The Hmisc Package

February 16, 2008

Version 3.4-3

Date 2007-10-31

Title Harrell Miscellaneous

Author Frank E Harrell Jr <f.harrell@vanderbilt.edu>, with contributions from many other users.

Maintainer Charles Dupont <charles.dupont@vanderbilt.edu>

Depends R (>= 2.4.0), methods

Imports lattice, cluster

Suggests lattice, grid, nnet, foreign, survival, chron, acepack, TeachingDemos, Design, cluster

Description The Hmisc library contains many functions useful for data analysis, high-level graphics, utility operations, functions for

License GPL version 2 or newer

LazyLoad no

URL http://biostat.mc.vanderbilt.edu/s/Hmisc, http://biostat.mc.vanderbilt.edu/twiki/pub/Main/RS/sintro.pdf, http://biostat.mc.vanderbilt.edu/twiki/pub/Main/StatReport/summary.pdf, http://biostat.mc.vanderbilt.edu/trac/Hmisc

R topics documented:

`Cs` *Character strings from unquoted names*

Description

Makes a vector of character strings from a list of valid S names

Usage

```
Cs(...)
```

Arguments

`...`	any number of names separated by commas

Value

character string vector

See Also

sys.frame, deparse

Examples

```
Cs(a,cat,dog)
# subset.data.frame <- dataframe[,Cs(age,sex,race,bloodpressure,height)]
```

`Ecdf` *Empirical Cumulative Distribution Plot*

Description

Computes coordinates of cumulative distribution function of x, and by defaults plots it as a step function. A grouping variable may be specified so that stratified estimates are computed and (by default) plotted. If there is more than one group, the `labcurve` function is used (by default) to label the multiple step functions or to draw a legend defining line types, colors, or symbols by linking them with group labels. A `weights` vector may be specified to get weighted estimates. Specify `normwt` to make `weights` sum to the length of `x` (after removing NAs). Other wise the total sample size is taken to be the sum of the weights.

`Ecdf` is actually a method, and `Ecdf.default` is what's called for a vector argument. `Ecdf.data.frame` is called when the first argument is a data frame. This function can automatically set up a matrix of ECDFs and wait for a mouse click if the matrix requires more than one page. Categorical variables, character variables, and variables having fewer than a set number of unique values are ignored. If `par(mfrow=..)` is not set up before `Ecdf.data.frame` is called, the function will try to figure the best layout depending on the number of variables in the data frame. Upon return the original `mfrow` is left intact.

When the first argument to `Ecdf` is a formula, a Trellis/Lattice function `Ecdf.formula` is called. This allows for multi-panel conditioning, superposition using a `groups` variable, and other Trellis features, along with the ability to easily plot transformed ECDFs using the `fun` argument. For example, if `fun=qnorm`, the inverse normal transformation will be used for the y-axis. If the transformed curves are linear this indicates normality. Like the `xYplot` function, `Ecdf` will create a function `Key` if the `groups` variable is used. This function can be invoked by the user to define the keys for the groups.

Usage

```
Ecdf(x, ...)

## Default S3 method:
Ecdf(x, what=c('F','1-F','f'), weights, normwt=FALSE,
     xlab, ylab, q, pl=TRUE, add=FALSE, lty=1,
     col=1, group=rep(1,length(x)), label.curves=TRUE, xlim,
     subtitles=TRUE, datadensity=c('none','rug','hist','density'),
     side=1,
     frac=switch(datadensity,none=NA,rug=.03,hist=.1,density=.1),
     dens.opts=NULL, lwd, ...)

## S3 method for class 'data.frame':
Ecdf(x, group=rep(1,nrows), weights, normwt,
     label.curves=TRUE, n.unique=10, na.big=FALSE, subtitles=TRUE,
     vnames=c('labels','names'),...)

## S3 method for class 'formula':
Ecdf(x, data=sys.frame(sys.parent()), groups=NULL,
     prepanel=prepanel.Ecdf, panel=panel.Ecdf, ..., xlab,
     ylab, fun=function(x)x, subset=TRUE)
```

Arguments

`x`	a numeric vector, data frame, or Trellis/Lattice formula
`what`	The default is `"F"` which results in plotting the fraction of values <= `x`. Set to `"1-F"` to plot the fraction > x or `"f"` to plot the cumulative frequency of values <= x.
`weights`	numeric vector of weights. Omit or specify a zero-length vector or NULL to get unweighted estimates.
`normwt`	see above
`xlab`	x-axis label. Default is label(x) or name of calling argument. For `Ecdf.formula`, `xlab` defaults to the `label` attribute of the x-axis variable.
`ylab`	y-axis label. Default is `"Proportion <= x"`, `"Proportion > x"`, or "Frequency <= x" depending on value of `what`.
`q`	a vector for quantiles for which to draw reference lines on the plot. Default is not to draw any.
`pl`	set to F to omit the plot, to just return estimates.
`add`	set toTRUE to add the cdf to an existing plot.
`lty`	integer line type for plot. If `group` is specified, this can be a vector.
`lwd`	line width for plot. Can be a vector corresponding to `groups`.
`col`	color for step function. Can be a vector.
`group`	a numeric, character, or `factor` categorical variable used for stratifying estimates. If `group` is present, as many ECDFs are drawn as there are non–missing group levels.
`label.curves`	applies if more than one `group` exists. Default is `TRUE` to use `labcurve` to label curves where they are farthest apart. Set `label.curves` to a `list` to specify options to `labcurve`, e.g., `label.curves=list(method="arrow", cex=.8)`. These option names may be abbreviated in the usual way arguments are abbreviated. Use for example `label.curves=list(keys=1:5)` to draw symbols periodically (as in `pch=1:5` - see `points`) on the curves and automatically position a legend in the most empty part of the plot. Set `label.curves=FALSE` to suppress drawing curve labels. The `col`, `lty`, and `type` parameters are automatically passed to `labcurve`, although you can override them here. You can set `label.curves=list(keys="lines")` to have different line types defined in an automatically positioned key.
`xlim`	x-axis limits. Default is entire range of `x`.

`subtitles`	set to `FALSE` to suppress putting a subtitle at the bottom left of each plot. The subtitle indicates the numbers of non-missing and missing observations, which are labeled `n`, `m`.
`datadensity`	If `datadensity` is not `"none"`, either `scat1d` or `histSpike` is called to add a rug plot (`datadensity="rug"`), spike histogram (`datadensity="hist"`), or smooth density estimate (`"density"`) to the bottom or top of the ECDF.
`side`	If `datadensity` is not `"none"`, the default is to place the additional information on top of the x-axis (`side=1`). Use `side=3` to place at the top of the graph.
`frac`	passed to `histSpike`
`dens.opts`	a list of optional arguments for `histSpike`
`...`	other parameters passed to plot if add=F. For data frames, other parameters to pass to `Ecdf.default`. For `Ecdf.formula`, if `groups` is not used, you can also add data density information to each panel's ECDF by specifying the `datadensity` and optional `frac`, `side`, `dens.opts` arguments.
`n.unique`	minimum number of unique values before an ECDF is drawn for a variable in a data frame. Default is 10.
`na.big`	set to `TRUE` to draw the number of NAs in larger letters in the middle of the plot for `Ecdf.data.frame`
`vnames`	By default, variable labels are used to label x-axes. Set `vnames="names"` to instead use variable names.
`method`	method for computing the empirical cumulative distribution. See `wtd.Ecdf`. The default is to use the standard `"i/n"` method as is used by the non-Trellis versions of `Ecdf`.
`fun`	a function to transform the cumulative proportions, for the Trellis-type usage of `Ecdf`
`data`	
`groups`	
`subset`	
`prepanel`	
`panel`	the usual Trellis/Lattice parameters, with `groups` causing `Ecdf.formula` to overlay multiple ECDFs on one panel.

Value

for `Ecdf.default` an invisible list with elements x and y giving the coordinates of the cdf. If there is more than one `group`, a list of such lists is returned. An attribute, `N`, is in the returned object. It contains the elements `n` and `m`, the number of non-missing and missing observations, respectively.

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`wtd.Ecdf`, `label`, `table`, `cumsum`, `labcurve`, `xYplot`, `histSpike`

Examples

```
set.seed(1)
ch <- rnorm(1000, 200, 40)
Ecdf(ch, xlab="Serum Cholesterol")
scat1d(ch)                        # add rug plot
histSpike(ch, add=TRUE, frac=.15)   # add spike histogram
# Better: add a data density display automatically:
Ecdf(ch, datadensity='density')

label(ch) <- "Serum Cholesterol"
Ecdf(ch)
other.ch <- rnorm(500, 220, 20)
Ecdf(other.ch,add=TRUE,lty=2)

sex <- factor(sample(c('female','male'), 1000, TRUE))
Ecdf(ch, q=c(.25,.5,.75))  # show quartiles
Ecdf(ch, group=sex,
     label.curves=list(method='arrow'))

# Example showing how to draw multiple ECDFs from paired data
pre.test <- rnorm(100,50,10)
post.test <- rnorm(100,55,10)
x <- c(pre.test, post.test)
g <- c(rep('Pre',length(pre.test)),rep('Post',length(post.test)))
Ecdf(x, group=g, xlab='Test Results', label.curves=list(keys=1:2))
# keys=1:2 causes symbols to be drawn periodically on top of curves

# Draw a matrix of ECDFs for a data frame
m <- data.frame(pre.test, post.test,
                sex=sample(c('male','female'),100,TRUE))
Ecdf(m, group=m$sex, datadensity='rug')

freqs <- sample(1:10, 1000, TRUE)
Ecdf(ch, weights=freqs)  # weighted estimates

# Trellis/Lattice examples:

region <- factor(sample(c('Europe','USA','Australia'),100,TRUE))
year <- factor(sample(2001:2002,1000,TRUE))
Ecdf(~ch | region*year, groups=sex)
Key()            # draw a key for sex at the default location
# Key(locator(1)) # user-specified positioning of key
```

```
age <- rnorm(1000, 50, 10)
Ecdf(~ch | equal.count(age), groups=sex)  # use overlapping shingles
Ecdf(~ch | sex, datadensity='hist', side=3)  # add spike histogram at top
```

`Hmisc-internal` *Internal Hmisc functions*

Description

Internal Hmisc functions.

Details

These are not to be called by the user or are undocumented.

`Lag` *Lag a Numeric, Character, or Factor Vector*

Description

Shifts a vector `shift` elements later. Character or factor variables are padded with `""`, numerics with `NA`.

Usage

```
Lag(x, shift = 1)
```

Arguments

`x`	a vector
`shift`	positive integer specifying the number of observations to be shifted to the right

Details

A.ttributes of the original object are carried along to the new lagged one, but factor vectors are converted to character.

Value

a vector like `x`

Author(s)

Frank Harrell

See Also

`lag`

Examples

```
Lag(1:5,2)
Lag(letters[1:4],2)
Lag(factor(letters[1:4]),2)
# Find which observations are the first for a given subject
id <- c('a','a','b','b','b','c')
id != Lag(id)
!duplicated(id)
```

`Misc` *Miscellaneous Functions*

Description

This documents miscellaneous small functions in Hmisc that may be of interest to users.

`clowess` runs `lowess` but if the `iter` argument exceeds zero, sometimes wild values can result, in which case `lowess` is re-run with `iter=0`.

`confbar` draws multi-level confidence bars using small rectangles that may be of different colors.

`getLatestSource` fetches and `source`s the most recent source code for functions in packages in the Vanderbilty University CVS repository.

`inverseFunction` generates a function to find all inverses of a monotonic or nonmonotonic function that is tabulated at vectors (x,y), typically 1000 points. If the original function is monotonic, simple linear interpolation is used and the result is a vector, otherwise linear interpolation is used within each interval in which the function is monotonic and the result is a matrix with number of columns equal to the number of monotonic intervals. If a requested y is not within any interval, the extreme x that pertains to the nearest extreme y is returned. Specifying what='sample' to the returned function will cause a vector to be returned instead of a matrix, with elements taken as a random choice of the possible inverses.

`james.stein` computes James-Stein shrunken estimates of cell means given a response variable (which may be binary) and a grouping indicator.

`km.quick` provides a fast way to invoke `survfit.km` in the `survival` package to get Kaplan-Meier estimates for a single stratum for a vector of time points (if `times` is given) or to get a vector of survival time quantiles (if `q` is given).

`lm.fit.qr.bare` is a fast stripped-down function for computing regression coefficients, residuals, R^2, and fitted values. It uses the Fortran routines `dqrls`.

`matxv` multiplies a matrix by a vector, handling automatic addition of intercepts if the matrix does not have a column of ones. If the first argument is not a matrix, it will be converted to one.

`nomiss` returns a data frame (if its argument is one) with rows corresponding to `NA`s removed, or it returns a matrix with rows with any element missing removed.

`outerText` uses `text()` to put test strings in left or right margins. It temporarily sets `par(xpd=NA)` if using R.

`sepUnitsTrans` converts character vectors containing values such as `c("3 days","3day","4month","2 years","2weeks","7")` to numeric vectors (here `c(3,3,122,730,14,7))` in a flexible fashion. The user can specify a vector of units of measurements and conversion factors. The units with a conversion factor of `1` are taken as the target units, and if those units are present in the character strings they are ignored. The target units are added to the resulting vector as the `"units"` attribute.

`trap.rule` computes the area under a curve using the trapezoidal rule, assuming `x` is sorted.

`trellis.strip.blank` sets up Trellis or Lattice graphs to have a clear background on the strips for panel labels.

`under.unix` is a scalar logical value that is `TRUE` if you are running Linux or Unix.

`.R.` is a logical value set to `TRUE` if running R, `FALSE` for S-Plus.

`.SV4.` is a logical value set to `TRUE` if running version 4 of the S language under S-Plus, `FALSE` otherwise.

`unPaste` provides a version of the S-Plus `unpaste` that works for R and S-Plus.

`whichClosePW` is a very fast function using weighted multinomial sampling to determine which element of a vector is "closest" to each element of another vector. `whichClosest` quickly finds the closest element without any randomness.

`xless` is a function for Linux/Unix users to invoke the system `xless` command to pop up a window to display the result of `print`ing an object. For Windows, `xless` uses the builtin `page` function but with better defaults.

Usage

```
confbar(at, est, se, width, q = c(0.7, 0.8, 0.9, 0.95, 0.99),
        col = if (.R.) gray(c(0, 0.25, 0.5, 0.75, 1)) else
              if (under.unix) c(1, 0.8, 0.5, 0.2, 0.065) else
              c(1, 4, 3, 2, 5),
        type = c("v", "h"), labels = TRUE, ticks = FALSE,
        cex = 0.5, side = "l", lwd = 5, clip = c(-1e+30, 1e+30),
        fun = function(x) x,
        qfun = function(x) ifelse(x == 0.5, qnorm(x),
                             ifelse(x < 0.5, qnorm(x/2),
                             qnorm((1 +  x)/2))))
getLatestSource(x=NULL, package='Hmisc', recent=NULL, avail=FALSE,
                type=c('svn','cvs'))

inverseFunction(x, y)
james.stein(y, group)
km.quick(S, times, q)
lm.fit.qr.bare(x, y, tolerance, intercept=TRUE, xpxi=FALSE)
matxv(a, b, kint=1)
nomiss(x)
outerText(string, y, setAside=string[1], side=4, space=1,
          adj=1, cex=par('cex'))
```

```
sepUnitsTrans(x, conversion=c(day=1, month=365.25/12, year=365.25, week=7),
              round=FALSE, digits=0)
trap.rule(x, y)
trellis.strip.blank()
unPaste(str, sep="/", extended=FALSE)
whichClosest(x, w)
whichClosePW(x, w, f=0.2)
xless(x, ..., title)
```

Arguments

`a`	a numeric matrix or vector
`adj`	0 for left justification, 0.5 for centered, 1 for right
`at`	x-coordinate for vertical confidence intervals, y-coordinate for horizontal
`avail`	set to `TRUE` to have `getLatestSource` return a data frame of available files and latest versions instead of fetching any
`b`	a numeric vector
`cex`	character expansion factor
`clip`	interval to truncate limits
`col`	vector of colors
`conversion`	a named numeric vector
`digits`	number of digits used for `round`
`est`	vector of point estimates for confidence limits
`extended`	see `strsplit` in R
`f`	a scaling constant
`fun`	function to transform scale
`group`	a categorical grouping variable
`intercept`	set to `FALSE` to not automatically add a column of ones to the `x` matrix
`kint`	which element of `b` to add to the result if `a` does not contain a column for intercepts
`labels`	set to `FALSE` to omit drawing confidence coefficients
`lwd`	line widths
`package`	name of package for `getLatestSource`, default is `'Hmisc'`
`q`	vector of confidence coefficients or quantiles
`qfun`	quantiles on transformed scale
`recent`	an integer telling `getLatestSource` to get the `recent` most recently modified files from the package
`round`	set to `TRUE` to round converted values
`S`	a `Surv` object
`se`	vector of standard errors
`sep`	a single character string specifying the delimiter

`setAside`	for `adj=1 side=4`, is a character string used to determine the space to set aside for all strings.
`side`	for `confbar` is `"b"`, `"l"`, `"t"`, `"r"` for bottom, left, top, right. For `outText` is the using integers 1-4 corresponding to these.
`space`	the number of extra characters to leave to the left of the string(s) (`adj=0`) or to the right (`adj=1`)
`str`	a character string vector
`string`	a character string vector
`ticks`	set to `TRUE` to draw lines between rectangles
`times`	a numeric vector of times
`title`	a character string to title a window or plot
`tolerance`	tolerance for judging singularity in matrix
`type`	`"v"` for vertical, `"h"` for horizontal. For `getLatestSource` this specifies the type of source code repository, `'svn'` (the default) or `'cvs'`, which is now outdated as Subversion has replaced CVS in the Vanderbilt Biostatistics server.
`w`	a numeric vector
`width`	width of confidence rectanges in user units
`x`	a numeric vector (matrix for `lm.fit.qr.bare`) or data frame. For `xless` may be any object that is sensible to `print`. For `sepUnitsTrans` is a character or factor variable. For `getLatestSource` is a character string or vector of character strings containing base file names to retrieve from CVS. Set `x='all'` to retrieve all source files. For `clowess`, `x` may also be a list with x and y components. For `inverseFunction`, `x` and `y` contain evaluations of the function whose inverse is needed. `x` is typically an equally-spaced grid of 1000 points.
`xpxi`	set to `TRUE` to add an element to the result containing the inverse of $X'X$
`y`	a numeric vector. For `inverseFunction` `y` is the evaluated function values at `x`.
`...`	arguments passed through to another function

Author(s)

Frank Harrell

Examples

```
trap.rule(1:100,1:100)

unPaste(c('a;b or c','ab;d','qr;s'), ';')

sepUnitsTrans(c('3 days','4 months','2 years','7'))

set.seed(1)
whichClosest(1:100, 3:5)
whichClosest(1:100, rep(3,20))

whichClosePW(1:100, rep(3,20))
whichClosePW(1:100, rep(3,20), f=.05)
whichClosePW(1:100, rep(3,20), f=1e-10)

x <- seq(-1, 1, by=.01)
y <- x^2
h <- inverseFunction(x,y)
formals(h)$turns   # vertex
a <- seq(0, 1, by=.01)
plot(0, 0, type='n', xlim=c(-.5,1.5))
lines(a, h(a)[,1])            ## first inverse
lines(a, h(a)[,2], col='red') ## second inverse
a <- c(-.1, 1.01, 1.1, 1.2)
points(a, h(a)[,1])

## Not run:
getLatestSource(recent=5)  # source() most recent 5 revised files in Hmisc
getLatestSource('cut2')    # fetch and source latest cut2.s
getLatestSource('all')     # get everything
getLatestSource(avail=TRUE) # list available files and latest versions
## End(Not run)
```

`Overview` *Overview of Hmisc Library*

Description

The Hmisc library contains many functions useful for data analysis, high-level graphics, utility operations, functions for computing sample size and power, translating SAS datasets into S, imputing missing values, advanced table making, variable clustering, character string manipulation, conversion of S objects to LaTeX code, recoding variables, and bootstrap repeated measures analysis. Most of these functions were written by F Harrell, but a few were collected from statlib and from s-news; other authors are indicated below. This collection of functions includes all of Harrell's submissions to statlib other than the functions in the Design and display libraries. A few of the functions do not have "Help" documentation.

To make Hmisc load silently, issue `options(Hverbose=FALSE)` before `library(Hmisc)`.

Functions

Function Name	**Purpose**
abs.error.pred	Computes various indexes of predictive accuracy based on absolute errors, for linear models
all.is.numeric	Check if character strings are legal numerics
approxExtrap	Linear extrapolation
aregImpute	Multiple imputation based on additive regression, bootstrapping, and predictive mean matching

areg.boot	Nonparametrically estimate transformations for both sides of a multiple additive regression, and bootstrap these estimates and R^2
ballocation	Optimum sample allocations in 2-sample proportion test
binconf	Exact confidence limits for a proportion and more accurate (narrower!) score stat.-based Wilson interval (Rollin Brant, mod. FEH)
bootkm	Bootstrap Kaplan-Meier survival or quantile estimates
bpower	Approximate power of 2-sided test for 2 proportions Includes bpower.sim for exact power by simulation
bpplot	Box-Percentile plot (Jeffrey Banfield, ⟨umsfjban@bill.oscs.montana.edu⟩)
bsamsize	Sample size requirements for test of 2 proportions
bystats	Statistics on a single variable by levels of >=1 factors
bystats2	2-way statistics
calltree	Calling tree of functions (David Lubinsky, ⟨david@hoqax.att.com⟩)
character.table	Shows numeric equivalents of all latin characters Useful for putting many special chars. in graph titles (Pierre Joyet, ⟨pierre.joyet@bluewin.ch⟩)
ciapower	Power of Cox interaction test
cleanup.import	More compactly store variables in a data frame, and clean up problem data when e.g. Excel spreadsheet had a non-numeric value in a numeric column
combine.levels	Combine infrequent levels of a categorical variable
comment	Attach a comment attribute to an object: comment(fit) <- 'Used old data' comment(fit) (prints comment)
confbar	Draws confidence bars on an existing plot using multiple confidence levels distinguished using color or gray scale
contents	Print the contents (variables, labels, etc.) of a data frame
cpower	Power of Cox 2-sample test allowing for noncompliance
Cs	Vector of character strings from list of unquoted names
csv.get	Enhanced importing of comma separated files labels
cut2	Like cut with better endpoint label construction and allows construction of quantile groups or groups with given n
datadensity	Snapshot graph of distributions of all variables in a data frame. For continuous variables uses scat1d.
dataRep	Quantify representation of new observations in a database
ddmmmyy	SAS "date7" output format for a chron object
deff	Kish design effect and intra-cluster correlation
describe	Function to describe different classes of objects. Invoke by saying describe(object). It calls one of the following:
describe.data.frame	Describe all variables in a data frame (generalization of SAS UNIVARIATE)
describe.default	Describe a variable (generalization of SAS UNIVARIATE)
do	Assists with batch analyses

dot.chart	Dot chart for one or two classification variables
Dotplot	Enhancement of Trellis dotplot allowing for matrix x-var., auto generation of Key function, superposition
drawPlot	Simple mouse-driven drawing program, including a function for fitting Bezier curves
Ecdf	Empirical cumulative distribution function plot
eip	Edit an object "in-place" (may be dangerous!), e.g. eip(sqrt) will replace the builtin sqrt function
errbar	Plot with error bars (Charles Geyer, U. Chi., mod FEH)
event.chart	Plot general event charts (Jack Lee, ⟨jjlee@mdanderson.org⟩, Ken Hess, Joel Dubin; Am Statistician 54:63-70,2000)
event.history	Event history chart with time-dependent cov. status (Joel Dubin, joel.dubin@yale.edu)
find.matches	Find matches (with tolerances) between columns of 2 matrices
first.word	Find the first word in an S expression (R Heiberger)
fit.mult.impute	Fit most regression models over multiple transcan imputations, compute imputation-adjusted variances and avg. betas
format.df	Format a matrix or data frame with much user control (R Heiberger and FE Harrell)
ftupwr	Power of 2-sample binomial test using Fleiss, Tytun, Ury
ftuss	Sample size for 2-sample binomial test using " " " " (Both by Dan Heitjan, ⟨dheitjan@biostats.hmc.psu.edu⟩)
gbayes	Bayesian posterior and predictive distributions when both the prior and the likelihood are Gaussian
getHdata	Fetch and list datasets on our web site
gs.slide	Sets nice defaults for graph sheets for S-Plus 2000 for copying graphs into Microsoft applications
hdquantile	Harrell-Davis nonparametric quantile estimator with s.e.
histbackback	Back-to-back histograms (Pat Burns, Salomon Smith Barney, London, ⟨pburns@dorado.sbi.com⟩)
hist.data.frame	Matrix of histograms for all numeric vars. in data frame Use hist.data.frame(data.frame.name)
histSpike	Add high-resolution spike histograms or density estimates to an existing plot
hoeffd	Hoeffding's D test (omnibus test of independence of X and Y)
impute	Impute missing data (generic method)
interaction	More flexible version of builtin function
is.present	Tests for non-blank character values or non-NA numeric values
james.stein	James-Stein shrinkage estimates of cell means from raw data
labcurve	Optimally label a set of curves that have been drawn on an existing plot, on the basis of gaps between curves. Also position legends automatically at emptiest rectangle.
label	Set or fetch a label for an S-object
Lag	Lag a vector, padding on the left with NA or ”
latex	Convert an S object to LaTeX (R Heiberger & FE Harrell)
ldBands	Lan-DeMets bands for group sequential tests
list.tree	Pretty-print the structure of any data object (Alan Zaslavsky, ⟨zaslavsk@hcp.med.harvard.edu⟩)

Load	Enhancement of `load`
mask	8-bit logical representation of a short integer value (Rick Becker)
matchCases	Match each case on one continuous variable
matxv	Fast matrix * vector, handling intercept(s) and NAs
mem	mem() types quick summary of memory used during session
mgp.axis	Version of axis() that uses appropriate mgp from mgp.axis.labels and gets around bug in axis(2, ...) that causes it to assume las=1
mgp.axis.labels	Used by survplot and plot in Design library (and other functions in the future) so that different spacing between tick marks and axis tick mark labels may be specified for x- and y-axes. ps.slide, win.slide, gs.slide set up nice defaults for mgp.axis.labels. Otherwise use mgp.axis.labels('default') to set defaults. Users can set values manually using mgp.axis.labels(x,y) where x and y are 2nd value of par('mgp') to use. Use mgp.axis.labels(type=w) to retrieve values, where w='x', 'y', 'x and y', 'xy', to get 3 mgp values (first 3 types) or 2 mgp.axis.labels.
minor.tick	Add minor tick marks to an existing plot
mtitle	Add outer titles and subtitles to a multiple plot layout
nomiss	Return a matrix after excluding any row with an NA
panel.bpplot	Panel function for trellis bwplot - box-percentile plots
panel.plsmo	Panel function for trellis xyplot - uses plsmo
pc1	Compute first prin. component and get coefficients on original scale of variables
plotCorrPrecision	Plot precision of estimate of correlation coefficient
plsmo	Plot smoothed x vs. y with labeling and exclusion of NAs Also allows a grouping variable and plots unsmoothed data
popower	Power and sample size calculations for ordinal responses (two treatments, proportional odds model)
prn	prn(expression) does print(expression) but titles the output with 'expression'. Do prn(expression,txt) to add a heading ('txt') before the 'expression' title
p.sunflowers	Sunflower plots (Andreas Ruckstuhl, Werner Stahel, Martin Maechler, Tim Hesterberg)
ps.slide	Set up postcript() using nice defaults for different types of graphics media
pstamp	Stamp a plot with date in lower right corner (pstamp()) Add ,pwd=T and/or ,time=T to add current directory name or time Put additional text for label as first argument, e.g. pstamp('Figure 1') will draw 'Figure 1 date'
putKey	Different way to use key()
putKeyEmpty	Put key at most empty part of existing plot
rcorr	Pearson or Spearman correlation matrix with pairwise deletion of missing data

rcorr.cens	Somers' Dyx rank correlation with censored data
rcorrp.cens	Assess difference in concordance for paired predictors
rcspline.eval	Evaluate restricted cubic spline design matrix
rcspline.plot	Plot spline fit with nonparametric smooth and grouped estimates
rcspline.restate	Restate restricted cubic spline in unrestricted form, and create TeX expression to print the fitted function
recode	Recodes variables
reShape	Reshape a matrix into 3 vectors, reshape serial data
rm.boot	Bootstrap spline fit to repeated measurements model, with simultaneous confidence region - least squares using spline function in time
rMultinom	Generate multinomial random variables with varying prob.
samplesize.bin	Sample size for 2-sample binomial problem (Rick Chappell, ⟨chappell@stat.wisc.edu⟩)
sas.get	Convert SAS dataset to S data frame
sasxport.get	Enhanced importing of SAS transport dataset in R
Save	Enhancement of `save`
scat1d	Add 1-dimensional scatterplot to an axis of an existing plot (like bar-codes, FEH/Martin Maechler, ⟨maechler@stat.math.ethz.ch⟩/Jens Oehlschlaegel-Akiyoshi, ⟨oehl@psyres-stuttgart.de⟩)
score.binary	Construct a score from a series of binary variables or expressions
sedit	A set of character handling functions written entirely in S. sedit() does much of what the UNIX sed program does. Other functions included are substring.location, substring<-, replace.string.wild, and functions to check if a string is numeric or contains only the digits 0-9
setpdf	Adobe PDF graphics setup for including graphics in books and reports with nice defaults, minimal wasted space
setps	Postscript graphics setup for including graphics in books and reports with nice defaults, minimal wasted space Internally uses psfig function by Antonio Possolo (⟨antonio@atc.boeing.com⟩). setps works with Ghostscript to convert .ps to .pdf
setTrellis	Set Trellis graphics to use blank conditioning panel strips, line thickness 1 for dot plot reference lines: setTrellis(); 3 optional arguments
show.col	Show colors corresponding to col=0,1,...,99
show.pch	Show all plotting characters specified by pch=. Just type show.pch() to draw the table on the current device.
showPsfrag	Use LaTeX to compile, and dvips and ghostview to display a postscript graphic containing psfrag strings
solvet	Version of solve with argument tol passed to qr
somers2	Somers' rank correlation and c-index for binary y
spearman	Spearman rank correlation coefficient spearman(x,y)

spearman.test	Spearman 1 d.f. and 2 d.f. rank correlation test
spearman2	Spearman multiple d.f. ρ^2, adjusted ρ^2, Wilcoxon-Kruskal-Wallis test, for multiple predictors
spower	Simulate power of 2-sample test for survival under complex conditions Also contains the Gompertz2,Weibull2,Lognorm2 functions.
spss.get	Enhanced importing of SPSS files using read.spss function
src	src(name) = source("name.s") with memory
store	store an object permanently (easy interface to assign function)
strmatch	Shortest unique identifier match (Terry Therneau, ⟨therneau@mayo.edu⟩)
subset	More easily subset a data frame
substi	Substitute one var for another when observations NA
summarize	Generate a data frame containing stratified summary statistics. Useful for passing to trellis.
summary.formula	General table making and plotting functions for summarizing data
symbol.freq	X-Y Frequency plot with circles' area prop. to frequency
sys	Execute unix() or dos() depending on what's running
tex	Enclose a string with the correct syntax for using with the LaTeX psfrag package, for postscript graphics
transace	ace() packaged for easily automatically transforming all variables in a matrix
transcan	automatic transformation and imputation of NAs for a series of predictor variables
trap.rule	Area under curve defined by arbitrary x and y vectors, using trapezoidal rule
trellis.strip.blank	To make the strip titles in trellis more visible, you can make the backgrounds blank by saying trellis.strip.blank(). Use before opening the graphics device.
t.test.cluster	2-sample t-test for cluster-randomized observations
uncbind	Form individual variables from a matrix
upData	Update a data frame (change names, labels, remove vars, etc.)
units	Set or fetch "units" attribute - units of measurement for var.
varclus	Graph hierarchical clustering of variables using squared Pearson or Spearman correlations or Hoeffding D as similarities Also includes the naclus function for examining similarities in patterns of missing values across variables.
xy.group	Compute mean x vs. function of y by groups of x
xYplot	Like trellis xyplot but supports error bars and multiple response variables that are connected as separate lines
win.slide	Setup win.graph or win.printer using nice defaults for presentations/slides/publications
wtd.mean	
wtd.var	
wtd.quantile	
wtd.Ecdf	
wtd.table	

wtd.rank	
wtd.loess.noiter	
num.denom.setup	Set of function for obtaining weighted estimates
zoom	Zoom in on any graphical display
	(Bill Dunlap, ⟨bill@statsci.com⟩)

Copyright Notice

If this software is used is work presented for publication, kindly reference it using for example:
Harrell FE (2004): Hmisc S function library. Programs available from `http://biostat.mc.vanderbilt.edu/s/Hmisc`.
Be sure to reference S-Plus or R itself and other libraries used.

Acknowledgements

This work was supported by grants from the Agency for Health Care Policy and Research (US Public Health Service) and the Robert Wood Johnson Foundation.

