := function(...) paste(..., collapse = ",", sep = """)
```
wrapFormula

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

```r
ch <- "is"
cat("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 ~ .`.

The potential use is slightly more general.

Usage

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

Arguments

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

Value

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

Note

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

Produce Regular Grid Matrix

Description

Produce the grid used by persp, contour, etc, as an N x 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 length(x) * length(y) rows.

Author(s)

**xy.unique.x**

*Uniqify (X,Y) Values using Weights*

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Author(s)**

Martin Maechler, 8 Mar 1993.

**See Also**

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

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

* Quasi Monte Carlo
  QUnif, 90
* algebra
  nearcor, 55
  posdefify, 77
* aplot
  eaxis, 23
  linesHyperb.lm, 44
  p.arrows, 58
  p.hboxp, 61
* apportionment
  roundfixS, 97
* arithmetic
  primes, 82
* arith
  digitsBase, 20
  inv.seq, 39
  is.whole, 40
  iterate.lin.recursion, 41
  nr.sign.chg, 57
  polyn.eval, 76
  relErr, 93
  roundfixS, 97
  signi, 105
  u.log, 120
  unif, 123
* array
  col01scale, 10
  diagX, 19
  empty.dinames, 29
  mpl, 51
  nearcor, 55
  posdefify, 77
  xy.grid, 127
* category
  tapplySimpl, 112
* character
  repChar, 94
* classif
  diagDA, 17
* datagen
  QUnif, 90
* datasets
  potatoes, 79
  str_data, 107
* data
  funEnv, 33
* debugging
  prt.DEBUG, 85
* device
  cairoSwd, 8
  ps.end, 85
  ps.latex, 87
* distribution
  KSd, 42
* documentation
  Deprecated, 17
  str_data, 107
* dplot
  axTexpr, 6
  p.scales, 65
  pretty10exp, 80
  u.boxplot.x, 116
* dynamic
  loessDemo, 46
  tkdensity, 113
* environment
  u.assign0, 116
  u.sys, 122
* file
  read.org.table, 91
  sourceAttach, 106
* hplot
  compresid2way, 11
  cum.Vert.funkt, 12
  ecdf.ksCI, 26
  errbar, 30
  histBxp, 36
loessDemo, 46
mpl, 51
mult.fig, 52
n.plot, 54
p.datum, 59
p.dnorm, 60
p.profileTraces, 62
p.res.2fact, 63
p.res.2x, 64
p.tachoPlot, 66
p.ts, 68
plotDS, 72
plotStep, 74
tkdensity, 113
* htest
  f.robftest, 31
* interface
  mat2tex, 48
  pkgLibs, 71
* iplot
  ellipsePoints, 27
* iteration
  tappylySimpl, 112
* loess
  loessDemo, 46
* low discrepancy sequence
  QUnif, 90
* manip
  AsciiToInt, 5
  Duplicated, 22
  last, 43
  lseq, 48
  rot2, 95
  rotn, 96
  roundfixS, 97
  seqXtend, 100
* math
  factorize, 32
  integrate.xy, 38
  primes, 82
  QUnif, 90
  rot2, 95
* misc
  sessionInfoX, 102
  toLatex.numeric, 114
* models
  diagDA, 17
  TA.plot, 110
  wrapFormula, 126
* multivariate
  QUnif, 90
* naïve Bayes classifier
  diagDA, 17
* nonlinear
  p.profileTraces, 62
* nonparametric
  plotStep, 74
* print
  empty.dimnames, 29
  helppdf, 35
  vcat, 125
* programming
  funEnv, 33
  missingCh, 50
* regression
  hatMat, 34
  linesHyperb.lm, 44
  p.res.2fact, 63
  p.res.2x, 64
  TA.plot, 110
* robust
  f.robftest, 31
  rrange, 99
* smooth
  D1D2, 13
  D2ss, 15
  hatMat, 34
* space filling
  QUnif, 90
* ts
  p.ts, 68
* univar
  rrange, 99
* utilities
  diagX, 19
  digitsBase, 20
  ellipsePoints, 27
  empty.dimnames, 29
  helppdf, 35
  integrate.xy, 38
  inv.seq, 39
  list_named, 45
  mat2tex, 48
  n.code, 53
  p.dnorm, 60
  paste.vec, 69
printTable2, 83
quadrant, 89
read.org.table, 91
relErr, 93
seqXtend, 100
shortRversion, 104
sourceAttach, 106
str_data, 107
Sys.cpuinfo, 108
Sys.ps, 109
u.assign0, 116
u.boxplot.x, 116
u.date, 117
u.Datumvonheute, 119
u.sys, 122
unif, 123
uniqueL, 124
vcat, 125
xy.unique.x, 128
```
.C, 72
.Call, 72
.Machine, 103
.Platform, 102
.libPaths, 70, 102, 103
::, 39
```
abline, 111
adjboxStats, 100
all.equal.numeric, 93
aov, 111
array, 29, 93, 112
arrows, 58, 59
as.character, 126
as.integer, 21
as.integer_basedInt (digitsBase), 20
AsciitoInt, 5
assign, 116
attach, 106
axis, 6, 7, 23–25, 81
axis.POSIXct, 68
axExpr, 6, 25, 81
axTicks, 7, 24, 25, 81
barplot, 37
bi2int (digitsBase), 20
bl.string (repChar), 94
boxplot, 37, 61, 63, 99, 100, 117
boxplot.stats, 99, 100
C.Monatsname (u.Datumvonheute), 119
C.weekday (u.Datumvonheute), 119
C.Wochentag (u.Datumvonheute), 119
C.Wochentagkurz (u.Datumvonheute), 119
cairo_pdf, 8
cairoSwd, 8
call, 39
capabilities, 103
capture.and.write, 9
capture.output, 9
cat, 85, 107, 126
ccat (vcat), 125
chars8bit (AsciitoInt), 5
code2n (n.code), 53
col01scale, 10
colcenter (col01scale), 10
complex, 76
compress2way, 11
contour, 127
cor, 55
cum.Vert.funkt, 12
curve, 60
D1D2, 13, 16
D1ss (D2ss), 15
D1tr (D2ss), 15
D2ss, 14, 15
data, 107
data.frame, 54, 92, 126
date, 59
DateTimeClasses, 68, 118
dchisq, 60
dDA (diagDA), 17
density, 113, 114
Deprecated, 17
dev.off, 85
dev.print, 88
dgamma, 60
diag, 20, 78
diagDA, 17
diagX, 19
digitsBase, 20
dimnames, 84
dnorm, 60
INDEX