Author(s)

Frank E Harrell Jr
Professor of Biostatistics
Chair, Department of Biostatistics
Vanderbilt University School of Medicine
Nashville, Tennessee
⟨f.harrell@vanderbilt.edu⟩

References

See Alzola CF, Harrell FE (2004): An Introduction to S and the Hmisc and Design Libraries at `http://biostat.mc.vanderbilt.edu/twiki/pub/Main/RS/sintro.pdf` for extensive documentation and examples for the Hmisc package.

`Save` *Faciliate Use of save and load to Remote Directories*

Description

These functions are slightly enhanced versions of `save` and `load` that allow a target directory to be specified using `options(LoadPath="pathname")`. If the `LoadPath` option is not set, the current working directory is used.

Usage

```
# options(LoadPath='mypath')
Save(object, name=deparse(substitute(object)))
Load(object)
```

Arguments

`object`	the name of an object, usually a data frame. It must not be quoted.
`name`	an optional name to assign to the object and file name prefix, if the argument name is not used

Details

`Save` creates a temporary version of the object under the name given by the user, so that `save` will internalize this name. Then subsequent `Load` or `load` will cause an object of the original name to be created in the global environment. The name of the R data file is assumed to be the name of the object (or the value of `name`) appended with `".rda"`. For `Save`, compression is used.

Author(s)

Frank Harrell

See Also

`save`, `load`

Examples

```
## Not run:
d <- data.frame(x=1:3, y=11:13)
options(LoadPath='../data/rda')
Save(d)   # creates ../data/rda/d.rda
Load(d)   # reads   ../data/rda/d.rda
Save(d, 'D')   # creates object D and saves it in .../D.rda
## End(Not run)
```

`abs.error.pred` *Indexes of Absolute Prediction Error for Linear Models*

Description

Computes the mean and median of various absolute errors related to ordinary multiple regression models. The mean and median absolute errors correspond to the mean square due to regression, error, and total. The absolute errors computed are derived from Yhat - median(Yhat), Yhat - Y, and Y - median(Y). The function also computes ratios that correspond to Rsquare and 1 - Rsquare (but these ratios do not add to 1.0); the Rsquare measure is the ratio of mean or median absolute Yhat - median(Yhat) to the mean or median absolute Y - median(Y). The 1 - Rsquare or SSE/SST measure is the mean or median absolute Yhat - Y divided by the mean or median absolute Y - median(Y).

Usage

```
abs.error.pred(fit, lp=NULL, y=NULL)

## S3 method for class 'abs.error.pred':
print(x, ...)
```

Arguments

`fit`	a fit object typically from `lm` or `ols` that contains a `y` vector (i.e., you should have specified `y=TRUE` to the fitting function) unless the `y` argument is given to `abs.error.pred`. If you do not specify the `lp` argument, `fit` must contain `fitted.values` or `linear.predictors`. You must specify `fit` or both of `lp` and `y`.
`lp`	a vector of predicted values (Y hat above) if `fit` is not given
`y`	a vector of response variable values if `fit` (with `y=TRUE` in effect) is not given
`x`	an object created by `abs.error.pred`
`...`	unused

Value

a list of class `abs.error.pred` (used by `print.abs.error.pred`) containing two matrices: `differences` and `ratios`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Schemper M (2003): Stat in Med 22:2299-2308.

See Also

lm, ols, cor, validate.ols

Examples

```
set.seed(1)          # so can regenerate results
x1 <- rnorm(100)
x2 <- rnorm(100)
y  <- exp(x1+x2+rnorm(100))
f <- lm(log(y) ~ x1 + poly(x2,3), y=TRUE)
abs.error.pred(lp=exp(fitted(f)), y=y)
rm(x1,x2,y,f)
```

`all.is.numeric` *Check if All Elements in Character Vector are Numeric*

Description

Tests, without issuing warnings, whether all elements of a character vector are legal numeric values, or optionally converts the vector to a numeric vector. Leading and trailing blanks in `x` are ignored.

Usage

```
all.is.numeric(x, what = c("test", "vector"), extras=c('.','NA'))
```

Arguments

`x`	a character vector
`what`	specify `what="vector"` to return a numeric vector if it passes the test, or the original character vector otherwise
`extras`	a vector of character strings to count as numeric values, other than `""`.

Value

a logical value if `what="test"` or a vector otherwise

Author(s)

Frank Harrell

See Also

as.numeric

Examples

```
all.is.numeric(c('1','1.2','3'))
all.is.numeric(c('1','1.2','3a'))
all.is.numeric(c('1','1.2','3'),'vector')
all.is.numeric(c('1','1.2','3a'),'vector')
all.is.numeric(c('1','',' .'),'vector')
```

`approxExtrap` *Linear Extrapolation*

Description

Works in conjunction with the `approx` function to do linear extrapolation. `approx` in R does not support extrapolation at all, and it is buggy in S-Plus 6.

Usage

```
approxExtrap(x, y, xout, method = "linear", n = 50, rule = 2, f = 0, ties = "ordere
```

Arguments

`x`	
`y`	
`xout`	
`method`	
`n`	
`rule`	
`f`	see `approx`
`ties`	applies only to R. See `approx`
`na.rm`	set to `TRUE` to remove `NA`s in `x` and `y` before proceeding

Details

Duplicates in `x` (and corresponding `y` elements) are removed before using `approx`.

Value

a vector the same length as `xout`

Author(s)

Frank Harrell

See Also

`approx`

Examples

```
approxExtrap(1:3,1:3,xout=c(0,4))
```

`areg` *Additive Regression with Optimal Transformations on Both Sides using Canonical Variates*

Description

Expands continuous variables into restricted cubic spline bases and categorical variables into dummy variables and fits a multivariate equation using canonical variates. This finds optimum transformations that maximize R^2. Optionally, the bootstrap is used to estimate the covariance matrix of both left- and right-hand-side transformation parameters, and to estimate the bias in the R^2 due to overfitting and compute the bootstrap optimism-corrected R^2. Cross-validation can also be used to get an unbiased estimate of R^2 but this is not as precise as the bootstrap estimate. The bootstrap and cross-validation may also used to get estimates of mean and median absolute error in predicted values on the original `y` scale. These two estimates are perhaps the best ones for gauging the accuracy of a flexible model, because it is difficult to compare R^2 under different y-transformations, and because R^2 allows for an out-of-sample recalibration (i.e., it only measures relative errors).

Note that uncertainty about the proper transformation of `y` causes an enormous amount of model uncertainty. When the transformation for `y` is estimated from the data a high variance in predicted values on the original `y` scale may result, especially if the true transformation is linear. Comparing bootstrap or cross-validated mean absolute errors with and without restricted the `y` transform to be linear (`ytype='l'`) may help the analyst choose the proper model complexity.

Usage

```
areg(x, y, xtype = NULL, ytype = NULL, nk = 4,
     B = 0, na.rm = TRUE, tolerance = NULL, crossval = NULL)

## S3 method for class 'areg':
print(x, digits=4, ...)

## S3 method for class 'areg':
plot(x, whichx = 1:ncol(x$x), ...)

## S3 method for class 'areg':
predict(object, x, type=c('lp','fitted'),
                        what=c('all','sample'), ...)
```

Arguments

`x`	A single predictor or a matrix of predictors. Categorical predictors are required to be coded as integers (as `factor` does internally). For `predict`, `x` is a data matrix with the same integer codes that were originally used for categorical variables.

`y`	a `factor`, categorical, character, or numeric response variable
`xtype`	a vector of one-letter character codes specifying how each predictor is to be modeled, in order of columns of `x`. The codes are `"s"` for smooth function (using restricted cubic splines), `"l"` for no transformation (linear), or `"c"` for categorical (to cause expansion into dummy variables). Default is `"s"` if `nk > 0` and `"l"` if `nk=0`.
`ytype`	same coding as for `xtype`. Default is `"s"` for a numeric variable with more than two unique values, `"l"` for a binary numeric variable, and `"c"` for a factor, categorical, or character variable.
`nk`	number of knots, 0 for linear, or 3 or more. Default is 4 which will fit 3 parameters to continuous variables (one linear term and two nonlinear terms)
`B`	number of bootstrap resamples used to estimate covariance matrices of transformation parameters. Default is no bootstrapping.
`na.rm`	set to `FALSE` if you are sure that observations with `NA`s have already been removed
`tolerance`	singularity tolerance. List source code for `lm.fit.qr.bare` for details.
`crossval`	set to a positive integer k to compute k-fold cross-validated R-squared (square of first canonical correlation) and mean and median absolute error of predictions on the original scale
`digits`	number of digits to use in formatting for printing
`object`	an object created by `areg`
`whichx`	integer or character vector specifying which predictors are to have their transformations plotted (default is all). The `y` transformation is always plotted.
`type`	tells `predict` whether to obtain predicted untransformed `y` (`type='lp'`, the default) or predicted `y` on the original scale (`type='fitted'`)
`what`	When the `y`-transform is non-monotonic you may specify `what='sample'` to `predict` to obtain a random sample of `y` values on the original scale instead of a matrix of all `y`-inverses. See `inverseFunction`.
`...`	arguments passed to the plot function.

Details

`areg` is a competitor of `ace` in the `acepack` package. Transformations from `ace` are seldom smooth enough and are often overfitted. With `areg` the complexity can be controlled with the `nk` parameter, and predicted values are easy to obtain because parametric functions are fitted.

If one side of the equation has a categorical variable with more than two categories and the other side has a continuous variable not assumed to act linearly, larger sample sizes are needed to reliably estimate transformations, as it is difficult to optimally score categorical variables to maximize R^2 against a simultaneously optimally transformed continuous variable.

Value

a list of class `"areg"` containing many objects

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Breiman and Friedman, Journal of the American Statistical Association (September, 1985).

See Also

`cancor`,`ace`, `transcan`

Examples

```
set.seed(1)

ns <- c(30,300,3000)
for(n in ns) {
  y <- sample(1:5, n, TRUE)
  x <- abs(y-3) + runif(n)
  par(mfrow=c(3,4))
  for(k in c(0,3:5)) {
    z <- areg(x, y, ytype='c', nk=k)
    plot(x, z$tx)
        title(paste('R2=',format(z$rsquared)))
    tapply(z$ty, y, range)
    a <- tapply(x,y,mean)
    b <- tapply(z$ty,y,mean)
    plot(a,b)
        abline(lsfit(a,b))
    # Should get same result to within linear transformation if reverse x and y
    w <- areg(y, x, xtype='c', nk=k)
    plot(z$ty, w$tx)
    title(paste('R2=',format(w$rsquared)))
    abline(lsfit(z$ty, w$tx))
 }
}

par(mfrow=c(2,2))
# Example where one category in y differs from others but only in variance of x
n <- 50
y <- sample(1:5,n,TRUE)
x <- rnorm(n)
x[y==1] <- rnorm(sum(y==1), 0, 5)
z <- areg(x,y,xtype='l',ytype='c')
z
plot(z)
z <- areg(x,y,ytype='c')
z
plot(z)
```

```
## Not run:
# Examine overfitting when true transformations are linear
par(mfrow=c(4,3))
for(n in c(200,2000)) {
  x <- rnorm(n); y <- rnorm(n) + x
    for(nk in c(0,3,5)) {
    z <- areg(x, y, nk=nk, crossval=10, B=100)
    print(z)
    plot(z)
    title(paste('n=',n))
  }
}
par(mfrow=c(1,1))

# Underfitting when true transformation is quadratic but overfitting
# when y is allowed to be transformed
set.seed(49)
n <- 200
x <- rnorm(n); y <- rnorm(n) + .5*x^2
#areg(x, y, nk=0, crossval=10, B=100)
#areg(x, y, nk=4, ytype='l', crossval=10, B=100)
z <- areg(x, y, nk=4) #, crossval=10, B=100)
z
# Plot x vs. predicted value on original scale.  Since y-transform is
# not monotonic, there are multiple y-inverses
xx <- seq(-3.5,3.5,length=1000)
yhat <- predict(z, xx, type='fitted')
plot(x, y, xlim=c(-3.5,3.5))
for(j in 1:ncol(yhat)) lines(xx, yhat[,j], col=j)
# Plot a random sample of possible y inverses
yhats <- predict(z, xx, type='fitted', what='sample')
points(xx, yhats, pch=2)
## End(Not run)

# True transformation of x1 is quadratic, y is linear
n <- 200
x1 <- rnorm(n); x2 <- rnorm(n); y <- rnorm(n) + x1^2
z <- areg(cbind(x1,x2),y,xtype=c('s','l'),nk=3)
par(mfrow=c(2,2))
plot(z)

# y transformation is inverse quadratic but areg gets the same answer by
# making x1 quadratic
n <- 5000
x1 <- rnorm(n); x2 <- rnorm(n); y <- (x1 + rnorm(n))^2
z <- areg(cbind(x1,x2),y,nk=5)
par(mfrow=c(2,2))
plot(z)

# Overfit 20 predictors when no true relationships exist
n <- 1000
x <- matrix(runif(n*20),n,20)
```

```
y <- rnorm(n)
z <- areg(x, y, nk=5)  # add crossval=4 to expose the problem

# Test predict function
n <- 50
x <- rnorm(n)
y <- rnorm(n) + x
g <- sample(1:3, n, TRUE)
z <- areg(cbind(x,g),y,xtype=c('s','c'))
range(predict(z, cbind(x,g)) - z$linear.predictors)
```

`aregImpute` *Multiple Imputation using Additive Regression, Bootstrapping, and Predictive Mean Matching*

Description

The `transcan` function creates flexible additive imputation models but provides only an approximation to true multiple imputation as the imputation models are fixed before all multiple imputations are drawn. This ignores variability caused by having to fit the imputation models. `aregImpute` takes all aspects of uncertainty in the imputations into account by using the bootstrap to approximate the process of drawing predicted values from a full Bayesian predictive distribution. Different bootstrap resamples are used for each of the multiple imputations, i.e., for the `i`th imputation of a sometimes missing variable, `i=1,2,... n.impute`, a flexible additive model is fitted on a sample with replacement from the original data and this model is used to predict all of the original missing and non-missing values for the target variable.

`areg` is used to fit the imputation models. By default, linearity is assumed for target variables (variables being imputed) and `nk=3` knots are assumed for continuous predictors transformed using restricted cubic splines. If `nk` is three or greater and `tlinear` is set to `FALSE`, `areg` simultaneously find transformations of the target variable and of all of the predictors, to get a good fit assuming additivity, maximizing R^2, using the same canonical correlation method as `transcan`. Flexible transformations may be overridden for specific variables by specifying the identity transformation for them. When a categorical variable is being predicted, the flexible transformation is Fisher's optimum scoring method. Nonlinear transformations for continuous variables may be nonmonotonic. If `nk` is a vector, `areg`'s bootstrap and `crossval=10` options will be used to help find the optimum validating value of `nk` over values of that vector, at the last imputation iteration. For the imputations, the minimum value of `nk` is used.

Instead of defaulting to taking random draws from fitted imputation models using random residuals as is done by `transcan`, `aregImpute` by default uses predictive mean matching with optional weighted probability sampling of donors rather than using only the closest match. Predictive mean matching works for binary, categorical, and continuous variables without the need for iterative maximum likelihood fitting for binary and categorical variables, and without the need for computing residuals or for curtailing imputed values to be in the range of actual data. Predictive mean matching is especially attractive when the variable being imputed is also being transformed automatically. See Details below for more information about the algorithm. A `"regression"` method is also available that is similar to that used in `transcan`. This option should be used when mechanistic missingness requires the use of extrapolation during imputation.

A `print` method summarizes the results, and a `plot` method plots distributions of imputed values. Typically, `fit.mult.impute` will be called after `aregImpute`.

If a target variable is transformed nonlinearly (i.e., if `nk` is greater than zero and `tlinear` is set to `FALSE`) and the estimated target variable transformation is non-monotonic, imputed values are not unique. When `type='regression'`, a random choice of possible inverse values is made.

Usage

```
aregImpute(formula, data, subset, n.impute=5, group=NULL,
           nk=3, tlinear=TRUE, type=c('pmm','regression'),
           match=c('weighted','closest'), fweighted=0.2,
           curtail=TRUE, boot.method=c('simple', 'approximate bayesian'),
           burnin=3, x=FALSE, pr=TRUE, plotTrans=FALSE, tolerance=NULL, B=75)
## S3 method for class 'aregImpute':
print(x, digits=3, ...)
## S3 method for class 'aregImpute':
plot(x, nclass=NULL, type=c('ecdf','hist'),
     datadensity=c("hist", "none", "rug", "density"),
     diagnostics=FALSE, maxn=10, ...)
```

Arguments

`formula`	an S model formula. You can specify restrictions for transformations of variables. The function automatically determines which variables are categorical (i.e., `factor`, `category`, or character vectors). Binary variables are automatically restricted to be linear. Force linear transformations of continuous variables by enclosing variables by the identify function (`I()`). It is recommended that `factor()` or `as.factor()` do not appear in the formula but instead variables be converted to factors as needed and stored in the data frame. That way imputations for factor variables (done using `impute.transcan` for example) will be correct.
`x`	an object created by `aregImpute`. For `aregImpute`, set `x` to `TRUE` to save the data matrix containing the final (number `n.impute`) imputations in the result. This is needed if you want to later do out-of-sample imputation. Categorical variables are coded as integers in this matrix.
`data`	
`subset`	These may be also be specified. You may not specify `na.action` as `na.retain` is always used.
`n.impute`	number of multiple imputations. `n.impute=5` is frequently recommended but 10 or more doesn't hurt.
`group`	a character or factor variable the same length as the number of observations in `data` and containing no `NA`s. When `group` is present, causes a bootstrap sample of the observations corresponding to non-`NA`s of a target variable to have the same frequency distribution of `group` as the that in the non-`NA`s of the original sample. This can handle k-sample problems as well as lower the chance that a bootstrap sample will have a missing cell when the original cell frequency was low.

`nk`	number of knots to use for continuous variables. When both the target variable and the predictors are having optimum transformations estimated, there is more instability than with normal regression so the complexity of the model should decrease more sharply as the sample size decreases. Hence set `nk` to 0 (to force linearity for non-categorical variables) or 3 (minimum number of knots possible with a linear tail-restricted cubic spline) for small sample sizes. Simulated problems as in the examples section can assist in choosing `nk`. See `nk` to a vector to get bootstrap-validated and 10-fold cross-validated R^2 and mean and median absolute prediction errors for imputing each sometimes-missing variable, with `nk` ranging over the given vector. The errors are on the original untransformed scale. The mean absolute error is the recommended basis for choosing the number of knots (or linearity).
`tlinear`	set to `FALSE` to allow a target variable (variable being imputed) to have a nonlinear left-hand-side transformation when `nk` is 3 or greater
`type`	The default is `"pmn"` for predictive mean matching, which is a more nonparametric approach that will work for categorical as well as continuous predictors. Alternatively, use `"regression"` when all variables that are sometimes missing are continuous and the missingness mechanism is such that entire intervals of population values are unobserved. See the Details section for more information. For the `plot` method, specify `type="hist"` to draw histograms of imputed values with rug plots at the top, or `type="ecdf"` (the default) to draw empirical CDFs with spike histograms at the bottom.
`match`	Defaults to `match="weighted"` to do weighted multinomial probability sampling using the tricube function (similar to lowess) as the weights. The argument of the tricube function is the absolute difference in transformed predicted values of all the donors and of the target predicted value, divided by a scaling factor. The scaling factor in the tricube function is `fweighted` times the mean absolute difference between the target predicted value and all the possible donor predicted values. Set `match="closest"` to find as the donor the observation having the closest predicted transformed value, even if that same donor is found repeatedly.
`fweighted`	Smoothing parameter (multiple of mean absolute difference) used when `match="weighted"`, with a default value of 0.2. Set `fweighted` to a number between 0.02 and 0.2 to force the donor to have a predicted value closer to the target, and set `fweighted` to larger values (but seldom larger than 1.0) to allow donor values to be less tightly matched. See the examples below to learn how to study the relationship between `fweighted` and the standard deviation of multiple imputations within individuals.
`curtail`	applies if `type='regression'`, causing imputed values to be curtailed at the observed range of the target variable. Set to `FALSE` to allow extrapolation outside the data range.
`boot.method`	By default, simple boostrapping is used in which the target variable is predicted using a sample with replacement from the observations with non-missing target variable. Specify `boot.method='approximate bayesian'` to build the imputation models from a sample with replacement from a sample with replacement of the observations with non-missing targets. Preliminary simulations

	have shown this results in good confidence coverage of the final model parameters when `type='regression'` is used. Not implemented when `group` is used.
`burnin`	`aregImpute` does `burnin + n.impute` iterations of the entire modeling process. The first `burnin` imputations are discarded. More burn-in iteractions may be requied when multiple variables are missing on the same observations.
`pr`	set to `FALSE` to suppress printing of iteration messages
`plotTrans`	set to `TRUE` to plot `ace` or `avas` transformations for each variable for each of the multiple imputations. This is useful for determining whether transformations are reasonable. If transformations are too noisy or have long flat sections (resulting in "lumps" in the distribution of imputed values), it may be advisable to place restrictions on the transformations (monotonicity or linearity).
`tolerance`	singularity criterion; list the source code in the `lm.fit.qr.bare` function for details
`B`	number of bootstrap resamples to use if `nk` is a vector
`digits`	number of digits for printing
`nclass`	number of bins to use in drawing histogram
`datadensity`	see `Ecdf`
`diagnostics`	Specify `diagnostics=TRUE` to draw plots of imputed values against sequential imputation numbers, separately for each missing observations and variable.
`maxn`	Maximum number of observations shown for diagnostics. Default is `maxn=10`, which limits the number of observations plotted to at most the first 10.
`...`	other arguments that are ignored

Details

The sequence of steps used by the `aregImpute` algorithm is the following.
(1) For each variable containing m `NA`s where m > 0, initialize the `NA`s to values from a random sample (without replacement if a sufficient number of non-missing values exist) of size m from the non-missing values.
(2) For `burnin+n.impute` iterations do the following steps. The first `burnin` iterations provide a burn-in, and imputations are saved only from the last `n.impute` iterations.
(3) For each variable containing any `NA`s, draw a sample with replacement from the observations in the entire dataset in which the current variable being imputed is non-missing. Fit a flexible additive model to predict this target variable while finding the optimum transformation of it (unless the identity transformation is forced). Use this fitted flexible model to predict the target variable in all of the original observations. Impute each missing value of the target variable with the observed value whose predicted transformed value is closest to the predicted transformed value of the missing value (if `match="closest"` and `type="pmm"`), or use a draw from a multinomial distribution with probabilities derived from distance weights, if `match="weighted"` (the default).
(4) After these imputations are computed, use these random draw imputations the next time the curent target variable is used as a predictor of other sometimes-missing variables.

When `match="closest"`, predictive mean matching does not work well when fewer than 3 variables are used to predict the target variable, because many of the multiple imputations for an observation will be identical. In the extreme case of one right-hand-side variable and assuming that only monotonic transformations of left and right-side variables are allowed, every bootstrap resample will give predicted values of the target variable that are monotonically related to predicted values from every other bootstrap resample. The same is true for Bayesian predicted values. This causes predictive mean matching to always match on the same donor observation.

When the missingness mechanism for a variable is so systematic that the distribution of observed values is truncated, predictive mean matching does not work. It will only yield imputed values that are near observed values, so intervals in which no values are observed will not be populated by imputed values. For this case, the only hope is to make regression assumptions and use extrapolation. With `type="regression"`, `aregImpute` will use linear extrapolation to obtain a (hopefully) reasonable distribution of imputed values. The `"regression"` option causes `aregImpute` to impute missing values by adding a random sample of residuals (with replacement if there are more `NA`s than measured values) on the transformed scale of the target variable. After random residuals are added, predicted random draws are obtained on the original untransformed scale using reverse linear interpolation on the table of original and transformed target values (linear extrapolation when a random residual is large enough to put the random draw prediction outside the range of observed values). The bootstrap is used as with `type="pmm"` to factor in the uncertainty of the imputation model.

As model uncertainty is high when the transformation of a target variable is unknown, `tlinear` defaults to `TRUE` to limit the variance in predicted values when `nk` is positive.

Value

a list of class `"aregImpute"` containing the following elements:

`call`	the function call expression
`formula`	the formula specified to `aregImpute`
`match`	the `match` argument
`fweighted`	the `fweighted` argument
`n`	total number of observations in input dataset
`p`	number of variables
`na`	list of subscripts of observations for which values were originally missing
`nna`	named vector containing the numbers of missing values in the data
`type`	vector of types of transformations used for each variable (`"s"`,`"l"`,`"c"` for smooth spline, linear, or categorical with dummy variables)
`tlinear`	value of `tlinear` parameter
`nk`	number of knots used for smooth transformations
`cat.levels`	list containing character vectors specifying the `levels` of categorical variables
`df`	degrees of freedom (number of parameters estimated) for each variable
`n.impute`	number of multiple imputations per missing value
`imputed`	a list containing matrices of imputed values in the same format as those created by `transcan`. Categorical variables are coded using their integer codes. Variables having no missing values will have `NULL` matrices in the list.
`x`	if `x` is `TRUE`, the original data matrix with integer codes for categorical variables
`rsq`	for the last round of imputations, a vector containing the R-squares with which each sometimes-missing variable could be predicted from the others by `ace` or `avas`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Little R, An H. Robust likelihood-based analysis of multivariate data with missing values. Statistica Sinica 14:933-952, 2004.

van Buuren S, Brand JPL, Groothuis-Oudshoorn CGM, Rubin DB. Fully conditional specifications in multivariate imputation. Draft available from `http://web.inter.nl.net/users/S.van.Buuren/publications/FCS%20(revised%20Jan%202005).pdf`.

See Also

`fit.mult.impute`, `transcan`, `areg`, `naclus`, `naplot`, `mice`, `dotchart2`, `Ecdf`

Examples

```
# Check that aregImpute can almost exactly estimate missing values when
# there is a perfect nonlinear relationship between two variables
# Fit restricted cubic splines with 4 knots for x1 and x2, linear for x3
set.seed(3)
x1 <- rnorm(200)
x2 <- x1^2
x3 <- runif(200)
m <- 30
x2[1:m] <- NA
a <- aregImpute(~x1+x2+I(x3), n.impute=5, nk=4, match='closest')
a
matplot(x1[1:m]^2, a$imputed$x2)
abline(a=0, b=1, lty=2)

x1[1:m]^2
a$imputed$x2

# Multiple imputation and estimation of variances and covariances of
# regression coefficient estimates accounting for imputation
# Example 1: large sample size, much missing data, no overlap in
# NAs across variables
x1 <- factor(sample(c('a','b','c'),1000,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(1000,0,2)
x3 <- rnorm(1000)
y  <- x2 + 1*(x1=='c') + .2*x3 + rnorm(1000,0,2)
orig.x1 <- x1[1:250]
orig.x2 <- x2[251:350]
x1[1:250] <- NA
x2[251:350] <- NA
d <- data.frame(x1,x2,x3,y)
# Find value of nk that yields best validating imputation models
# tlinear=FALSE means to not force the target variable to be linear
f <- aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), tlinear=FALSE,
                data=d, B=10) # normally B=75
f
# Try forcing target variable (x1, then x2) to be linear while allowing
# predictors to be nonlinear (could also say tlinear=TRUE)
f <- aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), data=d, B=10)
f

# Use 100 imputations to better check against individual true values
f <- aregImpute(~y + x1 + x2 + x3, n.impute=100, data=d)
f
par(mfrow=c(2,1))
plot(f)
modecat <- function(u) {
 tab <- table(u)
 as.numeric(names(tab)[tab==max(tab)][1])
}
table(orig.x1,apply(f$imputed$x1, 1, modecat))
par(mfrow=c(1,1))
plot(orig.x2, apply(f$imputed$x2, 1, mean))
fmi <- fit.mult.impute(y ~ x1 + x2 + x3, lm, f,
                       data=d)
sqrt(diag(Varcov(fmi)))
fcc <- lm(y ~ x1 + x2 + x3)
summary(fcc)   # SEs are larger than from mult. imputation

# Example 2: Very discriminating imputation models,
# x1 and x2 have some NAs on the same rows, smaller n
set.seed(5)
x1 <- factor(sample(c('a','b','c'),100,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(100,0,.4)
x3 <- rnorm(100)
y  <- x2 + 1*(x1=='c') + .2*x3 + rnorm(100,0,.4)
orig.x1 <- x1[1:20]
orig.x2 <- x2[18:23]
x1[1:20] <- NA
x2[18:23] <- NA
#x2[21:25] <- NA
d <- data.frame(x1,x2,x3,y)
n <- naclus(d)
plot(n); naplot(n)  # Show patterns of NAs
# 100 imputations to study them; normally use 5 or 10
f  <- aregImpute(~y + x1 + x2 + x3, n.impute=100, nk=0, data=d)
par(mfrow=c(2,3))
plot(f, diagnostics=TRUE, maxn=2)
# Note: diagnostics=TRUE makes graphs similar to those made by:
# r <- range(f$imputed$x2, orig.x2)
# for(i in 1:6) {  # use 1:2 to mimic maxn=2
#   plot(1:100, f$imputed$x2[i,], ylim=r,
#        ylab=paste("Imputations for Obs.",i))
#   abline(h=orig.x2[i],lty=2)
# }
```

```
table(orig.x1,apply(f$imputed$x1, 1, modecat))
par(mfrow=c(1,1))
plot(orig.x2, apply(f$imputed$x2, 1, mean))

fmi <- fit.mult.impute(y ~ x1 + x2, lm, f,
                       data=d)
sqrt(diag(Varcov(fmi)))
fcc <- lm(y ~ x1 + x2)
summary(fcc)   # SEs are larger than from mult. imputation

# Study relationship between smoothing parameter for weighting function
# (multiplier of mean absolute distance of transformed predicted
# values, used in tricube weighting function) and standard deviation
# of multiple imputations.  SDs are computed from average variances
# across subjects.  match="closest" same as match="weighted" with
# small value of fweighted.
# This example also shows problems with predicted mean
# matching almost always giving the same imputed values when there is
# only one predictor (regression coefficients change over multiple
# imputations but predicted values are virtually 1-1 functions of each
# other)

set.seed(23)
x <- runif(200)
y <- x + runif(200, -.05, .05)
r <- resid(lsfit(x,y))
rmse <- sqrt(sum(r^2)/(200-2))   # sqrt of residual MSE

y[1:20] <- NA
d <- data.frame(x,y)
f <- aregImpute(~ x + y, n.impute=10, match='closest', data=d)
# As an aside here is how to create a completed dataset for imputation
# number 3 as fit.mult.impute would do automatically.  In this degenerate
# case changing 3 to 1-2,4-10 will not alter the results.
completed <- d
imputed <- impute.transcan(f, imputation=3, data=d, list.out=TRUE,
                           pr=FALSE, check=FALSE)
completed[names(imputed)] <- imputed
completed
sd <- sqrt(mean(apply(f$imputed$y, 1, var)))

ss <- c(0, .01, .02, seq(.05, 1, length=20))
sds <- ss; sds[1] <- sd

for(i in 2:length(ss)) {
  f <- aregImpute(~ x + y, n.impute=10, fweighted=ss[i])
  sds[i] <- sqrt(mean(apply(f$imputed$y, 1, var)))
}

plot(ss, sds, xlab='Smoothing Parameter', ylab='SD of Imputed Values',
     type='b')
abline(v=.2,  lty=2)  # default value of fweighted
abline(h=rmse, lty=2)  # root MSE of residuals from linear regression

## Not run:
# Do a similar experiment for the Titanic dataset
getHdata(titanic3)
h <- lm(age ~ sex + pclass + survived, data=titanic3)
rmse <- summary(h)$sigma
set.seed(21)
f <- aregImpute(~ age + sex + pclass + survived, n.impute=10,
                data=titanic3, match='closest')
sd <- sqrt(mean(apply(f$imputed$age, 1, var)))

ss <- c(0, .01, .02, seq(.05, 1, length=20))
sds <- ss; sds[1] <- sd

for(i in 2:length(ss)) {
  f <- aregImpute(~ age + sex + pclass + survived, data=titanic3,
                  n.impute=10, fweighted=ss[i])
  sds[i] <- sqrt(mean(apply(f$imputed$age, 1, var)))
}

plot(ss, sds, xlab='Smoothing Parameter', ylab='SD of Imputed Values',
     type='b')
abline(v=.2,   lty=2)  # default value of fweighted
abline(h=rmse, lty=2)  # root MSE of residuals from linear regression
## End(Not run)
```

`biVar` *Bivariate Summaries Computed Separately by a Series of Predictors*

Description

`biVar` is a generic function that accepts a formula and usual `data`, `subset`, and `na.action` parameters plus a list `statinfo` that specifies a function of two variables to compute along with information about labeling results for printing and plotting. The function is called separately with each right hand side variable and the same left hand variable. The result is a matrix of bivariate statistics and the `statinfo` list that drives printing and plotting. The plot method draws a dot plot with x-axis values by default sorted in order of one of the statistics computed by the function.

`spearman2` computes the square of Spearman's rho rank correlation and a generalization of it in which `x` can relate non-monotonically to `y`. This is done by computing the Spearman multiple rho-squared between `(rank(x), rank(x)^2)` and `y`. When `x` is categorical, a different kind of Spearman correlation used in the Kruskal-Wallis test is computed (and `spearman2` can do the Kruskal-Wallis test). This is done by computing the ordinary multiple `R^2` between `k-1` dummy variables and `rank(y)`, where `x` has `k` categories. `x` can also be a formula, in which case each predictor is correlated separately with `y`, using non-missing observations for that predictor. `biVar` is used to do the looping and bookkeeping. By default the plot shows the adjusted `rho^2`, using the same formula used for the ordinary adjusted `R^2`. The `F` test uses the unadjusted R2.

`spearman` computes Spearman's rho on non-missing values of two variables. `spearman.test` is a simple version of `spearman2.default`.

chiSquare is set up like spearman2 except it is intended for a categorical response variable. Separate Pearson chi-square tests are done for each predictor, with optional collapsing of infrequent categories. Numeric predictors having more than g levels are categorized into g quantile groups. chiSquare uses biVar.

Usage

```
biVar(formula, statinfo, data=NULL, subset=NULL,
      na.action=na.retain, exclude.imputed=TRUE, ...)

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

## S3 method for class 'biVar':
plot(x, what=info$defaultwhat,
                       sort.=TRUE,
                       main, xlab, ...)

spearman2(x, ...)

## Default S3 method:
spearman2(x, y, p=1, minlev=0, na.rm=TRUE, exclude.imputed=na.rm, ...)

## S3 method for class 'formula':
spearman2(formula, data=NULL,
          subset, na.action=na.retain, exclude.imputed=TRUE, ...)

spearman(x, y)

spearman.test(x, y, p=1)

chiSquare(formula, data=NULL, subset=NULL, na.action=na.retain,
          exclude.imputed=TRUE, ...)
```

Arguments

`formula`	a formula with a single left side variable
`statinfo`	see `spearman2.formula` or `chiSquare` code
`data, subset, na.action`	the usual options for models. Default for `na.action` is to retain all values, NA or not, so that NAs can be deleted in only a pairwise fashion.
`exclude.imputed`	set to `FALSE` to include imputed values (created by `impute`) in the calculations.
`...`	other arguments that are passed to the function used to compute the bivariate statistics or to `dotchart2` for `plot`.
`na.rm`	logical; delete NA values?

`x`	a numeric matrix with at least 5 rows and at least 2 columns (if `y` is absent). For `spearman2`, the first argument may be a vector of any type, including character or factor. The first argument may also be a formula, in which case all predictors are correlated individually with the response variable. `x` may be a formula for `spearman2` in which case `spearman2.formula` is invoked. Each predictor in the right hand side of the formula is separately correlated with the response variable. For `print` or `plot`, `x` is an object produced by `biVar`. For `spearman` and `spearman.test` `x` is a numeric vector, as is `y`. For `chiSquare`, `x` is a formula.
`y`	a numeric vector
`p`	for numeric variables, specifies the order of the Spearman `rho^2` to use. The default is `p=1` to compute the ordinary `rho^2`. Use `p=2` to compute the quadratic rank generalization to allow non-monotonicity. `p` is ignored for categorical predictors.
`minlev`	minimum relative frequency that a level of a categorical predictor should have before it is pooled with other categories (see `combine.levels`) in `spearman2` and `chiSquare` (in which case it also applies to the response). The default, `minlev=0` causes no pooling.
`what`	specifies which statistic to plot. Possibilities include the column names that appear with the print method is used.
`sort.`	set `sort.=FALSE` to suppress sorting variables by the statistic being plotted
`main`	main title for plot. Default title shows the name of the response variable.
`xlab`	x-axis label. Default constructed from `what`.

Details

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values for Spearman, Wilcoxon, or Kruskal-Wallis tests are approximated by using the `t` or `F` distributions.

Value

`spearman2.default` (the function that is called for a single `x`, i.e., when there is no formula) returns a vector of statistics for the variable. `biVar`, `spearman2.formula`, and `chiSquare` return a matrix with rows corresponding to predictors.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Hollander M. and Wolfe D.A. (1973). Nonparametric Statistical Methods. New York: Wiley.

Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): Numerical Recipes in C. Cambridge: Cambridge University Press.

See Also

combine.levels, varclus, dotchart2, impute, chisq.test, cut2.

Examples

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4,   1, 0, 1, 4)
z <- c(1,   2, 3, 4, NA)
v <- c(1,   2, 3, 4, 5)

spearman2(x, y)
plot(spearman2(z ~ x + y + v, p=2))

f <- chiSquare(z ~ x + y + v)
f
```

---

binconf *Confidence Intervals for Binomial Probabilities*

Description

Produces 1-alpha confidence intervals for binomial probabilities.

Usage

```
binconf(x, n, alpha=0.05,
        method=c("wilson","exact","asymptotic","all"),
        include.x=FALSE, include.n=FALSE, return.df=FALSE)
```

Arguments

`x`	vector containing the number of "successes" for binomial variates
`n`	vector containing the numbers of corresponding observations
`alpha`	probability of a type I error, so confidence coefficient = 1-alpha
`method`	character string specifing which method to use. The "all" method only works when x and n are length 1. The "exact" method uses the F distribution to compute exact (based on the binomial cdf) intervals; the "wilson" interval is score-test-based; and the "asymptotic" is the text-book, asymptotic normal interval. Following Agresti and Coull, the Wilson interval is to be preferred and so is the default.
`include.x`	logical flag to indicate whether `x` should be included in the returned matrix or data frame
`include.n`	logical flag to indicate whether `n` should be included in the returned matrix or data frame
`return.df`	logical flag to indicate that a data frame rather than a matrix be returned

Value

a matrix or data.frame containing the computed intervals and, optionally, `x` and `n`.

Author(s)

Rollin Brant, Modified by Frank Harrell and
Brad Biggerstaff
Centers for Disease Control and Prevention
National Center for Infectious Diseases
Division of Vector-Borne Infectious Diseases
P.O. Box 2087, Fort Collins, CO, 80522-2087, USA
bkb5@cdc.gov

References

A. Agresti and B.A. Coull, Approximate is better than "exact" for interval estimation of binomial proportions, *American Statistician,* **52**:119–126, 1998.

R.G. Newcombe, Logit confidence intervals and the inverse sinh transformation, *American Statistician,* **55**:200–202, 2001.

L.D. Brown, T.T. Cai and A. DasGupta, Interval estimation for a binomial proportion (with discussion), *Statistical Science,* **16**:101–133, 2001.

Examples

```
binconf(0:10,10,include.x=TRUE,include.n=TRUE)
binconf(46,50,method="all")
```

`bootkm` *Bootstrap Kaplan-Meier Estimates*

Description

Bootstraps Kaplan-Meier estimate of the probability of survival to at least a fixed time (`times` variable) or the estimate of the `q` quantile of the survival distribution (e.g., median survival time, the default).

Usage

```
bootkm(S, q=0.5, B=500, times, pr=TRUE)
```

Arguments

`S`	a `Surv` object for possibly right-censored survival time
`q`	quantile of survival time, default is 0.5 for median
`B`	number of bootstrap repetitions (default=500)

`times`	time vector (currently only a scalar is allowed) at which to compute survival estimates. You may specify only one of `q` and `times`, and if `times` is specified `q` is ignored.
`pr`	set to `FALSE` to suppress printing the iteration number every 10 iterations

Details

`bootkm` uses Therneau's `survfit.km` function to efficiently compute Kaplan-Meier estimates.

Value

a vector containing `B` bootstrap estimates

Side Effects

updates `.Random.seed`, and, if `pr=TRUE`, prints progress of simulations

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Akritas MG (1986): Bootstrapping the Kaplan-Meier estimator. JASA 81:1032–1038.

See Also

`survfit`, `survfit.km`, `Surv`, `Survival.cph`, `Quantile.cph`

Examples

```
# Compute 0.95 nonparametric confidence interval for the difference in
# median survival time between females and males (two-sample problem)
set.seed(1)
library(survival)
S <- Surv(runif(200))      # no censoring
sex <- c(rep('female',100),rep('male',100))
med.female <- bootkm(S[sex=='female',], B=100) # normally B=500
med.male   <- bootkm(S[sex=='male',],   B=100)
describe(med.female-med.male)
quantile(med.female-med.male, c(.025,.975), na.rm=TRUE)
# na.rm needed because some bootstrap estimates of median survival
# time may be missing when a bootstrap sample did not include the
# longer survival times
```

`bpower` *Power and Sample Size for Two-Sample Binomial Test*

Description

Uses method of Fleiss, Tytun, and Ury (but without the continuity correction) to estimate the power (or the sample size to achieve a given power) of a two-sided test for the difference in two proportions. The two sample sizes are allowed to be unequal, but for `bsamsize` you must specify the fraction of observations in group 1. For power calculations, one probability (`p1`) must be given, and either the other probability (`p2`), an `odds.ratio`, or a `percent.reduction` must be given. For `bpower` or `bsamsize`, any or all of the arguments may be vectors, in which case they return a vector of powers or sample sizes. All vector arguments must have the same length.

Given `p1, p2`, `ballocation` uses the method of Brittain and Schlesselman to compute the optimal fraction of observations to be placed in group 1 that either (1) minimize the variance of the difference in two proportions, (2) minimize the variance of the ratio of the two proportions, (3) minimize the variance of the log odds ratio, or (4) maximize the power of the 2-tailed test for differences. For (4) the total sample size must be given, or the fraction optimizing the power is not returned. The fraction for (3) is one minus the fraction for (1).

`bpower.sim` estimates power by simulations, in minimal time. By using `bpower.sim` you can see that the formulas without any continuity correction are quite accurate, and that the power of a continuity-corrected test is significantly lower. That's why no continuity corrections are implemented here.

Usage

```
bpower(p1, p2, odds.ratio, percent.reduction,
       n, n1, n2, alpha=0.05)

bsamsize(p1, p2, fraction=.5, alpha=.05, power=.8)

ballocation(p1, p2, n, alpha=.05)

bpower.sim(p1, p2, odds.ratio, percent.reduction,
           n, n1, n2,
           alpha=0.05, nsim=10000)
```

Arguments

`p1`	population probability in the group 1
`p2`	probability for group 2
`odds.ratio`	
`percent.reduction`	
`n`	total sample size over the two groups. If you omit this for `ballocation`, the `fraction` which optimizes power will not be returned.

`n1`	
`n2`	the individual group sample sizes. For `bpower`, if `n` is given, `n1` and `n2` are set to `n/2`.
`alpha`	type I error
`fraction`	fraction of observations in group 1
`power`	the desired probability of detecting a difference
`nsim`	number of simulations of binomial responses

Details

For `bpower.sim`, all arguments must be of length one.

Value

for `bpower`, the power estimate; for `bsamsize`, a vector containing the sample sizes in the two groups; for `ballocation`, a vector with 4 fractions of observations allocated to group 1, optimizing the four criteria mentioned above. For `bpower.sim`, a vector with three elements is returned, corresponding to the simulated power and its lower and upper 0.95 confidence limits.

AUTHOR

Frank Harrell

Department of Biostatistics

Vanderbilt University

f.harrell@vanderbilt.edu

References

Fleiss JL, Tytun A, Ury HK (1980): A simple approximation for calculating sample sizes for comparing independent proportions. Biometrics 36:343–6.

Brittain E, Schlesselman JJ (1982): Optimal allocation for the comparison of proportions. Biometrics 38:1003–9.

Gordon I, Watson R (1996): The myth of continuity-corrected sample size formulae. Biometrics 52:71–6.

See Also

`samplesize.bin`, `chisq.test`, `binconf`

Examples

```
bpower(.1, odds.ratio=.9, n=1000, alpha=c(.01,.05))
bpower.sim(.1, odds.ratio=.9, n=1000)
bsamsize(.1, .05, power=.95)
ballocation(.1, .5, n=100)

# Plot power vs. n for various odds ratios  (base prob.=.1)
```

```
n  <- seq(10, 1000, by=10)
OR <- seq(.2,.9,by=.1)
plot(0, 0, xlim=range(n), ylim=c(0,1), xlab="n", ylab="Power", type="n")
for(or in OR) {
  lines(n, bpower(.1, odds.ratio=or, n=n))
  text(350, bpower(.1, odds.ratio=or, n=350)-.02, format(or))
}

# Another way to plot the same curves, but letting labcurve do the
# work, including labeling each curve at points of maximum separation
pow <- lapply(OR, function(or,n)list(x=n,y=bpower(p1=.1,odds.ratio=or,n=n)),
              n=n)
names(pow) <- format(OR)
labcurve(pow, pl=TRUE, xlab='n', ylab='Power')

# Contour graph for various probabilities of outcome in the control
# group, fixing the odds ratio at .8 ([p2/(1-p2) / p1/(1-p1)] = .8)
# n is varied also
p1 <- seq(.01,.99,by=.01)
n  <- seq(100,5000,by=250)
pow <- outer(p1, n, function(p1,n) bpower(p1, n=n, odds.ratio=.8))
# This forms a length(p1)*length(n) matrix of power estimates
contour(p1, n, pow)
```

---

bpplot *Box-percentile plots*

Description

Producess side-by-side box-percentile plots from several vectors or a list of vectors.

Usage

```
bpplot(..., name=TRUE, main="Box-Percentile Plot",
       xlab="", ylab="", srtx=0)
```

Arguments

`...`	vectors or lists containing numeric components (e.g., the output of `split`).
`name`	character vector of names for the groups. Default is `TRUE` to put names on the x-axis. Such names are taken from the data vectors or the `names` attribute of the first argument if it is a list. Set `name` to `FALSE` to suppress names. If a character vector is supplied the names in the vector are used to label the groups.
`main`	main title for the plot.
`xlab`	x axis label.
`ylab`	y axis label.
`srtx`	rotation angle for x-axis labels. Default is zero.

Value

There are no returned values

Side Effects

A plot is created on the current graphics device.

BACKGROUND

Box-percentile plots are similiar to boxplots, except box-percentile plots supply more information about the univariate distributions. At any height the width of the irregular "box" is proportional to the percentile of that height, up to the 50th percentile, and above the 50th percentile the width is proportional to 100 minus the percentile. Thus, the width at any given height is proportional to the percent of observations that are more extreme in that direction. As in boxplots, the median, 25th and 75th percentiles are marked with line segments across the box.

Author(s)

Jeffrey Banfield
umsfjban@bill.oscs.montana.edu
Modified by F. Harrell 30Jun97

References

Esty, W. W. and Banfield, J. D. (1992) "The Box-Percentile Plot," Technical Report (May 15, 1992), Department of Mathematical Sciences, Montana State University.

See Also

`panel.bpplot`, `boxplot`, `Ecdf`, `bwplot`

Examples

```
set.seed(1)
x1 <- rnorm(500)
x2 <- runif(500, -2, 2)
x3 <- abs(rnorm(500))-2
bpplot(x1, x2, x3)
g <- sample(1:2, 500, replace=TRUE)
bpplot(split(x2, g), name=c('Group 1','Group 2'))
rm(x1,x2,x3,g)
```

bystats *Statistics by Categories*

Description

For any number of cross-classification variables, `bystats` returns a matrix with the sample size, number missing `y`, and `fun(non-missing y)`, with the cross-classifications designated by rows. Uses Harrell's modification of the `interaction` function to produce cross-classifications. The default `fun` is `mean`, and if `y` is binary, the mean is labeled as `Fraction`. There is a `print` method as well as a `latex` method for objects created by `bystats`. `bystats2` handles the special case in which there are 2 classifcation variables, and places the first one in rows and the second in columns. The `print` method for `bystats2` uses the S-Plus `print.char.matrix` function to organize statistics for cells into boxes.

Usage

```
bystats(y, ..., fun, nmiss, subset)
## S3 method for class 'bystats':
print(x, ...)
## S3 method for class 'bystats':
latex(object, title, caption, rowlabel, ...)
bystats2(y, v, h, fun, nmiss, subset)
## S3 method for class 'bystats2':
print(x, abbreviate.dimnames=FALSE,
   prefix.width=max(nchar(dimnames(x)[[1]])), ...)
## S3 method for class 'bystats2':
latex(object, title, caption, rowlabel, ...)
```

Arguments

`y`	a binary, logical, or continuous variable or a matrix or data frame of such variables. If `y` is a data frame it is converted to a matrix. If `y` is a data frame or matrix, computations are done on subsets of the rows of `y`, and you should specify `fun` so as to be able to operate on the matrix. For matrix `y`, any column with a missing value causes the entire row to be considered missing, and the row is not passed to `fun`.
`...`	For `bystats`, one or more classifcation variables separated by commas. For `print.bystats`, options passed to `print.default` such as `digits`. For `latex.bystats`, and `latex.bystats2`, options passed to `latex.default` such as `digits`. If you pass `cdec` to `latex.default`, keep in mind that the first one or two positions (depending on `nmiss`) should have zeros since these correspond with frequency counts.
`v`	vertical variable for `bystats2`. Will be converted to `factor`.
`h`	horizontal variable for `bystats2`. Will be converted to `factor`.

`fun`	a function to compute on the non-missing `y` for a given subset. You must specify `fun=` in front of the function name or definition. `fun` may return a single number or a vector or matrix of any length. Matrix results are rolled out into a vector, with names preserved. When `y` is a matrix, a common `fun` is `function(y) apply(y, 2, ff)` where `ff` is the name of a function which operates on one column of `y`.
`nmiss`	A column containing a count of missing values is included if `nmiss=TRUE` or if there is at least one missing value.
`subset`	a vector of subscripts or logical values indicating the subset of data to analyze
`abbreviate.dimnames`	set to `TRUE` to abbreviate `dimnames` in output
`prefix.width`	see `print.char.matrix` if using S-Plus
`title`	`title` to pass to `latex.default`. Default is the first word of the character string version of the first calling argument.
`caption`	caption to pass to `latex.default`. Default is the `heading` attribute from the object produced by `bystats`.
`rowlabel`	`rowlabel` to pass to `latex.default`. Default is the `byvarnames` attribute from the object produced by `bystats`. For `bystats2` the default is `""`.
`x`	an object created by `bystats` or `bystats2`
`object`	an object created by `bystats` or `bystats2`

Value

for `bystats`, a matrix with row names equal to the classification labels and column names `N, Missing, funlab`, where `funlab` is determined from `fun`. A row is added to the end with the summary statistics computed on all observations combined. The class of this matrix is `bystats`. For `bystats`, returns a 3-dimensional array with the last dimension corresponding to statistics being computed. The class of the array is `bystats2`.

Side Effects

`latex` produces a `.tex` file.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`interaction`, `cut`, `cut2`, `latex`, `print.char.matrix`, `translate`

Examples

```
## Not run:
bystats(sex==2, county, city)
bystats(death, race)
bystats(death, cut2(age,g=5), race)
bystats(cholesterol, cut2(age,g=4), sex, fun=median)
bystats(cholesterol, sex, fun=quantile)
bystats(cholesterol, sex, fun=function(x)c(Mean=mean(x),Median=median(x)))
latex(bystats(death,race,nmiss=FALSE,subset=sex=="female"), digits=2)
f <- function(y) c(Hazard=sum(y[,2])/sum(y[,1]))
# f() gets the hazard estimate for right-censored data from exponential dist.
bystats(cbind(d.time, death), race, sex, fun=f)
bystats(cbind(pressure, cholesterol), age.decile,
        fun=function(y) c(Median.pressure   =median(y[,1]),
                          Median.cholesterol=median(y[,2])))
y <- cbind(pressure, cholesterol)
bystats(y, age.decile,
        fun=function(y) apply(y, 2, median))   # same result as last one
bystats(y, age.decile, fun=function(y) apply(y, 2, quantile, c(.25,.75)))
# The last one computes separately the 0.25 and 0.75 quantiles of 2 vars.
latex(bystats2(death, race, sex, fun=table))
## End(Not run)
```

`ciapower` *Power of Interaction Test for Exponential Survival*

Description

Uses the method of Peterson and George to compute the power of an interaction test in a 2 x 2 setup in which all 4 distributions are exponential. This will be the same as the power of the Cox model test if assumptions hold. The test is 2-tailed. The duration of accrual is specified (constant accrual is assumed), as is the minimum follow-up time. The maximum follow-up time is then `accrual + tmin`. Treatment allocation is assumed to be 1:1.

Usage

```
ciapower(tref, n1, n2, m1c, m2c, r1, r2, accrual, tmin,
         alpha=0.05, pr=TRUE)
```

Arguments

`tref`	time at which mortalities estimated
`n1`	total sample size, stratum 1
`n2`	total sample size, stratum 2
`m1c`	tref-year mortality, stratum 1 control
`m2c`	tref-year mortality, stratum 2 control
`r1`	% reduction in `m1c` by intervention, stratum 1

`r2`	% reduction in `m2c` by intervention, stratum 2
`accrual`	duration of accrual period
`tmin`	minimum follow-up time
`alpha`	type I error probability
`pr`	set to `FALSE` to suppress printing of details

Value

power

Side Effects

prints

AUTHOR

Frank Harrell

Department of Biostatistics

Vanderbilt University

f.harrell@vanderbilt.edu

References

Peterson B, George SL: Controlled Clinical Trials 14:511–522; 1993.

See Also

`cpower`, `spower`

Examples

```
# Find the power of a race x treatment test.  25% of patients will
# be non-white and the total sample size is 14000.
# Accrual is for 1.5 years and minimum follow-up is 5y.
# Reduction in 5-year mortality is 15% for whites, 0% or -5% for
# non-whites.  5-year mortality for control subjects if assumed to
# be 0.18 for whites, 0.23 for non-whites.
n <- 14000
for(nonwhite.reduction in c(0,-5)) {
  cat("\n\n\n% Reduction in 5-year mortality for non-whites:",
      nonwhite.reduction, "\n\n")
  pow <- ciapower(5,  .75*n, .25*n,  .18, .23,  15, nonwhite.reduction,
                  1.5, 5)
  cat("\n\nPower:",format(pow),"\n")
}
```

combine *Element Merging*

Description

Merges an object by the names of its elements. Inserting elements in `value` into `x` that do not exists in `x` and replacing elements in `x` that exists in `value` with `value` elements if `protect` is false.

Usage

```
combine(x, value, protect = FALSE, ...)

combine(x, protect = FALSE, ...) <- value
```

Arguments

`x`	named list or vector
`value`	named list or vector
`protect`	logical; should elements in `x` be kept instead of elements in `value`?
`...`	currently does nothing; included if ever want to make generic.

Author(s)

Charles Dupont

See Also

`names`

Examples

```
x <- 1:5
names(x) <- LETTERS[x]

y <- 6:10
names(y) <- LETTERS[y-2]

x                  # c(A=1,B=2,C=3,D=4,E=5)
y                  # c(D=6,E=7,F=8,G=9,H=10)

combine(x, y)      # c(A=1,B=2,C=3,D=6,E=7,F=8,G=9,H=10)
combine(x, y, protect=TRUE)      # c(A=1,B=2,C=3,D=4,E=5,F=8,G=9,H=10)
```

contents *Metadata for a Data Frame*

Description

`contents` is a generic method for which `contents.data.frame` is currently the only method. `contents.data.frame` creates an object containing the following attributes of the variables from a data frame: names, labels (if any), units (if any), number of factor levels (if any), factor levels, class, storage mode, and number of NAs. `print.contents.data.frame` will print the results, with options for sorting the variables. `html.contents.data.frame` creates HTML code for displaying the results. This code has hyperlinks so that if the user clicks on the number of levels the browser jumps to the correct part of a table of factor levels for all the `factor` variables. If long labels are present (`"longlabel"` attributes on variables), these are printed at the bottom and the `html` method links to them through the regular labels. Variables having the same `levels` in the same order have the levels factored out for brevity.

`contents.list` prints a directory of datasets when `sasxport.get` imported more than one SAS dataset.

Usage

```
contents(object, ...)
## S3 method for class 'data.frame':
contents(object, ...)
## S3 method for class 'contents.data.frame':
print(x,
    sort=c('none','names','labels','NAs'), prlevels=TRUE, ...)
## S3 method for class 'contents.data.frame':
html(object,
           sort=c('none','names','labels','NAs'), prlevels=TRUE,
           file=paste('contents',object$dfname,'html',sep='.'),
           levelType=c('list','table'),
           append=FALSE, ...)
## S3 method for class 'list':
contents(object, dslabels, ...)
## S3 method for class 'contents.list':
print(x,
    sort=c('none','names','labels','NAs','vars'), ...)
```

Arguments

`object`	a data frame. For `html` is an object created by `contents`. For `contents.list` is a list of data frames.
`x`	an object created by `contents`
`sort`	Default is to print the variables in their original order in the data frame. Specify one of `"names"`, `"labels"`, or `"NAs"` to sort the variables by, respectively, alphabetically by names, alphabetically by labels, or by increaseing order of

	number of missing values. For `contents.list`, `sort` may also be the value `"vars"` to cause sorting by the number of variables in the dataset.
`prlevels`	set to `FALSE` to not print all levels of `factor` variables
`file`	file to which to write the html code. Default is `"conents.dfname.html"` where `dfname` is the name of the data frame processed by `contents`.
`levelType`	By default, bullet lists of category levels are constructed in html. Set `levelType='table'` to put levels in html table format.
`append`	set to `TRUE` to add html code to an existing file
`...`	arguments passed from `html` to `format.df`, unused otherwise
`dslabels`	named vector of SAS dataset labels, created for example by `sasdsLabels`

Value

an object of class `"contents.data.frame"` or `"contents.list"`

Author(s)

Frank Harrell
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`describe`, `html`, `upData`

Examples

```
set.seed(1)
dfr <- data.frame(x=rnorm(400),y=sample(c('male','female'),400,TRUE))
contents(dfr)
dfr <- upData(dfr, labels=c(x='Label for x', y='Label for y'))
attr(dfr$x, 'longlabel') <-
 'A very long label for x that can continue onto multiple long lines of text'

k <- contents(dfr)
print(k, sort='names', prlevels=FALSE)
## Not run:
html(k)
html(contents(dfr))              # same result
w <- html(k, file='my.html')    # create my.html, don't display
## End(Not run)
```

`cpower` *Power of Cox/log-rank Two-Sample Test*

Description

Assumes exponential distributions for both treatment groups. Uses the George-Desu method along with formulas of Schoenfeld that allow estimation of the expected number of events in the two groups. To allow for drop-ins (noncompliance to control therapy, crossover to intervention) and noncompliance of the intervention, the method of Lachin and Foulkes is used.

Usage

```
cpower(tref, n, mc, r, accrual, tmin, noncomp.c=0, noncomp.i=0,
       alpha=0.05, nc, ni, pr=TRUE)
```

Arguments

`tref`	time at which mortalities estimated
`n`	total sample size (both groups combined). If allocation is unequal so that there are not `n/2` observations in each group, you may specify the sample sizes in `nc` and `ni`.
`mc`	tref-year mortality, control
`r`	% reduction in `mc` by intervention
`accrual`	duration of accrual period
`tmin`	minimum follow-up time
`noncomp.c`	% non-compliant in control group (drop-ins)
`noncomp.i`	% non-compliant in intervention group (non-adherers)
`alpha`	type I error probability. A 2-tailed test is assumed.
`nc`	number of subjects in control group
`ni`	number of subjects in intervention group. `nc` and `ni` are specified exclusive of `n`.
`pr`	set to `FALSE` to suppress printing of details

Details

For handling noncompliance, uses a modification of formula (5.4) of Lachin and Foulkes. Their method is based on a test for the difference in two hazard rates, whereas `cpower` is based on testing the difference in two log hazards. It is assumed here that the same correction factor can be approximately applied to the log hazard ratio as Lachin and Foulkes applied to the hazard difference.

Note that Schoenfeld approximates the variance of the log hazard ratio by `4/m`, where `m` is the total number of events, whereas the George-Desu method uses the slightly better `1/m1 + 1/m2`. Power from this function will thus differ slightly from that obtained with the SAS `samsizc` program.

Value

power

Side Effects

prints

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Peterson B, George SL: Controlled Clinical Trials 14:511–522; 1993.

Lachin JM, Foulkes MA: Biometrics 42:507–519; 1986.

Schoenfeld D: Biometrics 39:499–503; 1983.

See Also

spower, ciapower, bpower

Examples

```
#In this example, 4 plots are drawn on one page, one plot for each
#combination of noncompliance percentage.  Within a plot, the
#5-year mortality % in the control group is on the x-axis, and
#separate curves are drawn for several % reductions in mortality
#with the intervention.  The accrual period is 1.5y, with all
#patients followed at least 5y and some 6.5y.

par(mfrow=c(2,2),oma=c(3,0,3,0))

morts <- seq(10,25,length=50)
red <- c(10,15,20,25)

for(noncomp in c(0,10,15,-1)) {
  if(noncomp>=0) nc.i <- nc.c <- noncomp else {nc.i <- 25; nc.c <- 15}
  z <- paste("Drop-in ",nc.c,"%, Non-adherence ",nc.i,"%",sep="")
  plot(0,0,xlim=range(morts),ylim=c(0,1),
           xlab="5-year Mortality in Control Patients (%)",
           ylab="Power",type="n")
  title(z)
  cat(z,"\n")
  lty <- 0
  for(r in red) {
        lty <- lty+1
        power <- morts
        i <- 0
        for(m in morts) {
          i <- i+1
          power[i] <- cpower(5, 14000, m/100, r, 1.5, 5, nc.c, nc.i, pr=FALSE)
        }
        lines(morts, power, lty=lty)
  }
  if(noncomp==0)legend(18,.55,rev(paste(red,"% reduction",sep="")),
           lty=4:1,bty="n")
}
mtitle("Power vs Non-Adherence for Main Comparison",
           ll="alpha=.05, 2-tailed, Total N=14000",cex.l=.8)
#
# Point sample size requirement vs. mortality reduction
# Root finder (uniroot()) assumes needed sample size is between
# 1000 and 40000
#
nc.i <- 25; nc.c <- 15; mort <- .18
red <- seq(10,25,by=.25)
samsiz <- red

i <- 0
for(r in red) {
  i <- i+1
  samsiz[i] <- uniroot(function(x) cpower(5, x, mort, r, 1.5, 5,
                                          nc.c, nc.i, pr=FALSE) - .8,
                       c(1000,40000))$root
}

samsiz <- samsiz/1000
par(mfrow=c(1,1))
plot(red, samsiz, xlab='% Reduction in 5-Year Mortality',
         ylab='Total Sample Size (Thousands)', type='n')
lines(red, samsiz, lwd=2)
title('Sample Size for Power=0.80\nDrop-in 15%, Non-adherence 25%')
title(sub='alpha=0.05, 2-tailed', adj=0)
```

`csv.get` *Read Comma-Separated Text Data Files*

Description

Read comma-separated text data files, allowing optional translation to lower case for variable names after making them valid S names. There is a facility for reading long variable labels as one of the rows. If labels are not specified and a final variable name is not the same as that in the header, the original variable name is saved as a variable label.

Usage

```
csv.get(file, lowernames=FALSE, datevars=NULL, datetimevars=NULL,
        dateformat='%F',
        fixdates=c('none','year'), comment.char="", autodates=TRUE,
        allow=NULL, charfactor=FALSE,
        sep=',', skip=0, vnames=NULL, labels=NULL, ...)
```

Arguments

`file`	the file name for import.
`lowernames`	set this to `TRUE` to change variable names to lower case.
`datevars`	character vector of names (after `lowernames` is applied) of variables to consider as a factor or character vector containing dates in a format matching `dateformat`. The default is `"%F"` which uses the yyyy-mm-dd format.
`datetimevars`	character vector of names (after `lowernames` is applied) of variables to consider to be date-time variables, with date formats as described under `datevars` followed by a space followed by time in hh:mm:ss format. `chron` is used to store such variables. If all times in the variable are 00:00:00 the variable will be converted to an ordinary date variable.
`dateformat`	for `cleanup.import` is the input format (see `strptime`)
`fixdates`	for any of the variables listed in `datevars` that have a `dateformat` that `cleanup.import` understands, specifying `fixdates` allows corrections of certain formatting inconsistencies before the fields are attempted to be converted to dates (the default is to assume that the `dateformat` is followed for all observation for `datevars`). Currently `fixdates='year'` is implemented, which will cause 2-digit or 4-digit years to be shifted to the alternate number of digits when `dateform` is the default `"%F"` or is `"%y-%m-%d"`, `"%m/%d/%y"`, or `"%m/%d/%Y"`. Two-digits years are padded with `20` on the left. Set `dateformat` to the desired format, not the exceptional format.
`comment.char`	a character vector of length one containing a single character or an empty string. Use '""' to turn off the interpretation of comments altogether.
`autodates`	Set to true to allow function to guess at which variables are dates
`allow`	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
`charfactor`	set to `TRUE` to change character variables to factors if they have at least two characters in an observation but have fewer than n/2 unique values
`sep`	field separator, defaults to comma
`skip`	number of records to skip before data start. Required if `vnames` or `labels` is given.
`vnames`	number of row containing variable names, default is one
`labels`	number of row containing variable labels, default is no labels
`...`	arguments to pass to `read.csv` other than `skip` and `sep`.

Details

csv.get reads comma-separated text data files, allowing optional translation to lower case for variable names after making them valid S names. Original possibly non-legal names are taken to be variable labels if labels is not specified. Character or factor variables containing dates can be converted to date variables. cleanup.import is invoked to finish the job.

Value

a new data frame.

Author(s)

Frank Harrell, Vanderbilt University

See Also

sas.get, data.frame, cleanup.import, read.csv, strptime, POSIXct, Date

Examples

```
## Not run:
dat <- csv.get('myfile.csv')