Duplicated, 22, 125
duplicated, 22, 23, 108, 125
dyn.load, 72

eaxis, 7, 23, 81
ecdf, 27
ecdf.ksCI, 26, 43
eigen, 56, 77, 78
ellipsePoints, 27
ellipsoidhull, 28
ellipsoidPoints, 28
empty.dimnames, 29
environment, 33, 50
errbar, 30, 30
eval, 39
expand.grid, 127, 128
expression, 23, 39, 81, 115
extSoftVersion, 103

f.robftest, 31
factor, 23, 65
factorize, 32, 32, 83
file, 92
first, 44
fivenum, 100
format, 69, 115, 126
formula, 33, 64, 126, 127
ftable, 84
function, 33, 107, 128
funEnv, 33

gam, 126, 127
get, 116
getLoadedDLLs, 72
getOption, 88
glm, 111
grSoftVersion, 103

hatMat, 34
head, 9
help, 35, 36
helppdf, 35
hetcor, 55, 56
hist, 37
histBxp, 36, 61

ichar (AsciiToInt), 5
integer, 5, 21, 54, 82
integrate, 38
integrate.xy, 38
interaction.plot, 12
inv.seq, 39
invisible, 35, 107
is.integer, 40
is.whole, 40
isprime, 83
isRshared (sessionInfoX), 102
iterate.lin.recursion, 41

Ksd, 26, 27, 42

La_version, 103
last, 43
latex, 49
layout, 53
lda, 18
length, 5, 93, 100

LETTERS, 97
letters, 97
library.dynam, 72
lines, 44
linesHyperb.lm, 44
list, 5, 21, 24, 32, 45, 53, 54, 56, 70, 72, 102–104, 107
list.files, 102
list2env, 33
list_(list_named), 45
list_named, 45
lm, 44, 64, 111, 126
loess, 46, 47
loessDemo, 46
logical, 24, 35, 50, 82, 92, 102, 103, 107
LogSt, 120
logst, 120
lseq, 48