# Read a csv file with junk in the first row, variable names in the
# second, long variable labels in the third, and junk in the 4th row
dat <- csv.get('myfile.csv', vnames=2, labels=3, skip=4)
## End(Not run)
```

---

curveRep *Representative Curves*

Description

curveRep finds representative curves from a relatively large collection of curves. The curves usually represent time-response profiles as in serial (longitudinal or repeated) data with possibly unequal time points and greatly varying sample sizes per subject. After excluding records containing missing x or y, records are first stratified into kn groups having similar sample sizes per curve (subject). Within these strata, curves are next stratified according to the distribution of x points per curve (typically measurement times per subject). The clara clustering/partitioning function is used to do this, clustering on one, two, or three x characteristics depending on the minimum sample size in the current interval of sample size. If the interval has a minimum number of unique values of one, clustering is done on the single x values. If the minimum number of unique x values is two, clustering is done to create groups that are similar on both min(x) and max(x). For groups containing no fewer than three unique x values, clustering is done on the trio of values min(x), max(x), and the longest gap between any successive x. Then within sample size and x distribution strata, clustering of time-response profiles is based on p values of y all evaluated at the same p equally-spaced x's within the stratum. An option allows per-curve data to be smoothed with lowess before proceeding. Outer x values are taken as extremes of x across all curves within the stratum. Linear interpolation within curves is used to estimate y at the grid of x's. For curves within the stratum that do not extend to the most extreme x values in that stratum, extrapolation uses flat lines from the observed extremes in the curve unless extrap=TRUE. The p y values are clustered using clara.

print and plot methods show results. By specifying an auxiliary idcol variable to plot, other variables such as treatment may be depicted to allow the analyst to determine for example whether subjects on different treatments are assigned to different time-response profiles. To write the frequencies of a variable such as treatment in the upper left corner of each panel (instead of the grand total number of clusters in that panel), specify freq.

curveSmooth takes a set of curves and smooths them using lowess. If the number of unique x points in a curve is less than p, the smooth is evaluated at the unique x values. Otherwise it is evaluated at an equally spaced set of x points over the observed range. If fewer than 3 unique xs are in a curve, those points are used and smoothing is not done.

Usage

```
curveRep(x, y, id, kn = 5, kxdist = 5, k = 5, p = 5,
         force1 = TRUE, metric = c("euclidean", "manhattan"),
         smooth=FALSE, extrap=FALSE, pr=FALSE)

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

## S3 method for class 'curveRep':
plot(x, which=1:length(res), method=c('all','lattice'),
                        m=NULL, probs=c(.5, .25, .75), nx=NULL, fill=TRUE,
                        idcol=NULL, freq=NULL, plotfreq=FALSE,
                        xlim=range(x), ylim=range(y),
                        xlab='x', ylab='y', ...)
curveSmooth(x, y, id, p=NULL, pr=TRUE)
```

Arguments

x	a numeric vector, typically measurement times. For plot.curveRep is an object created by curveRep.
y	a numeric vector of response values
id	a vector of curve (subject) identifiers, the same length as x and y
kn	number of curve sample size groups to construct. curveRep tries to divide the data into equal numbers of curves across sample size intervals.
kxdist	maximum number of x-distribution clusters to derive using clara
k	maximum number of x-y profile clusters to derive using clara
p	number of x points at which to interpolate y for profile clustering. For curveSmooth is the number of equally spaced points at which to evaluate the lowess smooth, and if p is omitted the smooth is evaluated at the original x values (which will allow curveRep to still know the x distribution

`force1`	By default if any curves have only one point, all curves consisting of one point will be placed in a separate stratum. To prevent this separation, set `force1=FALSE`.
`metric`	see `clara`
`smooth`	By default, linear interpolation is used on raw data to obtain `y` values to cluster to determine x-y profiles. Specify `smooth=TRUE` to replace observed points with `lowess` before computing `y` points on the grid. Also, when `smooth` is used, it may be desirable to use `extrap=TRUE`.
`extrap`	set to `TRUE` to use linear extrapolation to evaluate `y` points for x-y clustering. Not recommended unless smoothing has been or is being done.
`pr`	set to `TRUE` to print progress notes
`which`	an integer vector specifying which sample size intervals to plot. Must be specified if `method='lattice'` and must be a single number in that case.
`method`	The default makes individual plots of possibly all x-distribution by sample size by cluster combinations. Fewer may be plotted by specifying `which`. Specify `method='lattice'` to show a lattice `xyplot` of a single sample size interval, with x distributions going across and clusters going down.
`m`	the number of curves in a cluster to randomly sample if there are more than `m` in a cluster. Default is to draw all curves in a cluster. For `method='lattice'` you can specify `m='quantiles'` to use the `xYplot` function to show quantiles of `y` as a function of x, with the quantiles specified by the `probs` argument. This cannot be used to draw a group containing `n=1`.
`nx`	applies if `m='quantiles'`. See `xYplot`.
`probs`	3-vector of probabilities with the central quantile first. Default uses quartiles.
`fill`	for `method='all'`, by default if a sample size x-distribution stratum did not have enough curves to stratify into `k` x-y profiles, empty graphs are drawn so that a matrix of graphs will have the next row starting with a different sample size range or x-distribution. See the example below.
`idcol`	a named vector to be used as a table lookup for color assignments (does not apply when `m='quantile'`). The names of this vector are curve `id`s and the values are color names or numbers.
`freq`	a named vector to be used as a table lookup for a grouping variable such as treatment. The names are curve `id`s and values are any values useful for grouping in a frequency tabulation.
`plotfreq`	set to `TRUE` to plot the frequencies from the `freq` variable as horizontal bars instead of printing them. Applies only to `method='lattice'`. By default the largest bar is 0.1 times the length of a panel's x-axis. Specify `plotfreq=.5` for example to make the longest bar half this long.
`xlim, ylim, xlab, ylab`	plotting parameters. Default ranges are the ranges in the entire set of raw data given to `curveRep`.
`...`	arguments passed to other functions.

Details

In the graph titles for the default graphic output, `n` refers to the minimum sample size, `x` refers to the sequential x-distribution cluster, and `c` refers to the sequential x-y profile cluster. Graphs from `method='lattice'` are produced by `xyplot` and in the panel titles `distribution` refers to the x-distribution stratum and `cluster` refers to the x-y profile cluster.

Value

a list of class `'curveRep'` with the following elements

`res`	a hierarchical list first split by sample size intervals, then by x distribution clusters, then containing a vector of cluster numbers with `id` values as a names attribute
`ns`	a table of frequencies of sample sizes per curve after removing `NA`s
`nomit`	total number of records excluded due to `NA`s
`missfreq`	a table of frequencies of number of `NA`s excluded per curve
`ncuts`	cut points for sample size intervals
`kn`	number of sample size intervals
`kxdist`	number of clusters on x distribution
`k`	number of clusters of curves within sample size and distribution groups
`p`	number of points at which to evaluate each curve for clustering
`x`	
`y`	
`id`	input data after removing `NA`s

`curveSmooth` returns a list with elements `x,y,id`.

Note

The references describe other methods for deriving representative curves, but those methods were not used here. The last reference which used a cluster analysis on principal components motivated `curveRep` however.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Segal M. (1994): Representative curves for longitudinal data via regression trees. J Comp Graph Stat 3:214-233.

Jones MC, Rice JA (1992): Displaying the important features of large collections of similar curves. Am Statistician 46:140-145.

Zheng X, Simpson JA, et al (2005): Data from a study of effectiveness suggested potential prognostic factors related to the patterns of shoulder pain. J Clin Epi 58:823-830.

See Also

clara,dataRep

Examples

```
## Not run:
# Simulate 200 curves with pre-curve sample sizes ranging from 1 to 10
# Make curves with odd-numbered IDs have an x-distribution that is random
# uniform [0,1] and those with even-numbered IDs have an x-dist. that is
# half as wide but still centered at 0.5.  Shift y values higher with
# increasing IDs
set.seed(1)
N <- 200
nc <- sample(1:10, N, TRUE)
id <- rep(1:N, nc)
x <- y <- id
for(i in 1:N) {
  x[id==i] <- if(i
  y[id==i] <- i + 10*(x[id==i] - .5) + runif(nc[i], -10, 10)
}

w <- curveRep(x, y, id, kxdist=2, p=10)
w
par(ask=TRUE, mfrow=c(4,5))
plot(w)                # show everything, profiles going across
par(mfrow=c(2,5))
plot(w,1)              # show n=1 results
# Use a color assignment table, assigning low curves to green and
# high to red.  Unique curve (subject) IDs are the names of the vector.
cols <- c(rep('green', N/2), rep('red', N/2))
names(cols) <- as.character(1:N)
plot(w, 3, idcol=cols)
par(ask=FALSE, mfrow=c(1,1))

plot(w, 1, 'lattice')  # show n=1 results
plot(w, 3, 'lattice')  # show n=4-5 results
plot(w, 3, 'lattice', idcol=cols)  # same but different color mapping
plot(w, 3, 'lattice', m=1)  # show a single "representative" curve
# Show median, 10th, and 90th percentiles of supposedly representative curves
plot(w, 3, 'lattice', m='quantiles', probs=c(.5,.1,.9))
# Same plot but with much less grouping of x variable
plot(w, 3, 'lattice', m='quantiles', probs=c(.5,.1,.9), nx=2)

# Smooth data before profiling.  This allows later plotting to plot
# smoothed representative curves rather than raw curves (which
# specifying smooth=TRUE to curveRep would do, if curveSmooth was not used)
d <- curveSmooth(x, y, id)
w <- with(d, curveRep(x, y, id))
```

```
# Example to show that curveRep can cluster profiles correctly when
# there is no noise.  In the data there are four profiles - flat, flat
# at a higher mean y, linearly increasing then flat, and flat at the
# first height except for a sharp triangular peak

set.seed(1)
x <- 0:100
m <- length(x)
profile <- matrix(NA, nrow=m, ncol=4)
profile[,1] <- rep(0, m)
profile[,2] <- rep(3, m)
profile[,3] <- c(0:3, rep(3, m-4))
profile[,4] <- c(0,1,3,1,rep(0,m-4))
col <- c('black','blue','green','red')
matplot(x, profile, type='l', col=col)
xeval <- seq(0, 100, length.out=5)
s <- x
matplot(x[s], profile[s,], type='l', col=col)

id <- rep(1:100, each=m)
X <- Y <- id
cols <- character(100)
names(cols) <- as.character(1:100)
for(i in 1:100) {
  s <- id==i
  X[s] <- x
  j <- sample(1:4,1)
  Y[s] <- profile[,j]
  cols[i] <- col[j]
}
table(cols)
yl <- c(-1,4)
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=4)
plot(w, 1, 'lattice', idcol=cols, ylim=yl)
# Found 4 clusters but two have same profile
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=3)
plot(w, 1, 'lattice', idcol=cols, freq=cols, plotfreq=TRUE, ylim=yl)
# Incorrectly combined black and red because default value p=5 did
# not result in different profiles at x=xeval
w <- curveRep(X, Y, id, kn=1, kxdist=1, k=4, p=40)
plot(w, 1, 'lattice', idcol=cols, ylim=yl)
# Found correct clusters because evaluated curves at 40 equally
# spaced points and could find the sharp triangular peak in profile 4
## End(Not run)
```

`cut2` *Cut a Numeric Variable into Intervals*

Description

Function like cut but left endpoints are inclusive and labels are of the form `[lower, upper)`, except that last interval is `[lower,upper]`. If cuts are given, will by default make sure that cuts include entire range of `x`. Also, if cuts are not given, will cut `x` into quantile groups (`g` given) or groups with a given minimum number of observations (`m`). Whereas cut creates a category object, `cut2` creates a factor object.

Usage

```
cut2(x, cuts, m, g, levels.mean, digits, minmax=TRUE, oneval=TRUE, onlycuts=FALSE)
```

Arguments

`x`	numeric vector to classify into intervals
`cuts`	cut points
`m`	desired minimum number of observations in a group
`g`	number of quantile groups
`levels.mean`	set to `TRUE` to make the new categorical vector have levels attribute that is the group means of `x` instead of interval endpoint labels
`digits`	number of significant digits to use in constructing levels. Default is 3 (5 if `levels.mean=TRUE`)
`minmax`	if cuts is specified but `min(x)<min(cuts)` or `max(x)>max(cuts)`, augments cuts to include min and max `x`
`oneval`	if an interval contains only one unique value, the interval will be labeled with the formatted version of that value instead of the interval endpoints, unless `oneval=FALSE`
`onlycuts`	set to `TRUE` to only return the vector of computed cuts. This consists of the interior values plus outer ranges.

Value

a factor variable with levels of the form `[a,b)` or formatted means (character strings) unless `onlycuts` is `TRUE` in which case a numeric vector is returned

See Also

`cut`, `quantile`

Examples

```
set.seed(1)
x <- runif(1000, 0, 100)
z <- cut2(x, c(10,20,30))
table(z)
table(cut2(x, g=10))      # quantile groups
table(cut2(x, m=50))      # group x into intevals with at least 50 obs.
```

`data.frame.create.modify.check` *Tips for Creating, Modifying, and Checking Data Frames*

Description

This help file contains a template for importing data to create an S-Plus data frame, correcting some problems resulting from the import and making the data frame be stored more efficiently, modifying the data frame (including better annotating it and changing the names of some of its variables), and checking and inspecting the data frame for reasonableness of the values of its variables and to describe patterns of missing data. Various built-in functions and functions in the Hmisc library are used. At the end some methods for creating data frames "from scratch" within S-Plus are presented.

The examples below attempt to clarify the separation of operations that are done on a data frame as a whole, operations that are done on a small subset of its variables without attaching the whole data frame, and operations that are done on many variables after attaching the data frame in search position one. It also tries to clarify that for analyzing several separate variables using S-Plus commands that do not support a `data=` argument, it is helpful to attach the data frame in a search position later than position one.

It is often useful to create, modify, and process datasets in the following order.
- Import external data into a data frame (if the raw data do not contain column names, provide these during the import if possible)
- Make global changes to a data frame (e.g., changing variable names)
- Change attributes or values of variables within a data frame
- Do analyses involving the whole data frame (without attaching it)
(Data frame still in .Data)
- Do analyses of individual variables (after attaching the data frame in search position two or later)

Details

The examples below use the `FEV` dataset from Rosner B (1995): *Fundamentals of Biostatistics, 4th Edition.* New York: Duxbury Press. Almost any dataset would do. The jcetable data are taken from Galobardes, et al. (1998), *J Clin Epi* 51:875-881.

Presently, giving a variable the `"units"` attribute (using the Hmisc `units` function) only benefits the Hmisc `describe` function and the Design library's version of the `Surv` function. Variables labels defined with the Hmisc `label` function are used by `describe`, `summary.formula`, and many of the plotting functions in Hmisc and Design.

References

Alzola CF, Harrell FE (2001): *An Introduction to S-Plus and the Hmisc and Design Libraries.* Chapters 3 and 4, `http://biostat.mc.vanderbilt.edu/twiki/pub/Main/RS/sintro.pdf`.

See Also

scan, read.table, cleanup.import, sas.get, data.frame, attach, detach, describe, datadensity, plot.data.frame, hist.data.frame, naclus, factor, label, units, names, expand.grid, summary.formula, summary.data.frame, casefold, edit, page, plot.data.frame, Cs, combine.levels, upData

Examples

```
## Not run: 
# First, we do steps that create or manipulate the data
# frame in its entirety.  These are done with .Data
# in search position one (the S-Plus default at the
# start of the session).
#
# -----------------------------------------------------------------------
# Step 1: Create initial draft of data frame
# 
# We usually begin by importing a dataset from
# # another application.  ASCII files may be imported
# using the scan and read.table functions.  SAS
# datasets may be imported using the Hmisc sas.get
# function (which will carry more attributes from
# SAS than using File ...  Import) from the GUI
# menus.  But for most applications (especially
# Excel), File ... Import will suffice.  If using
# the GUI, it is often best to provide variable
# names during the import process, using the Options
# tab, rather than renaming all fields later Of
# course, if the data to be imported already have
# field names (e.g., in Excel), let S-Plus use those
# automatically.  If using S-Plus 4.x on Windows/NT,
# you can use a command to execute File ...  Import,
# e.g.:

import.data(FileName = "/windows/temp/fev.asc",
            FileType = "ASCII", DataFrame = "FEV")

# Here we name the new data frame FEV rather than
# fev, because we wanted to distinguish a variable
# in the data frame named fev from the data frame
# name.  For S-Plus 6.x the command will look
# instead like the following:

FEV <- importData("/tmp/fev.asc")



# -----------------------------------------------------------------------
# Step 2: Clean up data frame / make it be more
# efficiently stored
#
# Unless using sas.get to import your dataset
# (sas.get already stores data efficiently), it is
# usually a good idea to run the data frame through
# the Hmisc cleanup.import function to change
# numeric variables that are always whole numbers to
# be stored as integers, the remaining numerics to
# single precision, strange values from Excel to
# NAs, and character variables that always contain
# legal numeric values to numeric variables.
# cleanup.import typically halves the size of the
# data frame.  If you do not specify any parameters
# to cleanup.import, the function assumes that no
# numeric variable needs more than 7 significant
# digits of precision, so all non-integer-valued
# variables will be converted to single precision.

FEV <- cleanup.import(FEV)



# -----------------------------------------------------------------------
# Step 3: Make global changes to the data frame
#
# A data frame has attributes that are "external" to
# its variables.  There are the vector of its
# variable names ("names" attribute), the
# observation identifiers ("row.names"), and the
# "class" (whose value is "data.frame").  The
# "names" attribute is the one most commonly in need
# of modification.  If we had wanted to change all
# the variable names to lower case, we could have
# specified lowernames=TRUE to the cleanup.import
# invocation above, or type

names(FEV) <- casefold(names(FEV))

# The upData function can also be used to change
# variable names in two ways (see below).
# To change names in a non-systematic way we use
# other options.  Under Windows/NT the most
# straigtforward approach is to change the names
# interactively.  Click on the data frame in the
# left panel of the Object Browser, then in the
# right pane click twice (slowly) on a variable.
# Use the left arrow and other keys to edit the
# name.  Click outside that name field to commit the
# change.  You can also rename columns while in a
# Data Sheet.  To instead use programming commands
# to change names, use something like:

names(FEV)[6] <- 'smoke'   # assumes you know the positions!
names(FEV)[names(FEV)=='smoking'] <- 'smoke'
names(FEV) <- edit(names(FEV))
```

```
# The last example is useful if you are changing
# many names.  But none of the interactive
# approaches such as edit() are handy if you will be
# re-importing the dataset after it is updated in
# its original application.  This problem can be
# addressed by saving the new names in a permanent
# vector in .Data:

new.names <- names(FEV)

# Then if the data are re-imported, you can type

names(FEV) <- new.names

# to rename the variables.



# -------------------------------------------------------------------------
# Step 4: Delete unneeded variables
#
# To delete some of the variables, you can
# right-click on variable names in the Object
# Browser's right pane, then select Delete.  You can
# also set variables to have NULL values, which
# causes the system to delete them.  We don't need
# to delete any variables from FEV but suppose we
# did need to delete some from mydframe.

mydframe$x1 <- NULL
mydframe$x2 <- NULL
mydframe[c('age','sex')] <- NULL   # delete 2 variables
mydframe[Cs(age,sex)]    <- NULL   # same thing

# The last example uses the Hmisc short-cut quoting
# function Cs.  See also the drop parameter to upData.



# -------------------------------------------------------------------------
# Step 5: Make changes to individual variables
#         within the data frame
#
# After importing data, the resulting variables are
# seldom self - documenting, so we commonly need to
# change or enhance attributes of individual
# variables within the data frame.
#
# If you are only changing a few variables, it is
# efficient to change them directly without
# attaching the entire data frame.

FEV$sex   <- factor(FEV$sex,   0:1, c('female','male'))
```

```
FEV$smoke <- factor(FEV$smoke, 0:1,
                    c('non-current smoker','current smoker'))
units(FEV$age)    <- 'years'
units(FEV$fev)    <- 'L'
label(FEV$fev)    <- 'Forced Expiratory Volume'
units(FEV$height) <- 'inches'

# When changing more than one or two variables it is
# more convenient change the data frame using the
# Hmisc upData function.

FEV2 <- upData(FEV,
  rename=c(smoking='smoke'),
  # omit if renamed above
  drop=c('var1','var2'),
  levels=list(sex  =list(female=0,male=1),
              smoke=list('non-current smoker'=0,
                         'current smoker'=1)),
  units=list(age='years', fev='L', height='inches'),
  labels=list(fev='Forced Expiratory Volume'))

# An alternative to levels=list(...) is for example
# upData(FEV, sex=factor(sex,0:1,c('female','male'))).
#
# Note that we saved the changed data frame into a
# new data frame FEV2.  If we were confident of the
# correctness of our changes we could have stored
# the new data frame on top of the old one, under
# the original name FEV.

# -----------------------------------------------------------------------
# Step 6:  Check the data frame
#
# The Hmisc describe function is perhaps the first
# function that should be used on the new data
# frame.  It provides documentation of all the
# variables and the frequency tabulation, counts of
# NAs,  and 5 largest and smallest values are
# helpful in detecting data errors.  Typing
# describe(FEV) will write the results to the
# current output window.  To put the results in a
# new window that can persist, even upon exiting
# S-Plus, we use the page function.  The describe
# output can be minimized to an icon but kept ready
# for guiding later steps of the analysis.

page(describe(FEV2), multi=TRUE)
# multi=TRUE allows that window to persist while
# control is returned to other windows

# The new data frame is OK.  Store it on top of the
# old FEV and then use the graphical user interface
# to delete FEV2 (click on it and hit the Delete
# key) or type rm(FEV2) after the next statement.

FEV <- FEV2

# Next, we can use a variety of other functions to
# check and describe all of the variables.  As we
# are analyzing all or almost all of the variables,
# this is best done without attaching the data
# frame.  Note that plot.data.frame plots inverted
# CDFs for continuous variables and dot plots
# showing frequency distributions of categorical
# ones.

summary(FEV)
# basic summary function (summary.data.frame)

plot(FEV)                # plot.data.frame
datadensity(FEV)
# rug plots and freq. bar charts for all var.

hist.data.frame(FEV)
# for variables having > 2 values

by(FEV, FEV$smoke, summary)
# use basic summary function with stratification



# ------------------------------------------------------------------------
# Step 7:  Do detailed analyses involving individual
#          variables
#
# Analyses based on the formula language can use
# data= so attaching the data frame may not be
# required.  This saves memory.  Here we use the
# Hmisc summary.formula function to compute 5
# statistics on height, stratified separately by age
# quartile and by sex.

options(width=80)
summary(height ~ age + sex, data=FEV,
        fun=function(y)c(smean.sd(y),
                         smedian.hilow(y,conf.int=.5)))
# This computes mean height, S.D., median, outer quartiles

fit <- lm(height ~ age*sex, data=FEV)
summary(fit)

# For this analysis we could also have attached the
# data frame in search position 2.  For other
# analyses, it is mandatory to attach the data frame
# unless FEV$ prefixes each variable name.
# Important: DO NOT USE attach(FEV, 1) or
# attach(FEV, pos=1, ...) if you are only analyzing
# and not changing the variables, unless you really
# need to avoid conflicts with variables in search
# position 1 that have the same names as the
# variables in FEV.  Attaching into search position
# 1 will cause S-Plus to be more of a memory hog.

attach(FEV)
# Use e.g. attach(FEV[,Cs(age,sex)]) if you only
# want to analyze a small subset of the variables
# Use e.g. attach(FEV[FEV$sex=='male',]) to
# analyze a subset of the observations

summary(height ~ age + sex,
        fun=function(y)c(smean.sd(y),
          smedian.hilow(y,conf.int=.5)))
fit <- lm(height ~ age*sex)

# Run generic summary function on height and fev,
# stratified by sex
by(data.frame(height,fev), sex, summary)

# Cross-classify into 4 sex x smoke groups
by(FEV, list(sex,smoke), summary)

# Plot 5 quantiles
s <- summary(fev ~ age + sex + height,
              fun=function(y)quantile(y,c(.1,.25,.5,.75,.9)))

plot(s, which=1:5, pch=c(1,2,15,2,1), #pch=c('=','[','o',']','='),
     main='A Discovery', xlab='FEV')

# Use the nonparametric bootstrap to compute a
# 0.95 confidence interval for the population mean fev
smean.cl.boot(fev)    # in Hmisc

# Use the Statistics ... Compare Samples ... One Sample
# keys to get a normal-theory-based C.I.  Then do it
# more manually.  The following method assumes that
# there are no NAs in fev

sd <- sqrt(var(fev))
xbar <- mean(fev)
xbar
sd
n <- length(fev)
qt(.975,n-1)
# prints 0.975 critical value of t dist. with n-1 d.f.

xbar + c(-1,1)*sd/sqrt(n)*qt(.975,n-1)
# prints confidence limits

# Fit a linear model
# fit <- lm(fev ~ other variables ...)

detach()

# The last command is only needed if you want to
# start operating on another data frame and you want
# to get FEV out of the way.



# -------------------------------------------------------------------------
# Creating data frames from scratch
#
# Data frames can be created from within S-Plus.  To
# create a small data frame containing ordinary
# data, you can use something like

dframe <- data.frame(age=c(10,20,30),
                     sex=c('male','female','male'))

# You can also create a data frame using the Data
# Sheet.  Create an empty data frame with the
# correct variable names and types, then edit in the
# data.

dd <- data.frame(age=numeric(0),sex=character(0))

# The sex variable will be stored as a factor, and
# levels will be automatically added to it as you
# define new values for sex in the Data Sheet's sex
# column.
#
# When the data frame you need to create is defined
# by systematically varying variables (e.g., all
# possible combinations of values of each variable),
# the expand.grid function is useful for quickly
# creating the data.  Then you can add
# non-systematically-varying variables to the object
# created by expand.grid, using programming
# statements or editing the Data Sheet.  This
# process is useful for creating a data frame
# representing all the values in a printed table.
# In what follows we create a data frame
# representing the combinations of values from an 8
# x 2 x 2 x 2 (event x method x sex x what) table,
# and add a non-systematic variable percent to the
# data.

jcetable <- expand.grid(
 event=c('Wheezing at any time',
         'Wheezing and breathless',
         'Wheezing without a cold',
         'Waking with tightness in the chest',
            'Waking with shortness of breath',
            'Waking with an attack of cough',
            'Attack of asthma',
            'Use of medication'),
    method=c('Mail','Telephone'),
    sex=c('Male','Female'),
    what=c('Sensitivity','Specificity'))

jcetable$percent <-
c(756,618,706,422,356,578,289,333,
  576,421,789,273,273,212,212,212,
  613,763,713,403,377,541,290,226,
  613,684,632,290,387,613,258,129,
  656,597,438,780,732,679,938,919,
  714,600,494,877,850,703,963,987,
  755,420,480,794,779,647,956,941,
  766,423,500,833,833,604,955,986) / 10

# In jcetable, event varies most rapidly, then
# method, then sex, and what.
## End(Not run)
```

`dataRep` *Representativeness of Observations in a Data Set*

Description

These functions are intended to be used to describe how well a given set of new observations (e.g., new subjects) were represented in a dataset used to develop a predictive model. The `dataRep` function forms a data frame that contains all the unique combinations of variable values that existed in a given set of variable values. Cross–classifications of values are created using exact values of variables, so for continuous numeric variables it is often necessary to round them to the nearest `v` and to possibly curtail the values to some lower and upper limit before rounding. Here `v` denotes a numeric constant specifying the matching tolerance that will be used. `dataRep` also stores marginal distribution summaries for all the variables. For numeric variables, all 101 percentiles are stored, and for all variables, the frequency distributions are also stored (frequencies are computed after any rounding and curtailment of numeric variables). For the purposes of rounding and curtailing, the `roundN` function is provided. A `print` method will summarize the calculations made by `dataRep`, and if `long=TRUE` all unique combinations of values and their frequencies in the original dataset are printed.

The `predict` method for `dataRep` takes a new data frame having variables named the same as the original ones (but whose factor levels are not necessarily in the same order) and examines the collapsed cross-classifications created by `dataRep` to find how many observations were similar to each of the new observations after any rounding or curtailment of limits is done. `predict` also does some calculations to describe how the variable values of the new observations "stack up" against the marginal distributions of the original data. For categorical variables, the percent of observations having a given variable with the value of the new observation (after rounding for variables that were through `roundN` in the formula given to `dataRep`) is computed. For numeric variables, the percentile of the original distribution in which the current value falls will be computed. For this purpose, the data are not rounded because the 101 original percentiles were retained; linear interpolation is used to estimate percentiles for values between two tabulated percentiles. The lowest marginal frequency of matching values across all variables is also computed. For example, if an age, sex combination matches 10 subjects in the original dataset but the age value matches 100 ages (after rounding) and the sex value matches the sex code of 300 observations, the lowest marginal frequency is 100, which is a "best case" upper limit for multivariable matching. I.e., matching on all variables has to result on a lower frequency than this amount. A `print` method for the output of `predict.dataRep` prints all calculations done by `predict` by default. Calculations can be selectively suppressed.

Usage

```
dataRep(formula, data, subset, na.action)

roundN(x, tol=1, clip=NULL)

## S3 method for class 'dataRep':
print(x, long=FALSE, ...)

## S3 method for class 'dataRep':
predict(object, newdata, ...)

## S3 method for class 'predict.dataRep':
print(x, prdata=TRUE, prpct=TRUE, ...)
```

Arguments

`formula`	a formula with no left-hand-side. Continuous numeric variables in need of rounding should appear in the formula as e.g. `roundN(x,5)` to have a tolerance of e.g. +/- 2.5 in matching. Factor or character variables as well as numeric ones not passed through `roundN` are matched on exactly.
`x`	a numeric vector or an object created by `dataRep`
`object`	the object created by `dataRep` or `predict.dataRep`
`data, subset, na.action`	standard modeling arguments. Default `na.action` is `na.delete`, i.e., observations in the original dataset having any variables missing are deleted up front.
`tol`	rounding constant (tolerance is actually `tol/2` as values are rounded to the nearest `tol`)
`clip`	a 2-vector specifying a lower and upper limit to curtail values of `x` before rounding
`long`	set to `TRUE` to see all unique combinations and frequency count
`newdata`	a data frame containing all the variables given to `dataRep` but not necessarily in the same order or having factor levels in the same order
`prdata`	set to `FALSE` to suppress printing `newdata` and the count of matching observations (plus the worst-case marginal frequency).

`prpct`	set to `FALSE` to not print percentiles and percents
`...`	unused

Value

`dataRep` returns a list of class `"dataRep"` containing the collapsed data frame and frequency counts along with marginal distribution information. `predict` returns an object of class `"predict.dataRep"` containing information determined by matching observations in `newdata` with the original (collapsed) data.

Side Effects

`print.dataRep` prints.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

See Also

`round`, `table`

Examples

```
set.seed(13)
num.symptoms <- sample(1:4, 1000,TRUE)
sex <- factor(sample(c('female','male'), 1000,TRUE))
x    <- runif(1000)
x[1] <- NA
table(num.symptoms, sex, .25*round(x/.25))

d <- dataRep(~ num.symptoms + sex + roundN(x,.25))
print(d, long=TRUE)

predict(d, data.frame(num.symptoms=1:3, sex=c('male','male','female'),
                      x=c(.03,.5,1.5)))
```

`deff` *Design Effect and Intra-cluster Correlation*

Description

Computes the Kish design effect and corresponding intra-cluster correlation for a single cluster-sampled variable

Usage

```
deff(y, cluster)
```

Arguments

`y`	variable to analyze
`cluster`	a variable whose unique values indicate cluster membership. Any type of variable is allowed.

Value

a vector with named elements `n` (total number of non-missing observations), `clusters` (number of clusters after deleting missing data), `rho` (intra-cluster correlation), and `deff` (design effect).

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`bootcov`, `robcov`

Examples

```
set.seed(1)
blood.pressure <- rnorm(1000, 120, 15)
clinic <- sample(letters, 1000, replace=TRUE)
deff(blood.pressure, clinic)
```

`describe` *Concise Statistical Description of a Vector, Matrix, Data Frame, or Formula*

Description

`describe` is a generic method that invokes `describe.data.frame`, `describe.matrix`, `describe.vector`, or `describe.formula`. `describe.vector` is the basic function for handling a single variable. This function determines whether the variable is character, factor, category, binary, discrete numeric, and continuous numeric, and prints a concise statistical summary according to each. A numeric variable is deemed discrete if it has <= 10 unique values. In this case, quantiles are not printed. A frequency table is printed for any non-binary variable if it has no more than 20 unique values. For any variable with at least 20 unique values, the 5 lowest and highest values are printed. This behavior can be overriden for long character variables with many levels using the `listunique` parameter, to get a complete tabulation.

`describe` is especially useful for describing data frames created by `*.get`, as labels, formats, value labels, and (in the case of `sas.get`) frequencies of special missing values are printed.

For a binary variable, the sum (number of 1's) and mean (proportion of 1's) are printed. If the first argument is a formula, a model frame is created and passed to describe.data.frame. If a variable is of class `"impute"`, a count of the number of imputed values is printed. If a date variable has an attribute `partial.date` (this is set up by `sas.get`), counts of how many partial dates are actually present (missing month, missing day, missing both) are also presented. If a variable was created by the special-purpose function `substi` (which substitutes values of a second variable if the first variable is NA), the frequency table of substitutions is also printed.

A latex method exists for converting the `describe` object to a LaTeX file. For numeric variables having at least 20 unique values, `describe` saves in its returned object the frequencies of 100 evenly spaced bins running from minimum observed value to the maximum. `latex` inserts a spike histogram displaying these frequency counts in the tabular material using the LaTeX picture environment. For example output see `http://biostat.mc.vanderbilt.edu/twiki/pub/Main/Hmisc/counties.pdf`. Note that the latex method assumes you have the following styles installed in your latex installation: setspace and relsize.

Sample weights may be specified to any of the functions, resulting in weighted means, quantiles, and frequency tables.

Usage

```
## S3 method for class 'vector':
describe(x, descript, exclude.missing=TRUE, digits=4,
         listunique=0, listnchar=12,
         weights=NULL, normwt=FALSE, minlength=NULL, ...)
## S3 method for class 'matrix':
describe(x, descript, exclude.missing=TRUE, digits=4, ...)
## S3 method for class 'data.frame':
describe(x, descript, exclude.missing=TRUE,
    digits=4, ...)
## S3 method for class 'formula':
describe(x, descript, data, subset, na.action,
    digits=4, weights, ...)
## S3 method for class 'describe':
print(x, condense=TRUE, ...)
## S3 method for class 'describe':
latex(object, title=NULL, condense=TRUE,
      file=paste('describe',first.word(expr=attr(object,'descript')),'tex',sep='.')
      append=FALSE, size='small', tabular=TRUE, greek=TRUE, ...)
## S3 method for class 'describe.single':
latex(object, title=NULL, condense=TRUE, vname,
      file, append=FALSE, size='small', tabular=TRUE, greek=TRUE, ...)
```

Arguments

`x`	a data frame, matrix, vector, or formula. For a data frame, the `describe.data.frame` function is automatically invoked. For a matrix, `describe.matrix` is called. For a formula, describe.data.frame(model.frame(x)) is invoked. The formula

	may or may not have a response variable. For `print` or `latex`, `x` is an object created by `describe`.
`descript`	optional title to print for x. The default is the name of the argument or the "label" attributes of individual variables. When the first argument is a formula, `descript` defaults to a character representation of the formula.
`exclude.missing`	set toTRUE to print the names of variables that contain only missing values. This list appears at the bottom of the printout, and no space is taken up for such variables in the main listing.
`digits`	number of significant digits to print
`listunique`	For a character variable that is not an `mChoice` variable, that has its longest string length greater than `listnchar`, and that has no more than `listunique` unique values, all values are listed in alphabetic order. Any value having more than one occurrence has the frequency of occurrence after it, in parentheses. Specify `listunique` equal to some value at least as large as the number of observations to ensure that all character variables will have all their values listed. For purposes of tabulating character strings, multiple white spaces of any kind are translated to a single space, leading and trailing white space are ignored, and case is ignored.
`listnchar`	see `listunique`
`weights`	a numeric vector of frequencies or sample weights. Each observation will be treated as if it were sampled `weights` times.
`minlength`	value passed to summary.mChoice.
`normwt`	The default, `normwt=FALSE` results in the use of `weights` as weights in computing various statistics. In this case the sample size is assumed to be equal to the sum of `weights`. Specify `normwt=TRUE` to divide `weights` by a constant so that `weights` sum to the number of observations (length of vectors specified to `describe`). In this case the number of observations is taken to be the actual number of records given to `describe`.
`object`	a result of `describe`
`title`	unused
`condense`	default isTRUE to condense the output with regard to the 5 lowest and highest values and the frequency table
`data`	
`subset`	
`na.action`	There are used if a formula is specified. `na.action` defaults to `na.retain` which does not delete any `NA`s from the data frame. Use `na.action=na.omit` or `na.delete` to drop any observation with any `NA` before processing.
`...`	arguments passed to `describe.default` which are passed to calls to `format` for numeric variables. For example if using R `POSIXct` or `Date` date/time formats, specifying `describe(d,format='%d%b%y')` will print date/time variables as `"01Jan2000"`. This is useful for omitting the time component. See the help file for `format.POSIXct` or `format.Date` for more information. For `latex` methods, ... is ignored.

`file`	name of output file (should have a suffix of .tex). Default name is formed from the first word of the `descript` element of the `describe` object, prefixed by `"describe"`. Set `file=""` to send LaTeX code to standard output instead of a file.
`append`	set to `TRUE` to have `latex` append text to an existing file named `file`
`size`	LaTeX text size (`"small"`, the default, or `"normalsize"`, `"tiny"`, `"scriptsize"`, etc.) for the `describe` output in LaTeX.
`tabular`	set to `FALSE` to use verbatim rather than tabular environment for the summary statistics output. By default, tabular is used if the output is not too wide.
`greek`	By default, the `latex` methods will change LaTeX names of greek letters that appear in variable labels to appropriate LaTeX symbols in math mode unless `greek=FALSE`. `greek=TRUE` is not implemented in S-Plus versions older than 6.2.
`vname`	unused argument in `latex.describe.single`

Details

If `options(na.detail.response=TRUE)` has been set and `na.action` is `"na.delete"` or `"na.keep"`, summary statistics on the response variable are printed separately for missing and non-missing values of each predictor. The default summary function returns the number of non-missing response values and the mean of the last column of the response values, with a `names` attribute of `c("N","Mean")`. When the response is a `Surv` object and the mean is used, this will result in the crude proportion of events being used to summarize the response. The actual summary function can be designated through `options(na.fun.response = "function name")`.

Value

a list containing elements `descript`, `counts`, `values`. The list is of class `describe`. If the input object was a matrix or a data frame, the list is a list of lists, one list for each variable analyzed. `latex` returns a standard `latex` object. For numeric variables having at least 20 unique values, an additional component `intervalFreq`. This component is a list with two elements, `range` (containing two values) and `count`, a vector of 100 integer frequency counts.

Author(s)

Frank Harrell
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`sas.get`, `quantile`, `table`, `summary`, `model.frame.default`, `naprint`, `lapply`, `tapply`, `Surv`, `na.delete`, `na.keep`, `na.detail.response`, `latex`

Examples

```
set.seed(1)
describe(runif(200),dig=2)    #single variable, continuous
```

```
                              #get quantiles .05,.10,...

dfr <- data.frame(x=rnorm(400),y=sample(c('male','female'),400,TRUE))
describe(dfr)

## Not run:
d <- sas.get(".","mydata",special.miss=TRUE,recode=TRUE)
describe(d)      #describe entire data frame
attach(d, 1)
describe(relig)  #Has special missing values .D .F .M .R .T
                 #attr(relig,"label") is "Religious preference"

#relig : Religious preference  Format:relig
#    n missing  D  F M R T unique
# 4038     263 45 33 7 2 1      8
#
#0:none (251, 6%), 1:Jewish (372, 9%), 2:Catholic (1230, 30%)
#3:Jehovah's Witnes (25, 1%), 4:Christ Scientist (7, 0%)
#5:Seventh Day Adv (17, 0%), 6:Protestant (2025, 50%), 7:other (111, 3%)

# Method for describing part of a data frame:
 describe(death.time ~ age*sex + rcs(blood.pressure))
 describe(~ age+sex)
 describe(~ age+sex, weights=freqs)  # weighted analysis

 fit <- lrm(y ~ age*sex + log(height))
 describe(formula(fit))
 describe(y ~ age*sex, na.action=na.delete)
# report on number deleted for each variable
 options(na.detail.response=TRUE)
# keep missings separately for each x, report on dist of y by x=NA
 describe(y ~ age*sex)
 options(na.fun.response="quantile")
 describe(y ~ age*sex)   # same but use quantiles of y by x=NA

 d <- describe(my.data.frame)
 d$age                   # print description for just age
 d[c('age','sex')]       # print description for two variables
 d[sort(names(d))]       # print in alphabetic order by var. names
 d2 <- d[20:30]          # keep variables 20-30
 page(d2)                # pop-up window for these variables

# Test date/time formats and suppression of times when they don't vary
 library(chron)
 d <- data.frame(a=chron((1:20)+.1),
                 b=chron((1:20)+(1:20)/100),
                 d=ISOdatetime(year=rep(2003,20),month=rep(4,20),day=1:20,
                               hour=rep(11,20),min=rep(17,20),sec=rep(11,20)),
                 f=ISOdatetime(year=rep(2003,20),month=rep(4,20),day=1:20,
                               hour=1:20,min=1:20,sec=1:20),
                 g=ISOdate(year=2001:2020,month=rep(3,20),day=1:20))
 describe(d)
```

```
# Make a function to run describe, latex.describe, and use the kdvi
# previewer in Linux to view the result and easily make a pdf file

 ldesc <- function(data) {
  options(xdvicmd='kdvi')
  d <- describe(data, desc=deparse(substitute(data)))
  dvi(latex(d, file='/tmp/z.tex'), nomargins=FALSE, width=8.5, height=11)
 }

 ldesc(d)
## End(Not run)
```

`dotchart2` *Enhanced Dot Chart*

Description

`dotchart2` is an enhanced version of the `dotchart` function with several new options.

Usage

```
dotchart2(data, labels, groups=NULL, gdata=NA, horizontal=TRUE, pch=16,
          xlab='', ylab='', auxdata, auxgdata=NULL, auxtitle,
          lty=if(.R.) 1 else 2, lines=TRUE, dotsize = .8,
          cex = par("cex"), cex.labels = cex,
          cex.group.labels = cex.labels*1.25, sort.=TRUE,
              add=FALSE, dotfont=par('font'), groupfont=2,
              reset.par=add, xaxis=TRUE, width.factor=1.1,
          lcolor=if(.R.)'gray' else par('col'), ...)
```

Arguments

`data`	a numeric vector whose values are shown on the x-axis
`labels`	a vector of labels for each point, corresponding to `x`. If omitted, `names(data)` are used, and if there are no `names`, integers prefixed by `"#"` are used.
`groups`	an optional categorical variable indicating how `data` values are grouped
`gdata`	data values for groups, typically summaries such as group medians
`horizontal`	set to `FALSE` to make the chart vertical instead of the default
`pch`	default character number or value for plotting dots in dot charts. The default is 16.
`xlab`	x-axis title
`ylab`	y-axis title
`auxdata`	a vector of auxiliary data given to `dotchart2`, of the same length as the first (`data`) argument. If present, this vector of values will be printed outside the right margin of the dot chart. Usually `auxdata` represents cell sizes.

`auxgdata`	similar to `auxdata` but corresponding to the `gdata` argument. These usually represent overall sample sizes for each group of lines.
`auxtitle`	if `auxdata` is given, `auxtitle` specifies a column heading for the extra printed data in the chart, e.g., `"N"`
`lty`	line type for horizontal lines. Default is 1 for R, 2 for S-Plus
`lines`	set to `FALSE` to suppress drawing of reference lines
`dotsize`	`cex` value for drawing dots. Default is 0.8. Note that the original `dotchart` function used a default of 1.2.
`cex`	see `par`
`cex.labels`	`cex` parameter that applies only to the line labels for the dot chart `cex` parameter for major grouping labels for `dotchart2`. Defaults to `cex`.
`cex.group.labels`	value of `cex` corresponding to `gdata`
`sort.`	set to `FALSE` to keep `dotchart2` from sorting the input data, i.e., it will assume that the data are already properly arranged. This is especially useful when you are using `gdata` and `groups` and you want to control the order that groups appear on the chart (from top to bottom).
`add`	set to `TRUE` to add to an existing plot
`dotfont`	font number of plotting dots. Default is one. Use `-1` to use "outline" fonts. For example, `pch=183, dotfont=-1` plots an open circle for UNIX on postscript. `pch=1` makes an open octagon under Windows.
`groupfont`	font number to use in drawing `group` labels for `dotchart2`. Default is `2` for boldface.
`reset.par`	set to `FALSE` to cause `dotchart2` to not reset the `par` parameters when finished. This is useful when `add=TRUE` is about to be used in another call. The default is to reset the `par` parameters if `add=TRUE` and not if `add=FALSE`, i.e., the program assumes that only one set of points will be added to an existing set. If you fail to use `reset.par=TRUE` for the first of a series of plots, the next call to `plot` with `add=TRUE` will result in distorted x-axis scaling.
`xaxis`	set to `FALSE` to suppress drawing x-axis
`width.factor`	When the calculated left margin turns out to be faulty, specify a factor by which to multiple the left margin as `width.factor` to get the appropriate space for labels on horizonal charts.
`lcolor`	color for horizontal reference lines. Default is `"gray"` for R, `par("col")` for S-Plus.
`...`	arguments passed to `plot.default`

Side Effects

`dotchart` will leave `par` altered if `reset.par=FALSE`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`dotchart`

`dropUnusedLevels` *Create Temporary Factor Subsetting Function*

Description

Calling this function makes Hmisc have its pre-version 3.0 behavior in which the R `[.factor` function was overridden by a customized version that caused unused `factor levels` to be dropped when the factor variable was subscripted (subsetted). `dropUnusedLevels()` creates a temporary version of `[.factor` in the global environment, which will take precedence. To later make this function keep unused levels on subsetting, issue `options(drop.unused.levels=FALSE)` or just remove this temporary function from the environment by issuing `remove('[.factor',pos='.GlobalEnv')`.

Usage

```
dropUnusedLevels()
```

Author(s)

Frank Harrell and Jens Oehlschlaegel

See Also

`factor`, `[.factor`

Examples

```
## Not run:
x <- factor(c('a','b','c'))
x[1:2]  # keeps level c
dropUnusedLevels()
x[1:2]  # no c any more
## End(Not run)
```

eip *Edit In Place*

Description

Invokes `edit( )` on the object `name` and stores the resulting edited object in place of the original, even if this is a remote place (as long as the user has write access). This is more useful in S-Plus than in R.

Usage

```
eip(name)
```

Arguments

`name`	an object, usually a function

Value

none

Author(s)

Frank Harrell

See Also

`edit`

Examples

```
## Not run:
eip(summary.formula
## End(Not run)  # make temporary bug fix in central area
```

equalBins *Multicolumn Formating*

Description

Expands the width either supercolumns or the subcolumns so that the the sum of the supercolumn widths is the same as the sum of the subcolumn widths.

Usage

```
equalBins(widths, subwidths)
```

Arguments

`widths`	widths of the supercolumns.
`subwidths`	list of widths of the subcolumns for each supercolumn.

Details

This determins the correct subwidths of each of various columns in a table for printing. The correct width of the multicolumns is deterimed by summing the widths of it subcolumns.

Value

widths of the the columns for a table.

Author(s)

Charles Dupont

See Also

`nchar`, `stringDims`

Examples

```
mcols <- c("Group 1", "Group 2")
mwidth <- nchar(mcols, type="width")
spancols <- c(3,3)
ccols <- c("a", "deer", "ad", "cat", "help", "bob")
cwidth <- nchar(ccols, type="width")

subwidths <- partition.vector(cwidth, spancols)

equalBins(mwidth, subwidths)
```

`errbar` *Plot Error Bars*

Description

`errbar` adds vertical error bars to an existing plot or makes a new plot with error bars. It can also make a horizontal error bar plot that shows error bars for group differences as well as bars for groups. For the latter type of plot, the lower x-axis scale corresponds to group estimates and the upper scale corresponds to differences. The spacings of the two scales are identical but the scale for differences has its origin shifted so that zero may be included. If at least one of the confidence intervals includes zero, a vertical dotted reference line at zero is drawn.

Usage

```
errbar(x, y, yplus, yminus, cap, xlab, ylab, add=FALSE,
       lty=1, ylim, lwd=1, Type=rep(1,length(y)), ... )
```

Arguments

`x`	vector of numeric x values (for vertical error bars) or a factor or character variable (for horizontal error bars, `x` representing the group labels)
`y`	vector of y values.
`yplus`	vector of y values: the tops of the error bars.
`yminus`	vector of y values: the bottoms of the error bars.
`cap`	The width of the little lines at the tops and bottoms of the error bars in units of the width of the plot. Default is .015.
`xlab`	
`ylab`	optional axis labels if `add=FALSE`. Defaults to blank for horizontal charts.
`add`	Set toTRUE to add bars to an existing plot (available only for vertical error bars)
`lty`	Line type for bars
`ylim`	Y-axis limits. Default is to use range of yminus and yplus. For horizonal charts, `ylim` is really the `x`-axis range, excluding differences.
`lwd`	Line width for line segments (not main line)
`Type`	used for horizontal bars only. Is an integer vector with values `1` if corresponding values represent simple estimates, `2` if they represent differences.
`...`	other parameters passed to all graphics functions except `axis`.

Author(s)

Charles Geyer, University of Chicago. Modified by Frank Harrell, Vanderbilt University, to handle missing data, to add the parameters `add` and `lty`, and to implement horizontal charts with differences.

Examples

```
set.seed(1)
x <- 1:10
y <- x + rnorm(10)
delta <- runif(10)
errbar( x, y, y + delta, y - delta )

# Show bootstrap nonparametric CLs for 3 group means and for
# pairwise differences on same graph
group <- sample(c('a','b','d'), 200, TRUE)
y     <- runif(200) + .25*(group=='b') + .5*(group=='d')
cla <- smean.cl.boot(y[group=='a'],B=100,reps=TRUE)  # usually B=1000
a   <- attr(cla,'reps')
clb <- smean.cl.boot(y[group=='b'],B=100,reps=TRUE)
b   <- attr(clb,'reps')
cld <- smean.cl.boot(y[group=='d'],B=100,reps=TRUE)
d   <- attr(cld,'reps')
a.b <- quantile(a-b,c(.025,.975))
a.d <- quantile(a-d,c(.025,.975))
b.d <- quantile(b-d,c(.025,.975))
```

```
errbar(c('a','b','d','a - b','a - d','b - d'),
       c(cla[1],clb[1],cld[1],cla[1]-clb[1],cla[1]-cld[1],clb[1]-cld[1]),
       c(cla[3],clb[3],cld[3],a.b[2],a.d[2],b.d[2]),
       c(cla[2],clb[2],cld[2],a.b[1],a.d[1],b.d[1]),
       Type=c(1,1,1,2,2,2))


rm(x,y,delta,group,a,b,d,a.b,a.d,b.d,cla,clb,cld)
```

`escapeRegex` *Escapes any characters that would have special meaning in a reqular expression.*

Description

Escapes any characters that would have special meaning in a reqular expression.

Usage

```
escapeRegex(string)
escapeBS(string)
```

Arguments

`string`	string being operated on.

Details

`escapeRegex` will escape any characters that would have special meaning in a reqular expression. For any string `grep(regexpEscape(string), string)` will always be true.

`escapeBS` will escape any backslash `\backslash` in a string.

Value

The value of the string with any characters that would have special meaning in a reqular expression escaped.

Author(s)

Charles Dupont
Department of Biostatistics
Vanderbilt University

See Also

`grep`

Examples

```
string <- "this\\(system) {is} [full]."
escapeRegex(string)

escapeBS(string)
```

---

event.chart *Flexible Event Chart for Time-to-Event Data*

Description

Creates an event chart on the current graphics device. Also, allows user to plot legend on plot area or on separate page. Contains features useful for plotting data with time-to-event outcomes Which arise in a variety of studies including randomized clinical trials and non-randomized cohort studies. This function can use as input a matrix or a data frame, although greater utility and ease of use will be seen with a data frame.

Usage

```
event.chart(data, subset.r = 1:dim(data)[1], subset.c = 1:dim(data)[2],
        sort.by = NA, sort.ascending =TRUE,
        sort.na.last =TRUE, sort.after.subset =TRUE,
        y.var = NA, y.var.type = 'n',
        y.jitter =FALSE, y.jitter.factor = 1,
        y.renum =FALSE, NA.rm =FALSE, x.reference = NA,
        now = max(data[,subset.c], na.rm =TRUE),
        now.line =FALSE, now.line.lty = 2,
        now.line.lwd = 1, now.line.col = 1, pty='m',
        date.orig = c(1,1,1960), titl = 'Event Chart',

        y.idlabels = NA, y.axis = 'auto',
        y.axis.custom.at = NA, y.axis.custom.labels = NA,
        y.julian =FALSE, y.lim.extend = c(0,0),
        y.lab = ifelse(is.na(y.idlabels), '' , as.character(y.idlabels)),

        x.axis.all =TRUE, x.axis = 'auto',
        x.axis.custom.at = NA, x.axis.custom.labels = NA,
        x.julian =FALSE, x.lim.extend = c(0,0), x.scale = 1,
        x.lab = ifelse(x.julian, 'Follow-up Time', 'Study Date'),

        line.by = NA, line.lty = 1, line.lwd = 1, line.col = 1,
        line.add = NA, line.add.lty = NA,
        line.add.lwd = NA, line.add.col = NA,
        point.pch = 1:length(subset.c),
        point.cex = rep(0.6,length(subset.c)),
        point.col = rep(1,length(subset.c)),

        legend.plot =FALSE, legend.location = 'o', legend.titl = titl,
        legend.titl.cex = 3.0, legend.titl.line = 1.0,
        legend.point.at = list(x = c(5,95), y = c(95,30)),
        legend.point.pch = point.pch,
        legend.point.text = ifelse(rep(is.data.frame(data),
              length(subset.c)), names(data[,subset.c]), subset.c),
        legend.cex = 2.5, legend.bty = 'n',
        legend.line.at = list(x = c(5,95), y = c(20,5)),
        legend.line.text = names(table(as.character(data[,line.by]),
              exclude = c('','NA'))),
        legend.line.lwd = line.lwd, legend.loc.num = 1,

        ...)

event.convert(data2, event.time = 1, event.code = 2)
```

Arguments

`data`	a matrix or data frame with rows corresponding to subjects and columns corresponding to variables. Note that for a data frame or matrix containing multiple time-to-event data (e.g., time to recurrence, time to death, and time to last follow-up), one column is required for each specific event.
`data2`	a matrix or dataframe with at least 2 columns; by default, the first column contains the event time and the second column contains the k event codes (e.g. 1=dead, 0=censord)
`subset.r`	subset of rows of original matrix or data frame to place in event chart. Logical arguments may be used here (e.g., treatment.arm == 'a', if the data frame, data, has been attached to the search directory; otherwise, `data$treatment.arm == "a"`).
`subset.c`	subset of columns of original matrix or data frame to place in event chart; if working with a data frame, a vector of data frame variable names may be used for subsetting purposes (e.g., c('randdate', 'event1').
`sort.by`	column(s) or data frame variable name(s) with which to sort the chart's output. The default is NA, thereby resulting in a chart sorted by original row number.
`sort.ascending`	logical flag (which takes effect only if the argument sort.by is utilized). If TRUE (default), sorting is done in ascending order; if F, descending order.
`sort.na.last`	logical flag (which takes effect only if the argument sort.by is utilized). If T (default), NA values are considered as last values in ordering.
`sort.after.subset`	logical flag (which takes effect only if the argument sort.by is utilized). If F, sorting data (via sort.by specified variables or columns) will be performed prior to row subsetting (via subset.r); if T (default), row subsetting of original data will be done before sorting.

`y.var` variable name or column number of original matrix or data frame with which to scale y-axis. Default is NA, which will result in equally spaced lines on y-axis (based on original data or sorted data if requested by sort.by). Otherwise, location of lines on y-axis will be dictated by specified variable or column. Examples of specified variables may be date of an event or a physiological covariate. Any observation which has a missing value for the y.var variable will not appear on the graph.

`y.var.type` type of variable specified in y.var (which will only take effect if argument y.var is utilized). If 'd', specifed variable is a date (either numeric julian date or an S-Plus dates object); if 'n', specifed variable is numeric (e.g., systolic blood pressure level) although not a julian date.

`y.jitter` logical flag (which takes effect only if the argument y.var is utilized). Due to potential ties in y.var variable, y.jitter (when T) will jitter the data to allow discrimination between observations at the possible cost of producing slightly inaccurate dates or covariate values; if F (the default), no jittering will be performed. The y.jitter algorithm assumes a uniform distribution of observations across the range of y.var. The algorithm is as follows:

size.jitter <- (diff(range(y.var)) / (2 * (length(y.var) - 1))) * y.jitter.factor .

The default of y.jitter.factor is 1. The entire product is then used as an argument into runif: y.var <- y.var + runif(length(y.var), -size.jitter, size.jitter) .

`y.jitter.factor`
an argument used with the y.jitter function to scale the range of added noise. Default is 1.

`y.renum` logical flag. If T, subset observations are listed on y-axis from 1 to length(subset.r); if F (default), subset observations are listed on y-axis in original form. As an example, if subset.r = 301:340 and y.renum ==TRUE, y-axis will be shown as 1 through 40. However, if y.renum ==FALSE, y-axis will be shown as 301 through 340. The above examples assume the following argument, NA.rm, is set to F.

`NA.rm` logical flag. If T, subset observations which have NA for each variable specified in subset.c will not have an entry on the y-axis. Also, if the following argument, x.reference, is specified, observations with missing x.reference values will also not have an entry on the y-axis. If F (default), user can identify those observations which do have NA for every variable specified in subset.c (or, if x.reference is specified, also those observations which are missing only the x.reference value); this can easily be done by examining the resulting y-axis and recognizing the observations without any plotting symbols.

`x.reference` column of original matrix or data frame with which to reference the x-axis. That is, if specified, all columns specified in subset.c will be substracted by x.reference. An example may be to see the timing of events before and after treatment or to see time-to-event after entry into study. The event times will be aligned using the x.reference argument as the reference point.

`now` the 'now' date which will be used for top of y-axis when creating the Goldman eventchart (see reference below). Default is max(data[, subset.c], na.rm =TRUE).

`now.line` logical flag. A feature utilized by the Goldman Eventchart. When x.reference is specified as the start of follow-up and y.var = x.reference, then the Goldman chart can be created. This argument, if T, will cause the plot region to be square, and will draw a line with a slope of -1 from the top of the y-axis to the right end of the x-axis. Essentially, it denotes end of current follow-up period for looking at the time-to-event data. Default is F.

`now.line.lty`	line type of now.line.
`now.line.lwd`	line width of now.line.
`now.line.col`	color of now.line.
`pty`	graph option, pty='m' is the default; use pty='s' for the square looking Goldman's event chart.
`date.orig`	date of origin to consider if dates are in julian, SAS , or S-Plus dates object format; default is January 1, 1960 (which is the default origin used by both S-Plus and SAS). Utilized when either y.julian =FALSE or x.julian = F.
`titl`	title for event chart. Default is 'Event Chart'.
`y.idlabels`	column or data frame variable name used for y-axis labels. For example, if c('pt.no') is specified, patient ID (stored in 'pt.no') will be seen on y-axis labels instead of sequence specified by subset.r. This argument takes precedence over both y.axis='auto' and y.axis='custom' (see below). NOTE: Program will issue warning if this argument is specified and if is.na(y.var) == F; y.idlabels will not be used in this situation. Also, attempting to plot too many patients on a single event chart will cause undesirable plotting of y.idlabels.
`y.axis`	character string specifying whether program will control labelling of y-axis (with argument 'auto'), or if user will control labelling (with argument 'custom'). If 'custom' is chosen, user must specify location and text of labels using y.axis.custom.at and y.axis.custom.labels arguments, respectively, listed below. This argument will not be utilized if y.idlabels is specified.
`y.axis.custom.at`	user-specified vector of y-axis label locations. Must be used when y.axis = 'custom'; will not be used otherwise.
`y.axis.custom.labels`	user-specified vector of y-axis labels. Must be used when y.axis = 'custom'; will not be used otherwise.
`y.julian`	logical flag (which will only be considered if y.axis == 'auto' and (!is.na(y.var) & y.var.type== 'd'). If F (default), will convert julian numeric dates or S-Plus dates objects into 'mm/dd/yy' format for the y-axis labels. If T, dates will be printed in julian (numeric) format.
`y.lim.extend`	two-dimensional vector representing the number of units that the user wants to increase ylim on bottom and top of y-axis, respectively. Default = c(0,0). This argument will not take effect if the Goldman chart is utilized.
`y.lab`	single label to be used for entire y-axis. Default will be the variable name or column number of y.idlabels (if non-missing) and blank otherwise.
`x.axis.all`	logical flag. If T (default), lower and upper limits of x-axis will be based on all observations (rows) in matrix or data frame. If F, lower and upper limits will be based only on those observations specified by subset.r (either before or after sorting depending on specification of sort.by and value of sort.after.subset).

`x.axis`	character string specifying whether program will control labelling of x-axis (with argument 'auto'), or if user will control labelling (with argument 'custom'). If 'custom' is chosen, user must specify location and text of labels using x.axis.custom.at and x.axis.custom.labels arguments, respectively, listed below.
`x.axis.custom.at`	user-specified vector of x-axis label locations. Must be used when x.axis == 'custom'; will not be used otherwise.
`x.axis.custom.labels`	user-specified vector of x-axis labels. Must be used when x.axis == 'custom'; will not be used otherwise.
`x.julian`	logical flag (which will only be considered if x.axis == 'auto'). If F (default), will convert julian dates or S-plus dates objects into 'mm/dd/yy' format for the x-axis labels. If T, dates will be printed in julian (numeric) format. NOTE: This argument should remain T if x.reference is specified.
`x.lim.extend`	two-dimensional vector representing the number of time units (usually in days) that the user wants to increase xlim on left-hand side and right-hand side of x-axis, respectively. Default = c(0,0). This argument will not take effect if the Goldman chart is utilized.
`x.scale`	a factor whose reciprocal is multiplied to original units of the x-axis. For example, if the original data frame is in units of days, x.scale = 365 will result in units of years (notwithstanding leap years). Default is 1.
`x.lab`	single label to be used for entire x-axis. Default will be 'On Study Date' if x.julian ==FALSE and 'Time on Study' if x.julian = T.
`line.by`	column or data frame variable name for plotting unique lines by unique values of vector (e.g., specify c('arm') to plot unique lines by treatment arm). Can take at most one column or variable name. Default is NA which produces identical lines for each patient.
`line.lty`	vector of line types corresponding to ascending order of line.by values. If line.by is specified, the vector should be the length of the number of unique values of line.by. If line.by is NA, only line.lty[1] will be used. The default is 1.
`line.lwd`	vector of line widths corresponding to ascending order of line.by values. If line.by is specified, the vector should be the length of the number of unique values of line.by. If line.by is NA, only line.lwd[1] will be used. The default is 1.
`line.col`	vector of line colors corresponding to ascending order of line.by values. If line.by is specified, the vector should be the length of the number of unique values of line.by. If line.by is NA, only line.col[1] will be used. The default is 1.
`line.add`	a 2xk matrix with k=number of pairs of additional line segments to add. For example, if it is of interest to draw additional line segments connecting events one and two, two and three, and four and five, (possibly with different colors), an appropriate line.add argument would be matrix(c('first.event','second.event','second.event','third.event', 'fourth.event','fifth.event'), 2, 3). One line segment would be drawn between first.event and second.event, a second line segment would be drawn between second.event and third.event, and a third line segment would be drawn between

fourth.event and fifth.event. Different line types, widths and colors can be specified (in arguments listed just below).

The convention use of subset.c and line.add must match (i.e., column name must be used for both or column number must be used for both).

If line.add != NA, length of line.add.lty, line.add.lwd, and line.add.col must be the same as number of pairs of additional line segments to add.

NOTE: The drawing of the original default line may be suppressed (with line.col = 0), and line.add can be used to do all the line plotting for the event chart.

`line.add.lty`	a kx1 vector corresponding to the columns of line.add; specifies the line types for the k line segments.
`line.add.lwd`	a kx1 vector corresponding to the columns of line.add; specifies the line widths for the k line segments.
`line.add.col`	a kx1 vector corresponding to the columns of line.add; specifies the line colors for the k line segments.
`point.pch`	vector of pch values for points representing each event. If similar events are listed in multiple columns (e.g., regular visits or a recurrent event), repeated pch values may be listed in the vector (e.g., c(2,4,rep(183,3))). If length(point.pch) < length(subset.c), point.pch will be repeated until lengths are equal; a warning message will verify this condition.
`point.cex`	vector of size of points representing each event. If length(point.cex) < length(subset.c), point.cex will be repeated until lengths are equal; a warning message will verify this condition.
`point.col`	vector of colors of points representing each event. If length(point.col) < length(subset.c), point.col will be repeated until lengths are equal; a warning message will verify this condition.
`legend.plot`	logical flag; if T, a legend will be plotted. Location of legend will be based on specification of legend.location along with values of other arguments listed below. Default is F (i.e., no legend plotting).
`legend.location`	will be used only if legend.plot=T. If 'o' (default), a one-page legend will precede the output of the chart. The user will need to hit <enter> in order for the event chart to be displayed. This feature is possible due to the **dev.ask** option. If 'i', an internal legend will be placed in the plot region based on legend.point.at. If 'l', a legend will be placed in the plot region using the locator option. Legend will map points to events (via column names, by default) and, if line.by is specified, lines to groups (based on levels of line.by).
`legend.titl`	title for the legend; default is title to be used for main plot. Only used when legend.location = 'o'.
`legend.titl.cex`	size of text for legend title. Only used when legend.location = 'o'.
`legend.titl.line`	line location of legend title dictated by mtext function with outer=FALSE option; default is 1.0. Only used when legend.location = 'o'.
`legend.point.at`	location of upper left and lower right corners of legend area to be utilized for describing events via points and text.

`legend.point.pch`	vector of pch values for points representing each event in the legend. Default is point.pch.
`legend.point.text`	text to be used for describing events; the default is setup for a data frame, as it will print the names of the columns specified by subset.c .
`legend.cex`	size of text for points and event descriptions. Default is 2.5 which is setup for legend.location = 'o'. A much smaller cex is recommended (possibly 0.75) for use with legend.location = 'i' or legend.location = 'l'.
`legend.bty`	option to put a box around the legend(s); default is to have no box (legend.bty = 'n'). Option legend.bty = 'o' will produce a legend box.
`legend.line.at`	if line.by was specified (with legend.location = 'o' or legend.location = 'i'), this argument will dictate the location of the upper left and lower right corners of legend area to be utilized for describing the different line.by values (e.g., treatment.arm). The default is setup for legend.location == 'o'.
`legend.line.text`	text to be used for describing line.by values; the default are the names of the unique non-missing line.by values as produced from the table function.
`legend.line.lwd`	vector of line widths corresponding to line.by values.
`legend.loc.num`	number used for locator argument when legend.locator = 'l'. If 1 (default), user is to locate only the top left corner of the legend box. If 2, user is to locate both the top left corner and the lower right corner. This will be done twice when line.by is specified (once for points and once for lines).
`event.time`	the column number in data contains the event time
`event.code`	the column number in data contains the event code
`...`	additional par arguments for use in main plot.

Details

if you want to put, say, two eventcharts side-by-side, in a plot region, you should not set up par(mfrow=c(1,2)) before running the first plot. Instead, you should add the argument mfg=c(1,1,1,2) to the first plot call followed by the argument mfg=c(1,2,1,2) to the second plot call.

if dates in original data frame are in a specialized form (eg., mm/dd/yy) of mode CHARACTER, the user must convert those columns to become class dates or julian numeric mode (see ?dates for more information). For example, in a data frame called testdata, with specialized dates in columns 4 thru 10, the following code could be used: as.numeric(dates(testdata[,4:10]). This will convert the columns to numeric julian dates based on the function's default origin of January 1, 1960. If original dates are in class dates or julian form, no extra work is necessary.

In the survival analysis, the data typically come in two columns: one column containing survival time and the other containing censoring indicator or event code. The event.convert function converts this type of data into multiple columns of event times, one column of each event type, suitable for the event.chart function.

Side Effects

an event chart is created on the current graphics device. If legend.plot =TRUE and legend.location = 'o', a one-page legend will precede the event chart. Please note that par parameters on completion of function will be reset to par parameters existing prior to start of function.

Author(s)

J. Jack Lee and Kenneth R. Hess
Department of Biostatistics
University of Texas
M.D. Anderson Cancer Center
Houston, TX 77030
jjlee@mdanderson.org, khess@mdanderson.org

Joel A. Dubin
Division of Biostatistics
Department of Epidemiology and Public Health
Yale University
joel.dubin@yale.edu

References

Lee J.J., Hess, K.R., Dubin, J.A. (2000). Extensions and applications of event charts. *The American Statistician,* **54:1**, 63–70.

Dubin, J.A., Lee, J.J., Hess, K.R. (1997). The Utility of Event Charts. *Proceedings of the Biometrics Section, American* Statistical Association.

Dubin, J.A., Muller H-G, Wang J-L (2001). Event history graphs for censored survival data. *Statistics in Medicine,* **20:** 2951–2964.

Goldman, A.I. (1992). EVENTCHARTS: Visualizing Survival and Other Timed-Events Data. *The American Statistician,* **46:1**, 13–18.

See Also

`event.history`

Examples