maintainer, 92
make.names, 108
margin2table (printTable2), 83
mat2tex, 48
match, 22, 23
matplot, 51
matrix, 21
max, 82
message, 92
missing, 50
missingCh, 50
month.name, 119
mpl, 51
mtext, 52, 59
mult.fig, 52, 68, 69
n.code, 53
n.plot, 54, 111
n2mfrow, 52
NA, 99
naiveBayes, 18
names, 5, 45, 70, 72
nchar, 95
nearcor, 55, 78
nearPD, 55, 56, 78
new.env, 33
nextprime, 83
nls, 62
normalizePath, 103
nr.sign.chg, 57
numeric, 48
options, 49, 85
p.arrows, 58
p.datum, 13, 59, 119
p.dchisq (p.dnorm), 60
p.dgamma (p.dnorm), 60
p.dnorm, 60
p.hboxp, 61
p.m(mpl), 51
p.profileTraces, 62
p.res.2fact, 63, 65
p.res.2x, 63, 64
p.scales, 65
p.tachoPlot, 66
p.ts, 68
packageDescription, 70, 102, 103
par, 24, 25, 30, 37, 52, 53, 55, 62, 66–68, 73, 75, 88
paste, 95, 123, 125, 126
paste.vec, 69
pcre_config, 103
pdf, 8, 85, 87, 88
pdf.do, 86, 88
pdf.do (ps.latex), 87
pdf.end, 87
pdf.end (ps.end), 85
pdf.latex, 85
df.latex (ps.latex), 87
persp, 127
pkgBuilt (pkgDesc), 70
pkgDesc, 70
pkgLibs, 71
plot, 54, 55, 67, 69, 73, 75
plot.default, 30, 47, 55, 63, 64, 67, 68, 73, 113
plot.ecdf, 12, 13, 75
plot.lm, 63, 65, 111
plot.mts, 51
plot.stepfun, 27, 75
plot.ts, 68, 69
plotDS, 72, 101
plotmath, 23, 25, 81
plotStep, 13, 74
pmax, 17
pmax.sa (Deprecated), 17
pmin, 17
pmin.sa (Deprecated), 17
points, 47
polygon, 47
polyln.eval, 76
posdefify, 56, 77
postscript, 85–88
potatoes, 79
predict, 18
predict.dDA (diagDA), 17
predict.lm, 44, 45
predict.polynomial, 76
pretty10exp, 7, 23–25, 80, 115
primes, 32, 82
print, 18, 21, 70, 102, 111, 115
print.basedInt (digitsBase), 20
print.dDA (diagDA), 17
print.default, 84
print.margin2table (printTable2), 83
print.sessionInfo, 102
print.sessionInfoX (sessionInfoX), 102
printTable2, 83
proc.time, 110
profile, 62
profile.nls, 62
prt.DEBUG, 85
ps.do, 85, 86, 88
ps.do (ps.latex), 87
ps.end, 85, 88
ps.latex, 87
ps.options, 88
qda, 18
INDEX
135
quadrant, 89
QUnif, 90
R.version, 103, 104
R.version.string, 104
range, 99, 100
read.org.table, 91
read.table, 91, 92
readLines, 92
relErr, 93
relErrV(relErr), 93
repChar, 94
rlm, 31
r2, 95, 97
rotn, 96
round, 98, 105, 115
roundfixS, 97, 101
rrange, 99
rug, 37, 68, 113
runif, 124
RweaveLatex, 9
scale, 10
scan, 109
scat1d, 37
segments, 58, 64, 75
seq, 41, 48, 100, 101
seqXtend, 73, 100
sessionInfo, 102, 103
sessionInfoX, 102
sHalton(QUnif), 90
shortRversion, 104
signi, 105
signif, 81, 105, 115
smooth.spline, 13–16, 34, 128
source, 106
sourceAttach, 106
split, 23
str, 107
str_data, 107
strcodes(AsciiToInt), 5
strrep, 94
summary, 31
summary.aov, 31
summary.rlm, 31
Sweave, 8
symbols, 67
sys.call, 45
Sys.cpuinfo, 108
Sys.getenv, 103
Sys.getlocale, 5
Sys.getpid, 110
Sys.info, 102, 110
Sys.memGB(Sys.cpuinfo), 108
Sys.meminfo(Sys.cpuinfo), 108
Sys.MIPS(Sys.cpuinfo), 108
Sys.procinfo(Sys.cpuinfo), 108
Sys.ps, 109, 109
Sys.ps.cmd(u.sys), 122
Sys.setlocale, 5
Sys.sizes(Sys.ps), 109
sys.source, 106
system, 36, 71, 109, 123
TA.plot, 54, 63, 65, 110
table, 84
tail, 9, 43
tapply, 23, 112, 113
tapplySimpl, 112
tc1Version, 103
text, 54, 55
textConnection, 92
title, 11, 27, 64
tkdensity, 113
toLatex, 115
toLatex.numeric, 81, 114
TRUE, 85
turnogram, 43, 44
u.assign0, 116
u.boxplot.x, 63, 116
u.date, 59, 117, 118, 119
u.datumdecode, 118
u.Datumvonheute, 59, 117, 119
u.get0(u.assign0), 116
u.log, 120
u.sys, 122
unif, 123
unique, 101, 124, 125, 128
uniquel, 23, 124
unname, 29
vcat, 125
wrapFormula, 126
x11, 85
xy.coords, 54, 73, 89, 128
xy.grid, 127
xy.unique.x, 128