```
# The sample data set is an augmented CDC AIDS dataset (ASCII)
# which is used in the examples in the help file.  This dataset is
# described in Kalbfleisch and Lawless (JASA, 1989).
# Here, we have included only children 4 years old and younger.
# We have also added a new field, dethdate, which
# represents a fictitious death date for each patient.  There was
# no recording of death date on the original dataset.
#
# All dates are julian with julian=0 being
# January 1, 1960, and julian=14000 being 14000 days beyond
# January 1, 1960 (i.e., May 1, 1998).

cdcaids <- data.frame(
age=c(4,2,1,1,2,2,2,4,2,1,1,3,2,1,3,2,1,2,4,2,2,1,4,2,4,1,4,2,1,1,3,3,1,3),
infedate=c(
7274,7727,7949,8037,7765,8096,8186,7520,8522,8609,8524,8213,8455,8739,
8034,8646,8886,8549,8068,8682,8612,9007,8461,8888,8096,9192,9107,9001,
9344,9155,8800,8519,9282,8673),
diagdate=c(
8100,8158,8251,8343,8463,8489,8554,8644,8713,8733,8854,8855,8863,8983,
9035,9037,9132,9164,9186,9221,9224,9252,9274,9404,9405,9433,9434,9470,
9470,9472,9489,9500,9585,9649),
diffdate=c(
826,431,302,306,698,393,368,1124,191,124,330,642,408,244,1001,391,246,
615,1118,539,612,245,813,516,1309,241,327,469,126,317,689,981,303,976),
dethdate=c(
8434,8304,NA,8414,8715,NA,8667,9142,8731,8750,8963,9120,9005,9028,9445,
9180,9189,9406,9711,9453,9465,9289,9640,9608,10010,9488,9523,9633,9667,
9547,9755,NA,9686,10084),
censdate=c(
NA,NA,8321,NA,NA,8519,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,
NA,NA,NA,NA,NA,NA,NA,NA,NA,10095,NA,NA))

cdcaids <- upData(cdcaids,
 labels=c(age     ='Age, y', infedate='Date of blood transfusion',
          diagdate='Date of AIDS diagnosis',
          diffdate='Incubation period (days from HIV to AIDS)',
          dethdate='Fictitious date of death',
          censdate='Fictitious censoring date'))

# Note that the style options listed with these
# examples are best suited for output to a postscript file (i.e., using
# the postscript function with horizontal=TRUE) as opposed to a graphical
# window (e.g., motif).

# To produce simple calendar event chart (with internal legend):
# postscript('example1.ps', horizontal=TRUE)
 event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'observation dates',
  y.lab='patients (sorted by AIDS diagnosis date)',
  titl='AIDS data calendar event chart 1',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  legend.point.at = list(c(7210, 8100), c(35, 27)), legend.bty='o')

# To produce simple interval event chart (with internal legend):
# postscript('example2.ps', horizontal=TRUE)
 event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since transfusion (in days)',
  y.lab='patients (sorted by AIDS diagnosis date)',
  titl='AIDS data interval event chart 1',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i', legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='infedate', x.julian=TRUE,
  legend.bty='o', legend.point.at = list(c(1400, 1950), c(7, -1)))

# To produce more complicated interval chart which is
# referenced by infection date, and sorted by age and incubation period:
# postscript('example3.ps', horizontal=TRUE)
 event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since diagnosis of AIDS (in days)',
  y.lab='patients (sorted by age and incubation length)',
  titl='AIDS data interval event chart 2 (sorted by age, incubation)',
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8),
  legend.plot=TRUE, legend.location='i',legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='diagdate', x.julian=TRUE, sort.by=c('age','diffdate'),
  line.by='age', line.lty=c(1,3,2,4), line.lwd=rep(1,4), line.col=rep(1,4),
  legend.bty='o', legend.point.at = list(c(-1350, -800), c(7, -1)),
  legend.line.at = list(c(-1350, -800), c(16, 8)),
  legend.line.text=c('age = 1', '       = 2', '       = 3', '       = 4'))

# To produce the Goldman chart:
# postscript('example4.ps', horizontal=TRUE)
 event.chart(cdcaids,
  subset.c=c('infedate','diagdate','dethdate','censdate'),
  x.lab = 'time since transfusion (in days)', y.lab='dates of observation',
  titl='AIDS data Goldman event chart 1',
  y.var = c('infedate'), y.var.type='d', now.line=TRUE, y.jitter=FALSE,
  point.pch=c(1,2,15,0), point.cex=c(1,1,0.8,0.8), mgp = c(3.1,1.6,0),
  legend.plot=TRUE, legend.location='i',legend.cex=1.0,
  legend.point.text=c('transfusion','AIDS diagnosis','death','censored'),
  x.reference='infedate', x.julian=TRUE,
  legend.bty='o', legend.point.at = list(c(1500, 2800), c(9300, 10000)))

# To convert coded time-to-event data, then, draw an event chart:
surv.time <- c(5,6,3,1,2)
cens.ind   <- c(1,0,1,1,0)
surv.data  <- cbind(surv.time,cens.ind)
event.data <- event.convert(surv.data)
event.chart(cbind(rep(0,5),event.data),x.julian=TRUE,x.reference=1)
```

`event.history` *Produces event.history graph for survival data*

Description

Produces an event history graph for right-censored survival data, including time-dependent covariate status, as described in Dubin, Muller, and Wang (2001). Effectively, a Kaplan-Meier curve is produced with supplementary information regarding individual survival information, censoring information, and status over time of an individual time-dependent covariate or time-dependent covariate function for both uncensored and censored individuals.

Usage

```
event.history(data, survtime.col, surv.col,
              surv.ind = c(1, 0), subset.rows = NULL,
              covtime.cols = NULL, cov.cols = NULL,
              num.colors = 1, cut.cov = NULL, colors = 1,
              cens.density = 10, mult.end.cens = 1.05,
              cens.mark.right =FALSE, cens.mark = "-",
              cens.mark.ahead = 0.5, cens.mark.cutoff = -1e-08,
              cens.mark.cex = 1,
              x.lab = "time under observation",
              y.lab = "estimated survival probability",
              title = "event history graph", ...)
```

Arguments

`data`	A matrix or data frame with rows corresponding to units (often individuals) and columns corresponding to survival time, event/censoring indicator. Also, multiple columns may be devoted to time-dependent covariate level and time change.
`survtime.col`	Column (in data) representing minimum of time-to-event or right-censoring time for individual.
`surv.col`	Column (in data) representing event indicator for an individual. Though, traditionally, such an indicator will be 1 for an event and 0 for a censored observation, this indicator can be represented by any two numbers, made explicit by the surv.ind argument.
`surv.ind`	Two-element vector representing, respectively, the number for an event, as listed in surv.col, followed by the number for a censored observation. Default is traditional survival data represention, i.e., c(1,0).
`subset.rows`	Subset of rows of original matrix or data frame (data) to place in event history graph. Logical arguments may be used here (e.g., treatment.arm == 'a', if the data frame, data, has been attached to the search directory;
`covtime.cols`	Column(s) (in data) representing the time when change of time-dependent covariate (or time-dependent covariate function) occurs. There should be a unique non-NA entry in the column for each such change (along with corresponding cov.cols column entry representing the value of the covariate or function at that change time). Default is NULL, meaning no time-dependent covariate information will be presented in the graph.
`cov.cols`	Column(s) (in data) representing the level of the time-dependent covariate (or time-dependent covariate function). There should be a unique non-NA column entry representing each change in the level (along with a corresponding covtime.cols column entry representing the time of the change). Default is NULL, meaning no time-dependent covariate information will be presented in the graph.
`num.colors`	Colors are utilized for the time-dependent covariate level for an individual. This argument provides the number of unique covariate levels which will be displayed by mapping the number of colors (via num.colors) to the number of desired covariate levels. This will divide the covariate span into roughly equally-sized intervals, via the S-Plus cut function. Default is one color, meaning no

time-dependent information will be presented in the graph. Note that this argument will be ignored/superceded if a non-NULL argument is provided for the cut.cov parameter.

`cut.cov` This argument allows the user to explicitly state how to define the intervals for the time-dependent covariate, such that different colors will be allocated to the user-defined covariate levels. For example, for plotting five colors, six ordered points within the span of the data's covariate levels should be provided. Default is NULL, meaning that the num.colors argument value will dictate the number of breakpoints, with the covariate span defined into roughly equally-sized intervals via the S-Plus cut function. However, if is.null(cut.cov) ==FALSE, then this argument supercedes any entry for the num.colors argument.

`colors` This is a vector argument defining the actual colors used for the time-dependent covariate levels in the plot, with the index of this vector corresponding to the ordered levels of the covariate. The number of colors (i.e., the length of the colors vector) should correspond to the value provided to the num.colors argument or the number of ordered points - 1 as defined in the cut.cov argument (with cut.cov superceding num.colors if is.null(cut.cov) ==FALSE). The function, as currently written, allows for as much as twenty distinct colors. This argument effectively feeds into the col argument for the S-Plus polygon function. Default is colors=1. See the col argument for the both the S-Plus par function and polygon function for more information.

`cens.density` This will provide the shading density at the end of the individual bars for those who are censored. For more information on shading density, see the density argument in the S-Plus polygon function. Default is cens.density=10.

`mult.end.cens`
This is a multiplier that extends the length of the longest surviving individual bar (or bars, if a tie exists) if right-censored, presuming that no event times eventually follow this final censored time. Default extends the length 5 percent beyond the length of the observed right-censored survival time.

`cens.mark.right`
A logical argument that states whether an explicit mark should be placed to the right of the individual right-censored survival bars. This argument is most useful for large sample sizes, where it may be hard to detect the special shading via cens.density, particularly for the short-term survivors.

`cens.mark` Character argument which describes the censored mark that should be used if cens.mark.right = T. Default is '-'.

`cens.mark.ahead`
A numeric argument, which specifies the absolute distance to be placed between the individual right-censored survival bars and the mark as defined in the above cens.mark argument. Default is .5 (that is, a half of day, if survival time is measured in days), but may very well need adjusting depending on the maximum survival time observed in the dataset.

`cens.mark.cutoff`
A negative number very close to 0 (by default cens.mark.cutoff = -1e-8) to ensure that the censoring marks get plotted correctly. See event.history code in order to see its usage. This argument typically will not need adjustment.

`cens.mark.cex`	Numeric argument defining the size of the mark defined in the cens.mark argument above. See more information by viewing the cex argument for the S-Plus par function. Default is cens.mark.cex=1.0.
`x.lab`	Single label to be used for entire x-axis. Default is 'time under observation'.
`y.lab`	Single label to be used for entire y-axis. Default is 'estimated survival probability'.
`title`	Title for the event history graph. Default is 'event history graph'.
`...`	This allows arguments to the plot function call within the event.history function. So, for example, the axes representations can be manipulated with appropriate arguments, or particular areas of the event.history graph can be "zoomed". See the details section for more comments about zooming.

Details

In order to focus on a particular area of the event history graph, zooming can be performed. This is best done by specifying appropriate xlim and ylim arguments at the end of the event.history function call, taking advantage of the ... argument link to the plot function. An example of zooming can be seen in Plate 4 of the paper referenced below.

Please read the reference below to understand how the individual covariate and survival information is provided in the plot, how ties are handled, how right-censoring is handled, etc.

WARNING

This function has been tested thoroughly, but only within a restricted version and environment, i.e., only within S-Plus 2000, Version 3, and within S-Plus 6.0, version 2, both on a Windows 2000 machine. Hence, we cannot currently vouch for the function's effectiveness in other versions of S-Plus (e.g., S-Plus 3.4) nor in other operating environments (e.g., Windows 95, Linux or Unix). The function has also been verified to work on R under Linux.

Note

The authors have found better control of the use of color by producing the graphs via the postscript plotting device in S-Plus. In fact, the provided examples utilize the postscript function. However, your past experiences may be different, and you may prefer to control color directly (to the graphsheet in Windows environment, for example). The event.history function will work with either approach.

Author(s)

Joel Dubin
joel.dubin@yale.edu

References

Dubin, J.A., Muller, H.-G., and Wang, J.-L. (2001). Event history graphs for censored survival data. *Statistics in Medicine*, **20**, 2951-2964.

See Also

plot,polygon, event.chart

Examples

```
# Code to produce event history graphs for SIM paper
#
# before generating plots, some pre-processing needs to be performed,
#  in order to get dataset in proper form for event.history function;
#  need to create one line per subject and sort by time under observation,
#  with those experiencing event coming before those tied with censoring time;
require('survival')
data(heart)

# creation of event.history version of heart dataset (call heart.one):

heart.one <- matrix(nrow=length(unique(heart$id)), ncol=8)
for(i in 1:length(unique(heart$id)))
 {
  if(length(heart$id[heart$id==i]) == 1)
   heart.one[i,] <- as.numeric(unlist(heart[heart$id==i, ]))
  else if(length(heart$id[heart$id==i]) == 2)
   heart.one[i,] <- as.numeric(unlist(heart[heart$id==i,][2,]))
 }

heart.one[,3][heart.one[,3] == 0] <- 2  ## converting censored events to 2, from 0
if(is.factor(heart$transplant))
 heart.one[,7] <- heart.one[,7] - 1
 ## getting back to correct transplantation coding
heart.one <- as.data.frame(heart.one[order(unlist(heart.one[,2]), unlist(heart.one[,3])),])
names(heart.one) <- names(heart)
# back to usual censoring indicator:
heart.one[,3][heart.one[,3] == 2] <- 0
# note: transplant says 0 (for no transplants) or 1 (for one transplant)
#        and event = 1 is death, while event = 0 is censored

# plot single Kaplan-Meier curve from heart data, first creating survival object
heart.surv <- survfit(Surv(heart.one$stop, heart.one$event), conf.int = FALSE)

# figure 3: traditional Kaplan-Meier curve
# postscript('ehgfig3.ps', horiz=TRUE)
# omi <- par(omi=c(0,1.25,0.5,1.25))
 plot(heart.surv, ylab='estimated survival probability',
      xlab='observation time (in days)')
 title('Figure 3: Kaplan-Meier curve for Stanford data', cex=0.8)
# dev.off()

## now, draw event history graph for Stanford heart data; use as Figure 4

# postscript('ehgfig4.ps', horiz=TRUE, colors = seq(0, 1, len=20))
# par(omi=c(0,1.25,0.5,1.25))
 event.history(heart.one,
                survtime.col=heart.one[,2], surv.col=heart.one[,3],
                covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
                cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
                num.colors=2, colors=c(6,10),
                x.lab = 'time under observation (in days)',
                title='Figure 4: Event history graph for\nStanford data',
                cens.mark.right =TRUE, cens.mark = '-',
                cens.mark.ahead = 30.0, cens.mark.cex = 0.85)
# dev.off()


# now, draw age-stratified event history graph for Stanford heart data;
#  use as Figure 5

# two plots, stratified by age status
# postscript('c:\temp\ehgfig5.ps', horiz=TRUE, colors = seq(0, 1, len=20))
# par(omi=c(0,1.25,0.5,1.25))
 par(mfrow=c(1,2))

 event.history(data=heart.one, subset.rows = (heart.one[,4] < 0),
                survtime.col=heart.one[,2], surv.col=heart.one[,3],
                covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
                cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
                num.colors=2, colors=c(6,10),
                x.lab = 'time under observation\n(in days)',
                title = 'Figure 5a:\nStanford data\n(age < 48)',
                cens.mark.right =TRUE, cens.mark = '-',
                cens.mark.ahead = 40.0, cens.mark.cex = 0.85,
                xlim=c(0,1900))

 event.history(data=heart.one, subset.rows = (heart.one[,4] >= 0),
                survtime.col=heart.one[,2], surv.col=heart.one[,3],
                covtime.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,1]),
                cov.cols = cbind(rep(0, dim(heart.one)[1]), heart.one[,7]),
                num.colors=2, colors=c(6,10),
                x.lab = 'time under observation\n(in days)',
                title = 'Figure 5b:\nStanford data\n(age >= 48)',
                cens.mark.right =TRUE, cens.mark = '-',
                cens.mark.ahead = 40.0, cens.mark.cex = 0.85,
                xlim=c(0,1900))
# dev.off()
# par(omi=omi)

# we will not show liver cirrhosis data manipulation, as it was
#  a bit detailed; however, here is the
#  event.history code to produce Figure 7 / Plate 1

# Figure 7 / Plate 1 : prothrombin ehg with color
## Not run:
second.arg <- 1                          ### second.arg is for shading
third.arg <- c(rep(1,18),0,1)            ### third.arg is for intensity

# postscript('c:\temp\ehgfig7.ps', horiz=TRUE,
# colors = cbind(seq(0, 1, len = 20), second.arg, third.arg))
# par(omi=c(0,1.25,0.5,1.25), col=19)
 event.history(cirrhos2.eh, subset.rows = NULL,
               survtime.col=cirrhos2.eh$time, surv.col=cirrhos2.eh$event,
                covtime.cols = as.matrix(cirrhos2.eh[, ((2:18)*2)]),
                cov.cols = as.matrix(cirrhos2.eh[, ((2:18)*2) + 1]),
                cut.cov =  as.numeric(quantile(as.matrix(cirrhos2.eh[, ((2:18)*2) + 1]),
                                c(0,.2,.4,.6,.8,1), na.rm=TRUE) + c(-1,0,0,0,0,1)),
                colors=c(20,4,8,11,14),
                x.lab = 'time under observation (in days)',
                title='Figure 7: Event history graph for liver cirrhosis data (color)',
                cens.mark.right =TRUE, cens.mark = '-',
                cens.mark.ahead = 100.0, cens.mark.cex = 0.85)
# dev.off()
## End(Not run)
```

`find.matches` *Find Close Matches*

Description

Compares each row in `x` against all the rows in `y`, finding rows in `y` with all columns within a tolerance of the values a given row of `x`. The default tolerance `tol` is zero, i.e., an exact match is required on all columns. For qualifying matches, a distance measure is computed. This is the sum of squares of differences between `x` and `y` after scaling the columns. The default scaling values are `tol`, and for columns with `tol=1` the scale values are set to 1.0 (since they are ignored anyway). Matches (up to `maxmatch` of them) are stored and listed in order of increasing distance.
The `summary` method prints a frequency distribution of the number of matches per observation in `x`, the median of the minimum distances for all matches per `x`, as a function of the number of matches, and the frequency of selection of duplicate observations as those having the smallest distance. The `print` method prints the entire `matches` and `distance` components of the result from `find.matches`.
`matchCases` finds all controls that match cases on a single variable `x` within a tolerance of `tol`. This is intended for prospective cohort studies that use matching for confounder adjustment (even though regression models usually work better).

Usage

```
find.matches(x, y, tol=rep(0, ncol(y)), scale=tol, maxmatch=10)
## S3 method for class 'find.matches':
summary(object, ...)
## S3 method for class 'find.matches':
print(x, digits, ...)

matchCases(xcase,    ycase,    idcase=names(ycase),
           xcontrol, ycontrol, idcontrol=names(ycontrol),
           tol=NULL,
           maxobs=max(length(ycase),length(ycontrol))*10,
           maxmatch=20, which=c('closest','random'))
```

Arguments

`x`	a numeric matrix or the result of `find.matches`
`y`	a numeric matrix with same number of columns as `x`
`xcase`	
`xcontrol`	vectors, not necessarily of the same length, specifying a numeric variable used to match cases and control
`ycase`	
`ycontrol`	vectors or matrices, not necessarily having the same number of rows, specifying a variable to carry along from cases and matching controls. If you instead want to carry along rows from a data frame, let `ycase` and `ycontrol` be non-overlapping integer subscripts of the donor data frame.
`tol`	a vector of tolerances with number of elements the same as the number of columns of `y`, for `find.matches`. For `matchCases` is a scalar tolerance.
`scale`	a vector of scaling constants with number of elements the same as the number of columns of `y`.
`maxmatch`	maximum number of matches to allow. For `matchCases`, maximum number of controls to match with a case (default is 20). If more than `maxmatch` matching controls are available, a random sample without replacement of `maxmatch` controls is used (if `which="random"`).
`object`	an object created by `find.matches`
`digits`	number of digits to use in printing distances
`idcase`	
`idcontrol`	vectors the same length as `xcase` and `xcontrol` respectively, specifying the id of cases and controls. Defaults are integers specifying original element positions within each of cases and controls.
`maxobs`	maximum number of cases and all matching controls combined (maximum dimension of data frame resulting from `matchControls`). Default is ten times the maximum of the number of cases and number of controls. `maxobs` is used to allocate space for the resulting data frame.
`which`	set to `"closest"` (the default) to match cases with up to `maxmatch` controls that most closely match on `x`. Set `which="random"` to use randomly chosen controls. In either case, only those controls within `tol` on `x` are allowed to be used.
`...`	unused

Value

`find.matches` returns a list of class `find.matches` with elements `matches` and `distance`. Both elements are matrices with the number of rows equal to the number of rows in `x`, and with `k` columns, where `k` is the maximum number of matches (`<= maxmatch`) that occurred. The elements of `matches` are row identifiers of `y` that match, with zeros if fewer than `maxmatch` matches are found (blanks if `y` had row names). `matchCases` returns a data frame with variables `idcase` (id of case currently being matched), `type` (factor variable with levels `"case"` and `"control"`), `id` (id of case if case row, or id of matching case), and `y`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Ming K, Rosenbaum PR (2001): A note on optimal matching with variable controls using the assignment algorithm. J Comp Graph Stat 10:455–463.

Cepeda MS, Boston R, Farrar JT, Strom BL (2003): Optimal matching with a variable number of controls vs. a fixed number of controls for a cohort study: trade-offs. J Clin Epidemiology 56:230-237. Note: These papers were not used for the functions here but probably should have been.

See Also

`scale`, `apply`

Examples

```
y <- rbind(c(.1, .2),c(.11, .22), c(.3, .4), c(.31, .41), c(.32, 5))
x <- rbind(c(.09,.21), c(.29,.39))
y
x
w <- find.matches(x, y, maxmatch=5, tol=c(.05,.05))

set.seed(111)       # so can replicate results
x <- matrix(runif(500), ncol=2)
y <- matrix(runif(2000), ncol=2)
w <- find.matches(x, y, maxmatch=5, tol=c(.02,.03))
w$matches[1:5,]
w$distance[1:5,]
# Find first x with 3 or more y-matches
num.match <- apply(w$matches, 1, function(x)sum(x > 0))
j <- ((1:length(num.match))[num.match > 2])[1]
x[j,]
y[w$matches[j,],]

summary(w)

# For many applications would do something like this:
# attach(df1)
# x <- cbind(age, sex) # Just do as.matrix(df1) if df1 has no factor objects
# attach(df2)
# y <- cbind(age, sex)
# mat <- find.matches(x, y, tol=c(5,0)) # exact match on sex, 5y on age

# Demonstrate matchCases
xcase     <- c(1,3,5,12)
xcontrol  <- 1:6
idcase    <- c('A','B','C','D')
```

```
idcontrol <- c('a','b','c','d','e','f')
ycase     <- c(11,33,55,122)
ycontrol  <- c(11,22,33,44,55,66)
matchCases(xcase, ycase, idcase,
           xcontrol, ycontrol, idcontrol, tol=1)

# If y is a binary response variable, the following code
# will produce a Mantel-Haenszel summary odds ratio that
# utilizes the matching.
# Standard variance formula will not work here because
# a control will match more than one case
# WARNING: The M-H procedure exemplified here is suspect
# because of the small strata and widely varying number
# of controls per case.

x    <- c(1, 2, 3, 3, 3, 6, 7, 12,  1, 1:7)
y    <- c(0, 0, 0, 1, 0, 1, 1,  1,  1, 0, 0, 0, 0, 1, 1, 1)
case <- c(rep(TRUE, 8), rep(FALSE, 8))
id   <- 1:length(x)

m <- matchCases(x[case],  y[case],  id[case],
                x[!case], y[!case], id[!case], tol=1)
iscase <- m$type=='case'
# Note: the first tapply on insures that event indicators are
# sorted by case id.  The second actually does something.
event.case    <- tapply(m$y[iscase],  m$idcase[iscase],  sum)
event.control <- tapply(m$y[!iscase], m$idcase[!iscase], sum)
n.control     <- tapply(!iscase,      m$idcase,          sum)
n             <- tapply(m$y,          m$idcase,          length)
or <- sum(event.case * (n.control - event.control) / n) /
      sum(event.control * (1 - event.case) / n)
or

# Bootstrap this estimator by sampling with replacement from
# subjects.  Assumes id is unique when combine cases+controls
# (id was constructed this way above).  The following algorithms
# puts all sampled controls back with the cases to whom they were
# originally matched.

ids <- unique(m$id)
idgroups <- split(1:nrow(m), m$id)
B   <- 50   # in practice use many more
ors <- numeric(B)
# Function to order w by ids, leaving unassigned elements zero
align <- function(ids, w) {
  z <- structure(rep(0, length(ids)), names=ids)
  z[names(w)] <- w
  z
}
for(i in 1:B) {
  j <- sample(ids, replace=TRUE)
  obs <- unlist(idgroups[j])
  u <- m[obs,]
```

```
    iscase <- u$type=='case'
    n.case <- align(ids, tapply(u$type, u$idcase,
                                function(v)sum(v=='case')))
    n.control <- align(ids, tapply(u$type, u$idcase,
                                   function(v)sum(v=='control')))
    event.case <- align(ids, tapply(u$y[iscase],  u$idcase[iscase],  sum))
    event.control <- align(ids, tapply(u$y[!iscase], u$idcase[!iscase], sum))
    n <- n.case + n.control
    # Remove sets having 0 cases or 0 controls in resample
    s             <- n.case > 0 & n.control > 0
    denom <- sum(event.control[s] * (n.case[s] - event.case[s]) / n[s])
    or <- if(denom==0) NA else
     sum(event.case[s] * (n.control[s] - event.control[s]) / n[s]) / denom
    ors[i] <- or
  }
  describe(ors)
```

`first.word` *First Word in a String or Expression*

Description

`first.word` finds the first word in an expression. A word is defined by unlisting the elements of the expression found by the S parser and then accepting any elements whose first character is either a letter or period. The principal intended use is for the automatic generation of temporary file names where it is important to exclude special characters from the file name. For Microsoft Windows, periods in names are deleted and only up to the first 8 characters of the word is returned.

Usage

```
first.word(x, i=1, expr=substitute(x))
```

Arguments

`x`	any scalar character string
`i`	word number, default value = 1. Used when the second or `i`th word is wanted. Currently only the `i=1` case is implemented.
`expr`	any S object of mode `expression`.

Value

a character string

Author(s)

Frank E. Harrell, Jr.,
Department of Biostatistics,
Vanderbilt University,
`f.harrell@vanderbilt.edu`

Richard M. Heiberger,
Department of Statistics,
Temple University, Philadelphia, PA.
`rmh@astro.ocis.temple.edu`

Examples

```
first.word(expr=expression(y ~ x + log(w)))
```

`format.df` *Format a Data Frame or Matrix for LaTeX or HTML*

Description

`format.df` does appropriate rounding and decimal alignment, and outputs a character matrix containing the formatted data. If `x` is a data.frame, then do each component separately. If `x` is a matrix, but not a data.frame, make it a data.frame with individual components for the columns. If a component `x$x` is a matrix, then do all columns the same.

Usage

```
format.df(x,
          digits, dec=NULL, rdec=NULL, cdec=NULL,
          numeric.dollar=cdot,
          na.blank=FALSE, na.dot=FALSE, blank.dot=FALSE,
          col.just=NULL, cdot=FALSE, dcolumn=FALSE, matrix.sep=' ',
          scientific=c(-4,4), math.row.names, math.col.names, ...)
```

Arguments

`x`	a matrix (usually numeric) or data frame
`digits`	causes all values in the table to be formatted to `digits` significant digits. `dec` is usually preferred.
`dec`	If `dec` is a scalar, all elements of the matrix will be rounded to `dec` decimal places to the right of the decimal. `dec` can also be a matrix whose elements correspond to `x`, for customized rounding of each element. A matrix `dec` must have number of columns equal to number of columns of input `x`. A scalar `dec` is expanded to a vector `cdec` with number of items equal to number of columns of input `x`.

`rdec`	a vector specifying the number of decimal places to the right for each row (`cdec` is more commonly used than `rdec`) A vector `rdec` must have number of items equal to number of rows of input `x`. `rdec` is expanded to matrix `dec`.
`cdec`	a vector specifying the number of decimal places for each column. The vector must have number of items equal to number of columns or components of input x.
`cdot`	Set to `TRUE` to use centered dots rather than ordinary periods in numbers. The output uses a syntax appropriate for `latex`.
`na.blank`	Set to `TRUE` to use blanks rather than `NA` for missing values. This usually looks better in `latex`.
`dcolumn`	Set to `TRUE` to use David Carlisle's `dcolumn` style for decimal alignment in `latex`. Default is `FALSE`. You will probably want to use `dcolumn` if you use `rdec`, as a column may then contain varying number of places to the right of the decimal. `dcolumn` can line up all such numbers on the decimal point, with integer values right justified at the decimal point location of numbers that actually contain decimal places. When you use `dcolumn=TRUE`, `numeric.dollar` is set by default to `FALSE`. When you use `dcolumn=TRUE`, the `"style"` element is set to `"dcolumn"` as the `latex \usepackage` must reference `[dcolumn]`. The three files `dcolumn.sty`, `newarray.sty`, and `array.sty` will need to be in a directory in your `$TEXINPUTS` path. When you use `dcolumn=TRUE`, `numeric.dollar` should be set to `FALSE`.
`numeric.dollar`	logical, default `!dcolumn`. Set to `TRUE` to place dollar signs around numeric values when `dcolumn=FALSE`. This assures that `latex` will use minus signs rather than hyphens to indicate negative numbers. Set to `FALSE` when `dcolumn=TRUE`, as `dcolumn.sty` automatically uses minus signs.
`math.row.names`	logical, set true to place dollar signs around the row names.
`math.col.names`	logical, set true to place dollar signs around the column names.
`na.dot`	Set to `TRUE` to use periods rather than `NA` for missing numeric values. This works with the `sas` convention that periods indicate missing values.
`blank.dot`	Set to `TRUE` to use periods rather than blanks for missing character values. This works with the `sas` convention that periods indicate missing values.
`col.just`	Input vector `col.just` must have number of columns equal to number of columns of the output matrix. When `NULL`, the default, the `col.just` attribute of the result is set to `"l"` for character columns and to `"r"` for numeric columns. The user can override the default by an argument vector whose length is equal to the number of columns of the result matrix. When `format.df` is called by `latex.default`, the `col.just` is used as the `cols` argument to the `\tabular` environment and the letters `"l"`, `"r"`, and `"c"` are valid values. When `format.df` is called by `sas`, the `col.just` is used to determine whether a `$` is needed on the `input` line of the `sysin` file, and the letters `"l"` and `"r"` are valid values.
`matrix.sep`	When `x` is a data frame containing a matrix, so that new column names are constructed from the name of the matrix object and the names of the individual

columns of the matrix, `matrix.sep` specifies the character to use to separate object names from individual column names.

`scientific`	specifies ranges of exponents (or a logical vector) specifying values not to convert to scientific notation. See `format.default` for details.
`...`	other arguments are accepted and ignored. For `latexVerbatim` these arguments are passed to the `print` function.

Value

a character matrix with character images of properly rounded `x`. Matrix components of input `x` are now just sets of columns of character matrix. `attr(,col.just)` repeats the input `col.just` when provided, otherwise, it includes the recommended justification for columns of output. See the discussion of the argument `col.just`. The default justification is `"l"` for characters and factors, `"r"` for numeric. When `dcolumn==TRUE`, numerics will have `"."` as the justification character.

Author(s)

Frank E. Harrell, Jr.,
Department of Biostatistics,
Vanderbilt University,
`f.harrell@vanderbilt.edu`

Richard M. Heiberger,
Department of Statistics,
Temple University, Philadelphia, PA.
`rmh@astro.ocis.temple.edu`

See Also

`latex`

Examples

```
x <- data.frame(a=1:2, b=3:4)
x$m <- matrix(5:8,nrow=2)
names(x)
dim(x)
x
format.df(x)
dim(format.df(x))
```

`format.pval` *Format P Values*

Description

`format.pval` is intended for formatting p-values.

Usage

```
format.pval(x, pv=x, digits = max(1, .Options$digits - 2), eps = .Machine$double.ep
```

Arguments

`pv`	a numeric vector.
`x`	argument for method compliance.
`digits`	how many significant digits are to be used.
`eps`	a numerical tolerance: see Details.
`na.form`	character representation of `NA`s.
`...`	arguments passed to `format` in the `format.pval` function body.

Details

`format.pval` is mainly an auxiliary function for `print.summary.lm` etc., and does separate formatting for fixed, floating point and very small values; those less than `eps` are formatted as `"< [eps]"` (where "[eps]" stands for `format(eps, digits)`).

Value

A character vector.

Note

This is the base `format.pval` function with the ablitiy to pass an `nsmall` argument to `format`

Examples

```
format.pval(c(runif(5), pi^-100, NA))
format.pval(c(0.1, 0.0001, 1e-27))
format.pval(c(0.1, 1e-27), nsmall=3)
```

`gbayes` *Gaussian Bayesian Posterior and Predictive Distributions*

Description

`gbayes` derives the (Gaussian) posterior and optionally the predictive distribution when both the prior and the likelihood are Gaussian, and when the statistic of interest comes from a 2-sample problem. This function is especially useful in obtaining the expected power of a statistical test, averaging over the distribution of the population effect parameter (e.g., log hazard ratio) that is obtained using pilot data. `gbayes` is also useful for summarizing studies for which the statistic of interest is approximately Gaussian with known variance. An example is given for comparing two proportions using the angular transformation, for which the variance is independent of unknown parameters except for very extreme probabilities. A `plot` method is also given. This plots the prior, posterior, and predictive distributions on a single graph using a nice default for the x-axis limits and using the `labcurve` function for automatic labeling of the curves.

`gbayes2` uses the method of Spiegelhalter and Freedman (1986) to compute the probability of correctly concluding that a new treatment is superior to a control. By this we mean that a 1-`alpha` normal theory-based confidence interval for the new minus old treatment effect lies wholly to the right of `delta.w`, where `delta.w` is the minimally worthwhile treatment effect (which can be zero to be consistent with ordinary null hypothesis testing, a method not always making sense). This kind of power function is averaged over a prior distribution for the unknown treatment effect. This procedure is applicable to the situation where a prior distribution is not to be used in constructing the test statistic or confidence interval, but is only used for specifying the distribution of `delta`, the parameter of interest.

Even though `gbayes2` assumes that the test statistic has a normal distribution with known variance (which is strongly a function of the sample size in the two treatment groups), the prior distribution function can be completely general. Instead of using a step-function for the prior distribution as Spiegelhalter and Freedman used in their appendix, `gbayes2` uses the built-in `integrate` function for numerical integration. `gbayes2` also allows the variance of the test statistic to be general as long as it is evaluated by the user. The conditional power given the parameter of interest `delta` is `1 - pnorm((delta.w - delta)/sd + z)`, where z is the normal critical value corresponding to 1 - `alpha`/2.

`gbayesMixPredNoData` derives the predictive distribution of a statistic that is Gaussian given `delta` when no data have yet been observed and when the prior is a mixture of two Gaussians.

`gbayesMixPost` derives the posterior density or cdf of `delta` given the statistic `x`, when the prior for `delta` is a mixture of two Gaussians and when `x` is Gaussian given `delta`.

`gbayesMixPowerNP` computes the power for a test for `delta` > `delta.w` for the case where (1) a Gaussian prior or mixture of two Gaussian priors is used as the prior distribution, (2) this prior is used in forming the statistical test or credible interval, (3) no prior is used for the distribution of `delta` for computing power but instead a fixed single `delta` is given (as in traditional frequentist hypothesis tests), and (4) the test statistic has a Gaussian likelihood with known variance (and mean equal to the specified `delta`). `gbayesMixPowerNP` is handy where you want to use an earlier study in testing for treatment effects in a new study, but you want to mix with this prior a non-informative prior. The mixing probability `mix` can be thought of as the "applicability" of the previous study. As with `gbayes2`, power here means the probability that the new study will yield a left credible interval that is to the right of `delta.w`. `gbayes1PowerNP` is a special case of `gbayesMixPowerNP` when the prior is a single Gaussian.

Usage

```
gbayes(mean.prior, var.prior, m1, m2, stat, var.stat,
       n1, n2, cut.prior, cut.prob.prior=0.025)

## S3 method for class 'gbayes':
plot(x, xlim, ylim, name.stat='z', ...)

gbayes2(sd, prior, delta.w=0, alpha=0.05, upper=Inf, prior.aux)

gbayesMixPredNoData(mix=NA, d0=NA, v0=NA, d1=NA, v1=NA,
                    what=c('density','cdf'))
```

```
gbayesMixPost(x=NA, v=NA, mix=1, d0=NA, v0=NA, d1=NA, v1=NA,
              what=c('density','cdf'))

gbayesMixPowerNP(pcdf, delta, v, delta.w=0, mix, interval,
                 nsim=0, alpha=0.05)

gbayes1PowerNP(d0, v0, delta, v, delta.w=0, alpha=0.05)
```

Arguments

`mean.prior`	mean of the prior distribution
`cut.prior` `cut.prob.prior`	
`var.prior`	variance of the prior. Use a large number such as 10000 to effectively use a flat (noninformative) prior. Sometimes it is useful to compute the variance so that the prior probability that `stat` is greater than some impressive value `u` is only `alpha`. The correct `var.prior` to use is then `((u-mean.prior)/qnorm(1-alpha))^2`. You can specify `cut.prior=u` and `cut.prob.prior=alpha` (whose default is 0.025) in place of `var.prior` to have `gbayes` compute the prior variance in this manner.
`m1`	sample size in group 1
`m2`	sample size in group 2
`stat`	statistic comparing groups 1 and 2, e.g., log hazard ratio, difference in means, difference in angular transformations of proportions
`var.stat`	variance of `stat`, assumed to be known. `var.stat` should either be a constant (allowed if `n1` is not specified), or a function of two arguments which specify the sample sizes in groups 1 and 2. Calculations will be approximate when the variance is estimated from the data.
`x`	an object returned by `gbayes` or the value of the statistic which is an estimator of delta, the parameter of interest
`sd`	the standard deviation of the treatment effect
`prior`	a function of possibly a vector of unknown treatment effects, returning the prior density at those values
`pcdf`	a function computing the posterior CDF of the treatment effect `delta`, such as a function created by `gbayesMixPost` with `what="cdf"`.
`delta`	a true unknown single treatment effect to detect
`v`	the variance of the statistic `x`, e.g., `s^2 * (1/n1 + 1/n2)`. Neither `x` nor `v` need to be defined to `gbayesMixPost`, as they can be defined at run time to the function created by `gbayesMixPost`.
`n1`	number of future observations in group 1, for obtaining a predictive distribution
`n2`	number of future observations in group 2
`xlim`	vector of 2 x-axis limits. Default is the mean of the posterior plus or minus 6 standard deviations of the posterior.

`ylim`	vector of 2 y-axis limits. Default is the range over combined prior and posterior densities.
`name.stat`	label for x-axis. Default is `"z"`.
`...`	optional arguments passed to `labcurve` from `plot.gbayes`
`delta.w`	the minimum worthwhile treatment difference to detech. The default is zero for a plain uninteristing null hypothesis.
`alpha`	type I error, or more accurately one minus the confidence level for a two-sided confidence limit for the treatment effect
`upper`	upper limit of integration over the prior distribution multiplied by the normal likelihood for the treatment effect statistic. Default is infinity.
`prior.aux`	argument to pass to `prior` from `integrate` through `gbayes2`. Inside of `power` the argument must be named `prior.aux` if it exists. You can pass multiple parameters by passing `prior.aux` as a list and pulling off elements of the list inside `prior`. This setup was used because of difficulties in passing `...` arguments through `integrate` for some situations.
`mix`	mixing probability or weight for the Gaussian prior having mean `d0` and variance `v0`. `mix` must be between 0 and 1, inclusive.
`d0`	mean of the first Gaussian distribution (only Gaussian for `gbayes1PowerNP` and is a required argument)
`v0`	variance of the first Gaussian (only Gaussian for `gbayes1PowerNP` and is a required argument)
`d1`	mean of the second Gaussian (if `mix` < 1)
`v1`	variance of the second Gaussian (if `mix` < 1). Any of these last 5 arguments can be omitted to `gbayesMixPredNoData` as they can be provided at run time to the function created by `gbayesMixPredNoData`.
`what`	specifies whether the predictive density or the CDF is to be computed. Default is `"density"`.
`interval`	a 2-vector containing the lower and upper limit for possible values of the test statistic `x` that would result in a left credible interval exceeding `delta.w` with probability 1-`alpha`/2
`nsim`	defaults to zero, causing `gbayesMixPowerNP` to solve numerically for the critical value of `x`, then to compute the power accordingly. Specify a nonzero number such as 20000 for `nsim` to instead have the function estimate power by simulation. In this case 0.95 confidence limits on the estimated power are also computed. This approach is sometimes necessary if `uniroot` can't solve the equation for the critical value.

Value

`gbayes` returns a list of class `"gbayes"` containing the following names elements: `mean.prior`,`var.prior`,`mean.po`
`var.post`, and if `n1` is specified, `mean.pred` and `var.pred`. Note that `mean.pred` is identical to `mean.post`. `gbayes2` returns a single number which is the probability of correctly rejecting the null hypothesis in favor of the new treatment. `gbayesMixPredNoData` returns a function that can be used to evaluate the predictive density or cumulative distribution. `gbayesMixPost`

returns a function that can be used to evaluate the posterior density or cdf. `gbayesMixPowerNP` returns a vector containing two values if `nsim` = 0. The first value is the critical value for the test statistic that will make the left credible interval > `delta.w`, and the second value is the power. If `nsim` > 0, it returns the power estimate and confidence limits for it if `nsim` > 0. The examples show how to use these functions.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Spiegelhalter DJ, Freedman LS, Parmar MKB (1994): Bayesian approaches to randomized trials. JRSS A 157:357–416. Results for `gbayes` are derived from Equations 1, 2, 3, and 6.

Spiegelhalter DJ, Freedman LS (1986): A predictive approach to selecting the size of a clinical trial, based on subjective clinical opinion. Stat in Med 5:1–13.

Joseph, Lawrence and Belisle, Patrick (1997): Bayesian sample size determination for normal means and differences between normal means. The Statistician 46:209–226.

Examples

```
# Compare 2 proportions using the var stabilizing transformation
# arcsin(sqrt((x+3/8)/(n+3/4))) (Anscombe), which has variance
# 1/[4(n+.5)]

m1 <- 100;     m2 <- 150
deaths1 <- 10; deaths2 <- 30

f <- function(events,n) asin(sqrt((events+3/8)/(n+3/4)))
stat <- f(deaths1,m1) - f(deaths2,m2)
var.stat <- function(m1, m2) 1/4/(m1+.5) + 1/4/(m2+.5)
cat("Test statistic:",format(stat),"  s.d.:",
    format(sqrt(var.stat(m1,m2))), "\n")
#Use unbiased prior with variance 1000 (almost flat)
b <- gbayes(0, 1000, m1, m2, stat, var.stat, 2*m1, 2*m2)
print(b)
plot(b)
#To get posterior Prob[parameter > w] use
# 1-pnorm(w, b$mean.post, sqrt(b$var.post))

#If g(effect, n1, n2) is the power function to
#detect an effect of 'effect' with samples size for groups 1 and 2
#of n1,n2, estimate the expected power by getting 1000 random
#draws from the posterior distribution, computing power for
#each value of the population effect, and averaging the 1000 powers
#This code assumes that g will accept vector-valued 'effect'
#For the 2-sample proportion problem just addressed, 'effect'
#could be taken approximately as the change in the arcsin of
```

```
#the square root of the probability of the event

g <- function(effect, n1, n2, alpha=.05) {
  sd <- sqrt(var.stat(n1,n2))
  z <- qnorm(1 - alpha/2)
  effect <- abs(effect)
  1 - pnorm(z - effect/sd) + pnorm(-z - effect/sd)
}

effects <- rnorm(1000, b$mean.post, sqrt(b$var.post))
powers <- g(effects, 500, 500)
hist(powers, nclass=35, xlab='Power')
describe(powers)



# gbayes2 examples
# First consider a study with a binary response where the
# sample size is n1=500 in the new treatment arm and n2=300
# in the control arm.  The parameter of interest is the
# treated:control log odds ratio, which has variance
# 1/[n1 p1 (1-p1)] + 1/[n2 p2 (1-p2)].  This is not
# really constant so we average the variance over plausible
# values of the probabilities of response p1 and p2.  We
# think that these are between .4 and .6 and we take a
# further short cut

v <- function(n1, n2, p1, p2) 1/(n1*p1*(1-p1)) + 1/(n2*p2*(1-p2))
n1 <- 500; n2 <- 300
ps <- seq(.4, .6, length=100)
vguess <- quantile(v(n1, n2, ps, ps), .75)
vguess
#        75%
# 0.02183459

# The minimally interesting treatment effect is an odds ratio
# of 1.1.  The prior distribution on the log odds ratio is
# a 50:50 mixture of a vague Gaussian (mean 0, sd 100) and
# an informative prior from a previous study (mean 1, sd 1)

prior <- function(delta)
  0.5*dnorm(delta, 0, 100)+0.5*dnorm(delta, 1, 1)
deltas <- seq(-5, 5, length=150)
plot(deltas, prior(deltas), type='l')

# Now compute the power, averaged over this prior
gbayes2(sqrt(vguess), prior, log(1.1))
# [1] 0.6133338

# See how much power is lost by ignoring the previous
# study completely

gbayes2(sqrt(vguess), function(delta)dnorm(delta, 0, 100), log(1.1))
```

```
# [1] 0.4984588

# What happens to the power if we really don't believe the treatment
# is very effective?  Let's use a prior distribution for the log
# odds ratio that is uniform between log(1.2) and log(1.3).
# Also check the power against a true null hypothesis

prior2 <- function(delta) dunif(delta, log(1.2), log(1.3))
gbayes2(sqrt(vguess), prior2, log(1.1))
# [1] 0.1385113

gbayes2(sqrt(vguess), prior2, 0)
# [1] 0.3264065

# Compare this with the power of a two-sample binomial test to
# detect an odds ratio of 1.25
bpower(.5, odds.ratio=1.25, n1=500, n2=300)
#     Power
# 0.3307486

# For the original prior, consider a new study with equal
# sample sizes n in the two arms.  Solve for n to get a
# power of 0.9.  For the variance of the log odds ratio
# assume a common p in the center of a range of suspected
# probabilities of response, 0.3.  For this example we
# use a zero null value and the uniform prior above

v   <- function(n) 2/(n*.3*.7)
pow <- function(n) gbayes2(sqrt(v(n)), prior2)
uniroot(function(n) pow(n)-0.9, c(50,10000))$root
# [1] 2119.675
# Check this value
pow(2119.675)
# [1] 0.9

# Get the posterior density when there is a mixture of two priors,
# with mixing probability 0.5.  The first prior is almost
# non-informative (normal with mean 0 and variance 10000) and the
# second has mean 2 and variance 0.3.  The test statistic has a value
# of 3 with variance 0.4.
f <- gbayesMixPost(3, 4, mix=0.5, d0=0, v0=10000, d1=2, v1=0.3)

args(f)

# Plot this density
delta <- seq(-2, 6, length=150)
plot(delta, f(delta), type='l')

# Add to the plot the posterior density that used only
# the almost non-informative prior
lines(delta, f(delta, mix=1), lty=2)

# The same but for an observed statistic of zero
```

```
lines(delta, f(delta, mix=1, x=0), lty=3)

# Derive the CDF instead of the density
g <- gbayesMixPost(3, 4, mix=0.5, d0=0, v0=10000, d1=2, v1=0.3,
                   what='cdf')
# Had mix=0 or 1, gbayes1PowerNP could have been used instead
# of gbayesMixPowerNP below

# Compute the power to detect an effect of delta=1 if the variance
# of the test statistic is 0.2
gbayesMixPowerNP(g, 1, 0.2, interval=c(-10,12))

# Do the same thing by simulation
gbayesMixPowerNP(g, 1, 0.2, interval=c(-10,12), nsim=20000)

# Compute by what factor the sample size needs to be larger
# (the variance needs to be smaller) so that the power is 0.9
ratios <- seq(1, 4, length=50)
pow <- single(50)
for(i in 1:50)
  pow[i] <- gbayesMixPowerNP(g, 1, 0.2/ratios[i], interval=c(-10,12))[2]

# Solve for ratio using reverse linear interpolation
approx(pow, ratios, xout=0.9)$y

# Check this by computing power
gbayesMixPowerNP(g, 1, 0.2/2.1, interval=c(-10,12))
# So the study will have to be 2.1 times as large as earlier thought
```

---

getHdata *Download and Install Datasets for Hmisc, Design, and Statistical Modeling*

Description

This function downloads and makes ready to use datasets from the main web site for the Hmisc and Design libraries. For R, the datasets were stored in compressed `save` format and `getHdata` makes them available by running `load()` after download. For S-Plus, the datasets were stored in `data.dump` format and are made available by running `data.restore()` after import. The dataset is run through the `cleanup.import` function to reduce multiple inheritance problems for SV4 (S-Plus 5 or later). Calling `getHdata` with no `file` argument provides a character vector of names of available datasets that are currently on the web site. For R, R's default browser can optionally be launched to view `html` files that were already prepared using the Hmisc command `html(contents( ))` or to view `.txt` or `.html` data description files when available.

Usage

```
getHdata(file, what = c("data", "contents", "description", "all"),
         where="http://biostat.mc.vanderbilt.edu/twiki/pub/Main/DataSets")
```

Arguments

`file`	an unquoted name of a dataset on the web site, e.g. `prostate`. Omit `file` to obtain a list of available datasets.
`what`	specify `what="contents"` to browse the contents (metadata) for the dataset rather than fetching the data themselves. Specify `what="description"` to browse a data description file if available. Specify `what="all"` to retrieve the data and see the metadata and description.
`where`	URL containing the data and metadata files

Details

For S-Plus, Hmisc defines a function `download.file` that is used by `getHdata`. This is a stripped-down version of the R `download.file` function that uses the system `wget` executable for fetching files from the Internet. For Unix and Linux systems, `wget` will be pre-installed usually. For windows S-Plus systems, get `wget` from `ftp://sunsite.dk/projects/wget/windows`. Once you unzip the file from there, move `wget.exe` to the same Windows directory that contains `ftp.exe`.

Value

`getHdata()` without a `file` argument returns a character vector of dataset base names. When a dataset is downloaded, the data frame is placed in search position one and is not returned as value of `getHdata`.

Author(s)

Frank Harrell

See Also

`download.file`, `cleanup.import`, `data.restore`, `load`

Examples

```
## Not run:
getHdata()           # download list of available datasets
getHdata(prostate)   # downloads, load( ) or data.restore( )
                     # runs cleanup.import for S-Plus 6
getHdata(valung, "contents")   # open browser (options(browser="whatever"))
                     # after downloading valung.html
                     # (result of html(contents()))
getHdata(support, "all")  # download and open one browser window
datadensity(support)
attach(support)      # make individual variables available
getHdata(plasma,  "all")  # download and open two browser windows
                          # (description file is available for plasma)
## End(Not run)
```

getZip *Open a Zip File From a URL.*

Description

Allows downloading and reading of a zip file containing one file

Usage

```
getZip(url, password=NULL)
```

Arguments

`url`	either a path to a local file or a valid URL.
`password`	required to decode password-protected zip files

Details

Allows downloading and reading of zip file containing one file. The file may be password protected. If a password is needed then one will be requested unless given.

Note: to make password-protected zip file z.zip, do zip -e z myfile

Value

Returns a file O/I pipe.

Author(s)

Frank E. Harrell

See Also

`pipe`

Examples

```
## Not run:
read.csv(getZip('http://biostat.mc.vanderbilt.edu/twiki/pub/Sandbox/WebHome/z.zip'))
## Password is 'foo'
## End(Not run)
```

`hdquantile` *Harrell-Davis Distribution-Free Quantile Estimator*

Description

Computes the Harrell-Davis (1982) quantile estimator and jacknife standard errors of quantiles. The quantile estimator is a weighted linear combination or order statistics in which the order statistics used in traditional nonparametric quantile estimators are given the greatest weight. In small samples the H-D estimator is more efficient than traditional ones, and the two methods are asymptotically equivalent. The H-D estimator is the limit of a bootstrap average as the number of bootstrap resamples becomes infinitely large.

Usage

```
hdquantile(x, probs = seq(0, 1, 0.25),
           se = FALSE, na.rm = FALSE, names = TRUE, weights=FALSE)
```

Arguments

`x`	a numeric vector
`probs`	vector of quantiles to compute
`se`	set to `TRUE` to also compute standard errors
`na.rm`	set to `TRUE` to remove `NA`s from `x` before computing quantiles
`names`	set to `FALSE` to prevent names attributions from being added to quantiles and standard errors
`weights`	set to `TRUE` to return a `"weights"` attribution with the matrix of weights used in the H-D estimator corresponding to order statistics, with columns corresponding to quantiles.

Details

A Fortran routine is used to compute the jackknife leave-out-one quantile estimates. Standard errors are not computed for quantiles 0 or 1 (`NA`s are returned).

Value

A vector of quantiles. If `se=TRUE` this vector will have an attribute `se` added to it, containing the standard errors. If `weights=TRUE`, also has a `"weights"` attribute which is a matrix.

Author(s)

Frank Harrell

References

Harrell FE, Davis CE (1982): A new distribution-free quantile estimator. Biometrika 69:635-640.

Hutson AD, Ernst MD (2000): The exact bootstrap mean and variance of an L-estimator. J Roy Statist Soc B 62:89-94.

See Also

`quantile`

Examples

```
set.seed(1)
x <- runif(100)
hdquantile(x, (1:3)/4, se=TRUE)

## Not run:
# Compare jackknife standard errors with those from the bootstrap
library(boot)
boot(x, function(x,i) hdquantile(x[i], probs=(1:3)/4), R=400)
## End(Not run)
```

`hist.data.frame` *Histograms for Variables in a Data Frame*

Description

This functions tries to compute the maximum number of histograms that will fit on one page, then it draws a matrix of histograms. If there are more qualifying variables than will fit on a page, the function waits for a mouse click before drawing the next page.

Usage

```
## S3 method for class 'data.frame':
hist(x, n.unique = 3, nclass = "compute",
                na.big = FALSE, rugs = FALSE, mtitl = FALSE, ...)
# For S-Plus you must use hist.data.frame( ) as hist is not generic there
```

Arguments

`x`	a data frame
`n.unique`	minimum number of unique values a variable must have before a histogram is drawn
`nclass`	number of bins. Default is max(2,trunc(min(n/10,25*log(n,10))/2)), where n is the number of non-missing values for a variable.
`na.big`	set to `TRUE` to draw the number of missing values on the top of the histogram in addition to in a subtitle. In the subtitle, n is the number of non-missing values and m is the number of missing values

`rugs`	set to `TRUE` to add rug plots at the top of each histogram
`mtitl`	set to a character string to set aside extra outside top margin and to use the string for an overall title
`...`	arguments passed to `scat1d`

Value

the number of pages drawn

Author(s)

Frank E Harrell Jr

See Also

`hist`, `scat1d`

Examples

```
d <- data.frame(a=runif(200), b=rnorm(200),
                w=factor(sample(c('green','red','blue'), 200, TRUE)))
hist.data.frame(d)   # in R, just say hist(d)
```

`histbackback` *Back to Back Histograms*

Description

Takes two vectors or a list with `x` and `y` components, and produces back to back histograms of the two datasets.

Usage

```
histbackback(x, y, brks=NULL, xlab=NULL, axes=TRUE, probability=FALSE,
             xlim=NULL, ylab='', ...)
```

Arguments

`x,y`	either two vectors or a list given as `x` with two components. If the components have names, they will be used to label the axis (modification FEH).
`brks`	vector of the desired breakpoints for the histograms.
`xlab`	a vector of two character strings naming the two datasets.
`axes`	logical flag stating whether or not to label the axes.
`probability`	logical flag: if `TRUE`, then the x-axis corresponds to the units for a density. If `FALSE`, then the units are counts.

`xlim`	x-axis limits. First value must be negative, as the left histogram is placed at negative x-values. Second value must be positive, for the right histogram. To make the limits symmetric, use e.g. `ylim=c(-20,20)`.
`ylab`	label for y-axis. Default is no label.
`...`	additional graphics parameters may be given.

Value

a list is returned invisibly with the following components:

`left`	the counts for the dataset plotted on the left.
`right`	the counts for the dataset plotted on the right.
`breaks`	the breakpoints used.

Side Effects

a plot is produced on the current graphics device.

Author(s)

Pat Burns
Salomon Smith Barney
London
pburns@dorado.sbi.com

See Also

`hist`, `histogram`

Examples

```
options(digits=3)
set.seed(1)
histbackback(rnorm(20), rnorm(30))

fool <- list(x=rnorm(40), y=rnorm(40))
histbackback(fool)
age <- rnorm(1000,50,10)
sex <- sample(c('female','male'),1000,TRUE)
histbackback(split(age, sex))
agef <- age[sex=='female']; agem <- age[sex=='male']
histbackback(list(Female=agef,Male=agem), probability=TRUE, xlim=c(-.06,.06))
```

`hoeffd` *Matrix of Hoeffding's D Statistics*

Description

Computes a matrix of Hoeffding's (1948) `D` statistics for all possible pairs of columns of a matrix. `D` is a measure of the distance between `F(x,y)` and `G(x)H(y)`, where `F(x,y)` is the joint CDF of `X` and `Y`, and `G` and `H` are marginal CDFs. Missing values are deleted in pairs rather than deleting all rows of `x` having any missing variables. The `D` statistic is robust against a wide variety of alternatives to independence, such as non-monotonic relationships. The larger the value of `D`, the more dependent are `X` and `Y` (for many types of dependencies). `D` used here is 30 times Hoeffding's original `D`, and ranges from -0.5 to 1.0 if there are no ties in the data. `print.hoeffd` prints the information derived by `hoeffd`. The higher the value of `D`, the more dependent are `x` and `y`.

Usage

```
hoeffd(x)
hoeffd(x, y)
## S3 method for class 'hoeffd':
print(x, ...)
```

Arguments

`x`	a numeric matrix with at least 5 rows and at least 2 columns (if `y` is absent), or an object created by `hoeffd`
`y`	a numeric vector or matrix which will be concatenated to `x`
`...`	ignored

Details

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values are approximated by linear interpolation on the table in Hollander and Wolfe, which uses the asymptotically equivalent Blum-Kiefer-Rosenblatt statistic. For `P<.0001` or `>0.5`, `P` values are computed using a well-fitting linear regression function in `log P` vs. the test statistic. Ranks (but not bivariate ranks) are computed using efficient algorithms (see reference 3).

Value

a list with elements `D`, the matrix of D statistics, `n` the matrix of number of observations used in analyzing each pair of variables, and `P`, the asymptotic P-values. Pairs with fewer than 5 non-missing values have the D statistic set to NA. The diagonals of `n` are the number of non-NAs for the single variable corresponding to that row and column.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Hoeffding W. (1948): A non-parametric test of independence. Ann Math Stat 19:546–57.

Hollander M. and Wolfe D.A. (1973). Nonparametric Statistical Methods, pp. 228–235, 423. New York: Wiley.

Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): Numerical Recipes in C. Cambridge: Cambridge University Press.

See Also

`rcorr`, `varclus`

Examples

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4,   1, 0, 1, 4)
z <- c(1,   2, 3, 4, NA)
q <- c(1,   2, 3, 4, 5)
hoeffd(cbind(x,y,z,q))

# Hoeffding's test can detect even one-to-many dependency
set.seed(1)
x <- seq(-10,10,length=200)
y <- x*sign(runif(200,-1,1))
plot(x,y)
hoeffd(x,y)
```

`html` *Convert an S object to HTML*

Description

`html` is a generic function, for which only two methods are currently implemented, `html.latex` and a rudimentary `html.data.frame`. The former uses the `HeVeA` LaTeX to HTML translator by Maranget to create an HTML file from a LaTeX file like the one produced by `latex`. The resulting HTML file may be displayed using a `show` or a `print` method. The browser specified in `options(browser=)` for R (`help.browser` for S-Plus) is launched to display the HTML file. `html.default` just runs `html.data.frame`.

Usage

```
html(object, ...)
## S3 method for class 'latex':
html(object, file, ...)
## S3 method for class 'data.frame':
html(object,
  file=paste(first.word(deparse(substitute(object))),'html',sep='.'),
     append=FALSE, link=NULL, linkCol=1, linkType=c('href','name'), ...)
## Default S3 method:
html(object,
     file=paste(first.word(deparse(substitute(object))),'html',sep='.'),
     append=FALSE, link=NULL, linkCol=1, linkType=c('href','name'), ...)
## S3 method for class 'html':
print(x, ...)
## S3 method for class 'html':
show(object)
```

Arguments

`object`	a data frame or an object created by `latex`. For `show` is an object created by `html`. For the generic `html` is any object for which an `html` method exists.
`file`	name of the file to create. The default file name is `object.html` where `object` is the first word in the name of the argument for `object`.
`append`	set to `TRUE` to append to an existing file
`link`	character vector specifying hyperlink names to attach to selected elements of the matrix or data frame. No hyperlinks are used if `link` is omitted or for elements of `link` that are `""`. To allow multiple links per link, `link` may also be a character matrix shaped as `object` in which case `linkCol` is ignored.
`linkCol`	column number of `object` to which hyperlinks are attached. Defaults to first column.
`linkType`	defaults to `"href"`
`...`	arguments passed to `format.df`
`x`	an object created by `html`

Side Effects

`print` or `show` launch a browser

Author(s)

Frank E. Harrell, Jr.
Department of Biostatistics,
Vanderbilt University,
⟨f.harrell@vanderbilt.edu⟩

References

Maranget, Luc. HeVeA: a LaTeX to HTML translater. URL: http://para.inria.fr/ maranget/hevea/

See Also

latex

Examples

```
## Not run:
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','e')))
w <- latex(x)
h <- html(w) # run HeVeA to convert .tex to .html
h <- html(x) # convert x directly to html
options(browser='konqueror')  # use help.browser for S-Plus
h             # launch html browser by running print.html
w <- html(x, link=c('','B'))   # hyperlink first row first col to B
## End(Not run)
```

---

impute *Generic Functions and Methods for Imputation*

Description

These functions do simple and `transcan` imputation and print, summarize, and subscript variables that have NAs filled-in with imputed values. The simple imputation method involves filling in NAs with constants, with a specified single-valued function of the non-NAs, or from a sample (with replacement) from the non-NA values (this is useful in multiple imputation). More complex imputations can be done with the `transcan` function, which also works with the generic methods shown here, i.e., `impute` can take a `transcan` object and use the imputed values created by `transcan` (with `imputed=TRUE`) to fill-in NAs. The `print` method places * after variable values that were imputed. The `summary` method summarizes all imputed values and then uses the next `summary` method available for the variable. The subscript method preserves attributes of the variable and subsets the list of imputed values corresponding with how the variable was subsetted. The `is.imputed` function is for checking if observations are imputed.

Usage

```
impute(x, ...)

## Default S3 method:
impute(x, fun=median, ...)

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

## S3 method for class 'impute':
summary(object, ...)
```

```
is.imputed(x)
```

Arguments

`x`	a vector or an object created by `transcan`, or a vector needing basic unconditional imputation. If there are no `NA`s and `x` is a vector, it is returned unchanged.
`fun`	the name of a function to use in computing the (single) imputed value from the non-NAs. The default is `median`. If instead of specifying a function as `fun`, a single value or vector (numeric, or character if `object` is a factor) is specified, those values are used for insertion. `fun` can also be the character string `"random"` to draw random values for imputation, with the random values not forced to be the same if there are multiple NAs. For a vector of constants, the vector must be of length one (indicating the same value replaces all NAs) or must be as long as the number of NAs, in which case the values correspond to consecutive NAs to replace. For a factor `object`, constants for imputation may include character values not in the current levels of `object`. In that case new levels are added. If `object` is of class `"factor"`, `fun` is ignored and the most frequent category is used for imputation.
`object`	an object of class `"impute"`
`...`	ignored

Value

a vector with class `"impute"` placed in front of existing classes. For `is.imputed`, a vector of logical values is returned (all `TRUE` if `object` is not of class `impute`).

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`transcan`, `impute.transcan`, `describe`, `na.include`, `sample`

Examples

```
age <- c(1,2,NA,4)
age.i <- impute(age)
# Could have used impute(age,2.5), impute(age,mean), impute(age,"random")
age.i
summary(age.i)
is.imputed(age.i)
```

`inc-dec` *Increment and Decrement*

Description

`inc<-` increments `x` by `value`. Equivelent to `x <- x + value`.

`dec<-` decrements `x` by the `value`. Equivelent to `x <- x - value`.

Usage

```
inc(x) <- value

dec(x) <- value
```

Arguments

`x`	object to be incremented or decremented
`value`	value by which `x` will be modified

Author(s)

Charles Dupont

Examples

```
x <- 1:5
inc(x) <- 5
x            # c(6,7,8,9,10)

dec(x) <- 3
x            # c(3,4,5,6,7)
```

`labcurve` *Label Curves, Make Keys, and Interactively Draw Points and Curves*

Description

`labcurve` Optionally draws a set of curves then labels the curves. A variety of methods for drawing labels are implemented, ranging from positioning using the mouse to automatic labeling to automatic placement of key symbols with manual placement of key legends to automatic placement of legends. For automatic positioning of labels or keys, a curve is labeled at a point that is maximally separated from all of the other curves. Gaps occurring when curves do not start or end at the same x-coordinates are given preference for positioning labels. If labels are offset from the curves (the default behaviour), if the closest curve to curve i is above curve i, curve i is labeled below its line. If the closest curve is below curve i, curve i is labeled above its line. These directions are reversed if the resulting labels would appear outside the plot region.

Both ordinary lines and step functions are handled, and there is an option to draw the labels at the same angle as the curve within a local window.

Unless the mouse is used to position labels or plotting symbols are placed along the curves to distinguish them, curves are examined at 100 (by default) equally spaced points over the range of x-coordinates in the current plot area. Linear interpolation is used to get y-coordinates to line up (step function or constant interpolation is used for step functions). There is an option to instead examine all curves at the set of unique x-coordinates found by unioning the x-coordinates of all the curves. This option is especially useful when plotting step functions. By setting `adj="auto"` you can have `labcurve` try to optimally left- or right-justify labels depending on the slope of the curves at the points at which labels would be centered (plus a vertical offset). This is especially useful when labels must be placed on steep curve sections.

You can use the `on top` method to write (short) curve names directly on the curves (centered on the y-coordinate). This is especially useful when there are many curves whose full labels would run into each other. You can plot letters or numbers on the curves, for example (using the `keys` option), and have `labcurve` use the `key` function to provide long labels for these short ones (see the end of the example). There is another option for connecting labels to curves using arrows. When `keys` is a vector of integers, it is taken to represent plotting symbols (`pch`s), and these symbols are plotted at equally-spaced x-coordinates on each curve (by default, using 5 points per curve). The points are offset in the x-direction between curves so as to minimize the chance of collisions.

To add a legend defining line types, colors, or line widths with no symbols, specify `keys="lines"`, e.g., `labcurve(curves, keys="lines", lty=1:2)`.

`putKey` provides a different way to use `key()` by allowing the user to specify vectors for labels, line types, plotting characters, etc. Elements that do not apply (e.g., `pch` for lines (`type="l"`)) may be `NA`. When a series of points is represented by both a symbol and a line, the corresponding elements of both `pch` and `lty`, `col.`, or `lwd` will be non-missing.

`putKeyEmpty`, given vectors of all the x-y coordinates that have been plotted, uses `largest.empty` to find the largest empty rectangle large enough to hold the key, and draws the key using `putKey`.

`drawPlot` is a simple mouse-driven function for drawing series of lines, step functions, polynomials, Bezier curves, and points, and automatically labeling the point groups using `labcurve` or `putKeyEmpty`. When `drawPlot` is invoked it creates temporary functions `Points`, `Curve`, and `Abline` in the session frame (frame zero). The user calls these functions inside the call to `drawPlot` to define groups of points in the order they are defined with the mouse. `Abline` is used to call `abline` and not actually great a group of points. For some curve types, the curve generated to represent the corresponding series of points is drawn after all points are entered for that series, and this curve may be different than the simple curve obtained by connecting points at the mouse clicks. For example, to draw a general smooth Bezier curve the user need only click on a few points, and she must overshoot the final curve coordinates to define the curve. The originally entered points are not erased once the curve is drawn. The same goes for step functions and polynomials. If you `plot()` the object returned by `drawPlot`, however, only final curves will be shown. The last examples show how to use `drawPlot`.

The `largest.empty` function finds the largest rectangle that is large enough to hold a rectangle of a given height and width, such that the rectangle does not contain any of a given set of points. This is used by `labcurve` and `putKeyEmpty` to position keys at the most empty part of an existing plot.

Usage

```
labcurve(curves, labels=names(curves),
         method=NULL, keys=NULL, keyloc=c("auto","none"),
         type="l", step.type=c("left", "right"),
         xmethod=if(any(type=="s")) "unique" else "grid",
         offset=NULL, xlim=NULL,
         tilt=FALSE, window=NULL, npts=100, cex=NULL,
         adj="auto", angle.adj.auto=30,
         lty=pr$lty, lwd=pr$lwd, col.=pr$col, transparent=TRUE,
         arrow.factor=1, point.inc=NULL, opts=NULL, key.opts=NULL,
         empty.method=c('area','maxdim'), numbins=25,
         pl=!missing(add), add=FALSE,
         ylim=NULL, xlab="", ylab="",
         whichLabel=1:length(curves),
         grid=FALSE, xrestrict=NULL, ...)

putKey(z, labels, type, pch, lty, lwd,
       cex=par('cex'), col=rep(par('col'),nc),
       transparent=TRUE, plot=TRUE, key.opts=NULL, grid=FALSE)

putKeyEmpty(x, y, labels, type=NULL,
            pch=NULL, lty=NULL, lwd=NULL,
            cex=par('cex'), col=rep(par('col'),nc),
            transparent=TRUE, plot=TRUE, key.opts=NULL,
            empty.method=c('area','maxdim'),
            numbins=25,
            xlim=pr$usr[1:2], ylim=pr$usr[3:4], grid=FALSE)

drawPlot(..., xlim=c(0,1), ylim=c(0,1), xlab='', ylab='',
         ticks=c('none','x','y','xy'),
         key=FALSE, opts=NULL)

# Points(label=' ', type=c('p','r'),
#        n, pch=pch.to.use[1], cex=par('cex'),
#        rug = c('none','x','y','xy'), ymean)

# Curve(label=' ',
#       type=c('bezier','polygon','linear','pol','step','gauss'),
#       n=NULL, lty=1, lwd=par('lwd'), degree=2,
#      evaluation=100, ask=FALSE)

# Abline(...)

## S3 method for class 'drawPlot':
plot(x, file, xlab, ylab, ticks,
     key=x$key, keyloc=x$keyloc, ...)

largest.empty(x, y, width, height,
              numbins=25, method=c('area','maxdim'),
              xlim=pr$usr[1:2], ylim=pr$usr[3:4],
              pl=FALSE, grid=FALSE)
```

Arguments

`curves`	a list of lists, each of which have at least two components: a vector of `x` values and a vector of corresponding `y` values. `curves` is mandatory except when `method="mouse"` or `"locator"`, in which case `labels` is mandatory. Each list in `curves` may optionally have any of the parameters `type`, `lty`, `lwd`, or `col` for that curve, as defined below (see one of the last examples).
`z`	a two-element list specifying the coordinate of the center of the key, e.g. `locator(1)` to use the mouse for positioning
`labels`	For `labcurve`, a vector of character strings used to label curves (which may contain newline characters to stack labels vertically). The default labels are taken from the names of the `curves` list. Setting `labels=FALSE` will suppress drawing any labels (for `labcurve` only). For `putKey` and `putKeyEmpty` is a vector of character strings specifying group labels
`x`	
`y`	for `putKeyEmpty` and `largest.empty`, `x` and `y` are same-length vectors specifying points that have been plotted. `x` can also be an object created by `drawPlot`.
`...`	For `drawPlot` is a series of invocations of `Points` and `Curve` (see example). Any number of point groups can be defined in this way. For `Abline` these may be any arguments to `abline`. For `labcurve`, other parameters to pass to `text`. For `plot.drawPlot` other parameters to pass to `setps`.
`width`	
`height`	for `largest.empty`, specifies the minimum allowable width in `x` units and the minimum allowable height in `y` units
`method`	`"offset"` (the default) offsets labels at largest gaps between curves, and draws labels beside curves. `"on top"` draws labels on top of the curves (especially good when using keys). `"arrow"` draws arrows connecting labels to the curves. `"mouse"` or `"locator"` positions labels according to mouse clicks. If `keys` is specified and is an integer vector or is `"lines"`, `method` defaults to `"on top"`. If `keys` is character, `method` defaults to `"offset"`. Set `method="none"` to suppress all curve labeling and key drawing, which is useful when `pl=TRUE` and you only need `labcurve` to draw the curves and the rest of the basic graph. For `largest.empty` specifies the method determining the best rectangle among all those that qualify with respect to `width` and `height`. Use `method="area"` (the default) to find the one having the largest area, or `method="maxdim"` to use the last rectangle searched that had both the largest width and largest height over all previous rectangles.
`keys`	This causes keys (symbols or short text) to be drawn on or beside curves, and if `keyloc` is not equal to `"none"`, a legend to be automatically drawn. The legend links keys with full curve labels and optionally with colors and line types.

	Set `keys` to a vector of character strings, or a vector of integers specifying plotting character (`pch` values - see `points`). For the latter case, the default behavior is to plot the symbols periodically, at equally spaced x-coordinates.
`keyloc`	When `keys` is specified, `keyloc` specifies how the legend is to be positioned for drawing using the `key` function in `trellis`. The default is `"auto"`, for which the `largest.empty` function to used to find the most empty part of the plot. If no empty rectangle large enough to hold the key is found, no key will be drawn. Specify `keyloc="none"` to suppress drawing a legend, or set `keyloc` to a 2-element list containing the x and y coordinates for the center of the legend. For example, use `keyloc=locator(1)` to click the mouse at the center. `keyloc` specifies the coordinates of the center of the key to be drawn with `plot.drawPlot` when `key=TRUE`.
`type`	for `labcurve`, a scalar or vector of character strings specifying the method that the points in the curves were connected. `"l"` means ordinary connections between points and `"s"` means step functions. For `putKey` and `putKeyEmpty` is a vector of plotting types, `"l"` for regular line, `"p"` for point, `"b"` for both point and line, and `"n"` for none. For `Points` is either `"p"` (the default) for regular points, or `"r"` for rugplot (one-dimensional scatter diagram to be drawn using the `scat1d` function). For `Curve`, `type` is `"bezier"` (the default) for drawing a smooth Bezier curves (which can represent a non-1-to-1 function such as a circle), `"polygon"` for orginary line segments, `"linear"` for a straight line defined by two endpoints, `"pol"` for a `degree`-degree polynomial to be fitted to the mouse-clicked points, `"step"` for a left-step-function, `"gauss"` to plot a Gaussian density fitted to 3 clicked points, or a function to draw a user-specified function, evaluated at `evaluation` points spanning the whole x-axis. For the density the user must click in the left tail, at the highest value (at the mean), and in the right tail, with the two tail values being approximately equidistant from the mean. The density is scaled to fit in the highest value regardless of its area.
`step.type`	type of step functions used (default is `"left"`)
`xmethod`	method for generating the unique set of x-coordinates to examine (see above). Default is `"grid"` for `type="l"` or `"unique"` for `type="s"`.
`offset`	distance in y-units between the center of the label and the line being labeled. Default is 0.75 times the height of an "m" that would be drawn in a label. For R grid/lattice you must specify offset using the `grid unit` function, e.g., `offset=unit(2,"native")` or `offset=unit(.25,"cm")` (`"native"` means data units)
`xlim`	limits for searching for label positions, and is also used to set up plots when `pl=TRUE` and `add=FALSE`. Default is total x-axis range for current plot (`par("usr")[1:2]`). For `largest.empty`, `xlim` limits the search for largest rectanges, but it has the same default as above. For `pl=TRUE,add=FALSE` you may want to extend `xlim` somewhat to allow large keys to fit, when using `keyloc="auto"`. For `drawPlot` default is `c(0,1)`.
`tilt`	set to `TRUE` to tilt labels to follow the curves, for `method="offset"` when `keys` is not given.
`window`	width of a window, in x-units, to use in determining the local slope for tilting

	labels. Default is 0.5 times number of characters in the label times the x-width of an "m" in the current character size and font.
`npts`	number of points to use if `xmethod="grid"`
`cex`	character size to pass to `text` and `key`. Default is current `par("cex")`. For `putKey`, `putKeyEmpty`, and `Points` is the size of the plotting symbol.
`adj`	Default is `"auto"` which has `labcurve` figure justification automatically when `method="offset"`. This will cause centering to be used when the local angle of the curve is less than `angle.adj.auto` in absolute value, left justification if the angle is larger and either the label is under a curve of positive slope or over a curve of negative slope, and right justification otherwise. For step functions, left justification is used when the label is above the curve and right justifcation otherwise. Set `adj=.5` to center labels at computed coordinates. Set to 0 for left-justification, 1 for right. Set `adj` to a vector to vary adjustments over the curves.
`angle.adj.auto`	see `adj`. Does not apply to step functions.
`lty`	vector of line types which were used to draw the curves. This is only used when keys are drawn. If all of the line types, line widths, and line colors are the same, lines are not drawn in the key.
`lwd`	vector of line widths which were used to draw the curves. This is only used when keys are drawn. See `lty` also.
`col.`	
`col`	vector of integer color numbers for use in curve labels, symbols, lines, and legends. Default is `par("col")` for all curves. See `lty` also.
`transparent`	Default is `TRUE` to make `key` draw transparent legends, i.e., to suppress drawing a solid rectangle background for the legend. Set to `FALSE` otherwise.
`arrow.factor`	factor by which to multiply default arrow lengths
`point.inc`	When `keys` is a vector of integers, `point.inc` specifies the x-increment between the point symbols that are overlaid periodically on the curves. By default, `point.inc` is equal to the range for the x-axis divided by 5.
`opts`	an optional list which can be used to specify any of the options to `labcurve`, with the usual element name abbreviations allowed. This is useful when `labcurve` is being called from another function. Example: `opts=list(method="arrow", cex=.8, np=200)`. For `drawPlot` a list of `labcurve` options to pass as `labcurve(..., opts=)`.
`key.opts`	a list of extra arguments you wish to pass to `key()`, e.g., `key.opts=list(background=1, between=3)`. The argument names must be spelled out in full.
`empty.method`	
`numbins`	These two arguments are passed to the `largest.empty` function's `method` and `numbins` arguments (see below). For `largest.empty` specifies the number of bins in which to discretize both the `x` and `y` directions for searching for rectangles. Default is 25.
`pl`	set to `TRUE` (or specify `add`) to cause the curves in `curves` to be drawn, under the control of `type`,`lty`,`lwd`,`col` parameters defined either in the `curves`

	lists or in the separate arguments given to `labcurve` or through `opts`. For `largest.empty`, set `pl=TRUE` to show the rectangle the function found by drawing it with a solid color.
`add`	By default, when curves are actually drawn by `labcurve` a new plot is started. To add to an existing plot, set `add=TRUE`.
`ylim`	When a plot has already been started, `ylim` defaults to `par("usr")[3:4]`. When `pl=TRUE`, `ylim` and `xlim` are determined from the ranges of the data. Specify `ylim` yourself to take control of the plot construction. In some cases it is advisable to make `ylim` larger than usual to allow for automatically-positioned keys. For `largest.empty`, `ylim` specifies the limits on the y-axis to limit the search for rectangle. Here `ylim` defaults to the same as above, i.e., the range of the y-axis of an open plot from `par`. For `drawPlot` the default is `c(0,1)`.
`xlab`	
`ylab`	x-axis and y-axis labels when `pl=TRUE` and `add=FALSE` or for `drawPlot`. Defaults to `""` unless the first curve has names for its first two elements, in which case the names of these elements are taken as `xlab` and `ylab`.
`whichLabel`	integer vector corresponding to `curves` specifying which curves are to be labelled or have a legend
`grid`	set to `TRUE` if the R `grid` package was used to draw the current plot. This prevents `labcurve` from using `par("usr")` etc. If using R `grid` you can pass coordinates and lengths having arbitrary units, as documented in the `unit` function. This is especially useful for `offset`.
`xrestrict`	When having `labcurve` label curves where they are most separated, you can restrict the search for this separation point to a range of the x-axis, specified as a 2-vector `xrestrict`. This is useful when one part of the curve is very steep. Even though steep regions may have maximum separation, the labels will collide when curves are steep.
`pch`	vector of plotting characters for `putKey` and `putKeyEmpty`. Can be any value including `NA` when only a line is used to indentify the group. Is a single plotting character for `Points`, with the default being the next unused value from among 1, 2, 3, 4, 16, 17, 5, 6, 15, 18, 19.
`file`	a file name suffix. If specified, `plot.drawPlot` will send its output to a postscript file "file.ps" using the `setps` function to get nice defaults for inclusion in reports.
`plot`	set to `FALSE` to keep `putKey` or `putKeyEmpty` from actually drawing the key. Instead, the size of the key will be return by `putKey`, or the coordinates of the key by `putKeyEmpty`.
`ticks`	tells `drawPlot` which axes to draw tick marks and tick labels. Default is `"none"`.
`key`	for `drawPlot` and `plot.drawPlot`. Default is `FALSE` so that `labcurve` is used to label points or curves. Set to `TRUE` to use `putKeyEmpty`.

Details

The internal functions `Points`, `Curve`, `Abline` have unique arguments as follows.

label: for `Points` and `Curve` is a single character string to label that group of points

n: number of points to accept from the mouse. Default is to input points until a right mouse click.

rug: for `Points`. Default is `"none"` to not show the marginal x or y distributions as rug plots, for the points entered. Other possibilities are used to execute `scat1d` to show the marginal distribution of x, y, or both as rug plots.

ymean: for `Points`, subtracts a constant from each y-coordinate entered to make the overall mean `ymean`

degree: degree of polynomial to fit to points by `Curve`

evaluation: number of points at which to evaluate Bezier curves, polynomials, and other functions in `Curve`

ask: set `ask=TRUE` to give the user the opportunity to try again at specifying points for Bezier curves, step functions, and polynomials

The `labcurve` function used some code from the function `plot.multicurve` written by Rod Tjoelker of The Boeing Company (`tjoelker@espresso.rt.cs.boeing.com`).

If there is only one curve, a label is placed at the middle x-value, and no fancy features such as `angle` or positive/negative offsets are used.

`key` is called once (with the argument `plot=FALSE`) to find the key dimensions. Then an empty rectangle with at least these dimensions is searched for using `largest.empty`. Then `key` is called again to draw the key there, using the argument `corner=c(.5,.5)` so that the center of the rectangle can be specified to `key`.

If you want to plot the data, an easier way to use `labcurve` is through `xYplot` as shown in some of its examples.

Value

`labcurve` returns an invisible list with components `x, y, offset, adj, cex, col`, and if `tilt=TRUE`, `angle`. `offset` is the amount to add to `y` to draw a label. `offset` is negative if the label is drawn below the line. `adj` is a vector containing the values 0, .5, 1.

`largest.empty` returns a list with elements `x` and `y` specifying the coordinates of the center of the rectangle which was found.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`approx`, `text`, `legend`, `setps`, `scat1d`, `xYplot`, `abline`

Examples

```
n <- 2:8
m <-  length(n)
type <- c('l','l','l','l','s','l','l')
# s=step function l=ordinary line (polygon)
curves <- vector('list', m)

plot(0,1,xlim=c(0,1),ylim=c(-2.5,4),type='n')

set.seed(39)

for(i in 1:m) {
  x <- sort(runif(n[i]))
  y <- rnorm(n[i])
  lines(x, y, lty=i, type=type[i], col=i)
  curves[[i]] <- list(x=x,y=y)
}

labels <- paste('Label for',letters[1:m])
labcurve(curves, labels, tilt=TRUE, type=type, col=1:m)

# Put only single letters on curves at points of
# maximum space, and use key() to define the letters,
# with automatic positioning of the key in the most empty
# part of the plot
# Have labcurve do the plotting, leaving extra space for key

names(curves) <- labels
labcurve(curves, keys=letters[1:m], type=type, col=1:m,
         pl=TRUE, ylim=c(-2.5,4))

# Put plotting symbols at equally-spaced points,
# with a key for the symbols, ignoring line types

labcurve(curves, keys=1:m, lty=1, type=type, col=1:m,
         pl=TRUE, ylim=c(-2.5,4))



# Plot and label two curves, with line parameters specified with data
set.seed(191)
ages.f <- sort(rnorm(50,20,7))
ages.m <- sort(rnorm(40,19,7))
height.f <- pmin(ages.f,21)*.2+60
height.m <- pmin(ages.m,21)*.16+63

labcurve(list(Female=list(ages.f,height.f,col=2),
              Male  =list(ages.m,height.m,col=3,lty='dashed')),
         xlab='Age', ylab='Height', pl=TRUE)
# add ,keys=c('f','m') to label curves with single letters
# For S-Plus use lty=2
# Plot power for testing two proportions vs. n for various odds ratios,
# using 0.1 as the probability of the event in the control group.
# A separate curve is plotted for each odds ratio, and the curves are
# labeled at points of maximum separation

n  <- seq(10, 1000, by=10)
OR <- seq(.2,.9,by=.1)
pow <- lapply(OR, function(or,n)list(x=n,y=bpower(p1=.1,odds.ratio=or,n=n)),
              n=n)
names(pow) <- format(OR)
labcurve(pow, pl=TRUE, xlab='n', ylab='Power')

# Plot some random data and find the largest empty rectangle
# that is at least .1 wide and .1 tall

x <- runif(50)
y <- runif(50)
plot(x, y)
z <- largest.empty(x, y, .1, .1)
z
points(z,pch=3)  # mark center of rectangle, or
#key(z$x, z$y, ... stuff for legend)



# Use the mouse to draw a series of points using one symbol, and
# two smooth curves or straight lines (if two points are clicked),
# none of these being labeled

# d <- drawPlot(Points(), Curve(), Curve())
# plot(d, file='/tmp/z')  # send result to /tmp/z.ps

## Not run:
# Use the mouse to draw a Gaussian density, two series of points
# using 2 symbols, one Bezier curve, a step function, and raw data
# along the x-axis as a 1-d scatter plot (rug plot).  Draw a key.
# The density function is fit to 3 mouse clicks
# Abline draws a dotted horizontal reference line
d <- drawPlot(Curve('Normal',type='gauss'),
              Points('female'), Points('male'),
              Curve('smooth',ask=TRUE,lty=2), Curve('step',type='s',lty=3),
              Points(type='r'), Abline(h=.5, lty=2),
              xlab='X', ylab='y', xlim=c(0,100), key=TRUE)
plot(d, ylab='Y')
plot(d, key=FALSE)  # label groups using labcurve
## End(Not run)
```

`label` *Label Attribute of an Object*

Description

`label(x)` retrieves the `label` attribute of `x`. `label(x) <- "a label"` stores the label attribute, and also puts the class `labelled` as the first class of `x` (for S-Plus 5 and later this class is not used and methods for handling this class are not defined so the `"label"` and `"units"` attributes are lost upon subsetting). The reason for having this class is so that the subscripting method for `labelled`, `[.labelled`, can preserve the `label` attribute in R and S-Plus 2000. Also, the `print` method for `labelled` objects prefaces the print with the object's `label` (and `units` if there). If the variable is also given a `"units"` attribute using the `units` function, subsetting the variable (using `[.labelled`) will also retain the `"units"` attribute.

`label` can optionally append a `"units"` attribute to the string, and it can optionally return a string or expression (for R's `plotmath` facility) suitable for plotting. `labelPlotmath` is a function that also has this function, when the input arguments are the `'label'` and `'units'` rather than a vector having those attributes. When `plotmath` mode is used to construct labels, the `'label'` or `'units'` may contain math expressions but they are typed verbatim if they contain percent signs, blanks, or underscores.

`Label` (actually `Label.data.frame`) is a function which generates S-Plus source code that makes the labels in all the variables in a data frame easy to edit.

`llist` is like `list` except that it preserves the names or labels of the component variables in the variables `label` attribute. This can be useful when looping over variables or using `sapply` or `lapply`. By using `llist` instead of `list` one can annotate the output with the current variable's name or label. `llist` also defines a `names` attribute for the list and pulls the `names` from the arguments' expressions for non-named arguments.

`plotmathTranslate` is a simple function that translates certain character strings to character strings that can be used as part of R `plotmath` expressions. If the input string has a space or percent inside, the string is surrounded by a call to `plotmath`'s `paste` function.

`as.data.frame.labelled` is a utility function that is called by `[.data.frame`. It is just a copy of `as.data.frame.vector`. `data.frame.labelled` is another utility function, that adds a class `"labelled"` to every variable in a data frame that has a `"label"` attribute but not a `"labelled"` class.

`reLabelled` is used to add a `'labelled'` class back to variables in data frame that have a 'label' attribute but no 'labelled' oldClass. Useful for changing `cleanup.import()`'d S-Plus 6 data frames back to general form for R and S-Plus 2000.

Usage

```
label(x, ...)

## Default S3 method:
label(x, units=FALSE, plot=FALSE, default=NULL,
      grid=FALSE, ...)

label(x) <- value

## S3 replacement method for class 'default':
label(x) <- value
```

```
labelPlotmath(label, units=NULL, plotmath=.R., grid=FALSE)

## S3 method for class 'labelled':
print(x, ...)   ## or x - calls print.labelled

Label(object, ...)

## S3 method for class 'data.frame':
Label(object, file='', append=FALSE, ...)

llist(..., labels=TRUE)

plotmathTranslate(x)

data.frame.labelled(object)

reLabelled(object)
```

Arguments

`x`	any object (for `plotmathTranslate` is a character string)
`units`	set to `TRUE` to append the `'units'` attribute (if present) to the returned label. The `'units'` are surrounded by brackets. For `labelPlotmath` is a character string containing the units of measurement.
`plot`	set to `TRUE` to return a label suitable for R's `plotmath` facility (returns an expression instead of a character string) if R is in effect. If `units` is also `TRUE`, and if both `'label'` and `'units'` attributes are present, the `'units'` will appear after the label but in smaller type and will not be surrounded by brackets.
`default`	if `x` does not have a `'label'` attribute and `default` (a character string) is specified, the label will be taken as `default`
`grid`	Currently R's `lattice` and `grid` functions do not support `plotmath` expressions for `xlab` and `ylab` arguments. When using `lattice` functions in R, set the argument `grid` to `TRUE` so that `labelPlotmath` can return an ordinary character string instead of an expression.
`label`	a character string containing a variable's label
`plotmath`	set to `TRUE` to have `labelMathplot` return an expression for plotting using R's `plotmath` facility. If R is not in effect, an ordinary character string is returned.
`value`	the label of the object, or "".
`object`	a data frame
`...`	a list of variables or expressions to be formed into a `list`. Ignored for `print.labelled`.
`file`	the name of a file to which to write S-Plus source code. Default is `""`, meaning standard output.
`append`	set to `TRUE` to append code generated by `Label` to file `file`
`labels`	set to `FALSE` to make `llist` ignore the variables' `label` attribute and use the variables' names.

Value

`label` returns the label attribute of x, if any; otherwise, "". `label` is used most often for the individual variables in data frames. The function `sas.get` copies labels over from SAS if they exist.

See Also

`sas.get`, `describe`

Examples

```
age <- c(21,65,43)
y   <- 1:3
label(age) <- "Age in Years"
plot(age, y, xlab=label(age))

x1 <- 1:10
x2 <- 10:1
label(x2) <- 'Label for x2'
units(x2) <- 'mmHg'
x2
x2[1:5]
dframe <- data.frame(x1, x2)
Label(dframe)

##In these examples of llist, note that labels are printed after
##variable names, because of print.labelled
a <- 1:3
b <- 4:6
label(b) <- 'B Label'
llist(a,b)
llist(a,b,d=0)
llist(a,b,0)

w <- llist(a, b>5, d=101:103)
sapply(w, function(x){
  hist(as.numeric(x), xlab=label(x))
  # locator(1)   ## wait for mouse click
})

# Or: for(u in w) {hist(u); title(label(u))}
```

`latex` *Convert an S object to LaTeX, and Related Utilities*

Description

`latex` converts its argument to a `.tex` file appropriate for inclusion in a LaTeX2e document. `latex` is a generic function that calls one of `latex.default`, `latex.function`, `latex.list`.

latex.default does appropriate rounding and decimal alignment and produces a file containing a LaTeX tabular environment to print the matrix or data.frame x as a table.

latex.function prepares an S function for printing by issuing sed commands that are similar to those in the S.to.latex procedure in the s.to.latex package (Chambers and Hastie, 1993).

latex.list calls latex recursively for each element in the argument.

latexTranslate translates particular items in character strings to LaTeX format, e.g., makes a^2 = a$^2$ for superscript within variable labels. LaTeX names of greek letters (e.g., "alpha") will have backslashes added if greek==TRUE. Math mode is inserted as needed. latexTranslate assumes that input text always has matches, e.g. [) [] (] (), and that surrounding by $$ is OK.

latexSN converts a vector floating point numbers to character strings using LaTeX exponents. Dollar signs to enter math mode are not added.

latexVerbatim on an object executes the object's print method, capturing the output for a file inside a LaTeX verbatim environment.

dvi uses the system latex command to compile LaTeX code produced by latex, including any needed styles. dvi will put a documentclass{report} and end{document} wrapper around a file produced by latex. By default, the geometry LaTeX package is used to omit all margins and to set the paper size to a default of 5.5in wide by 7in tall. The result of dvi is a .dvi file. To both format and screen display a non-default size, use for example print(dvi(latex(x), width=3, height=4),width=3,height=4). Note that you can use something like xdvi -geometry 460x650 -margins 2.25in file without changing LaTeX defaults to emulate this.

dvips will use the system dvips command to print the .dvi file to the default system printer, or create a postscript file if file is specified.

dvigv uses the system dvips command to convert the input object to a .dvi file, and uses the system dvips command to convert it to postscript. Then the postscript file is displayed using Ghostview (assumed to be the system command gv).

There are show methods for displaying typeset LaTeX on the screen using the system xdvi command. If you show a LaTeX file created by latex without running it through dvi using show.dvi(object), the show method will run it through dvi automatically. These show methods are not S Version 4 methods so you have to use full names such as show.dvi and show.latex. Use the print methods for more automatic display of typesetting, e.g. typing latex(x) will invoke xdvi to view the typeset document.

Usage

```
latex(object, title=first.word(deparse(substitute(object))), ...)

## Default S3 method:
latex(object,
    title=first.word(deparse(substitute(object))),
    file=paste(title, ".tex", sep=""),
    append=FALSE, label=title,
    rowlabel=title, rowlabel.just="l", cgroup=NULL, n.cgroup=NULL,
    rgroup=NULL, n.rgroup=NULL,
    cgroupTexCmd="bfseries",
    rgroupTexCmd="bfseries",
    rownamesTexCmd=NULL,
    colnamesTexCmd=NULL,
    cellTexCmds=NULL,
    rowname, cgroup.just=rep("c",length(n.cgroup)),
    colheads=dimnames(cx)[[2]],
    extracolheads=NULL, extracolsize='scriptsize',
    dcolumn=FALSE, numeric.dollar=!dcolumn, cdot=FALSE,
    longtable=FALSE, draft.longtable=TRUE, ctable=FALSE, booktabs=FALSE,
    table.env=TRUE, here=FALSE, lines.page=40,
    caption=NULL, caption.lot=NULL, caption.loc=c('top','bottom'),
    double.slash=FALSE,
    vbar=FALSE, collabel.just=rep("c",nc), na.blank=TRUE,
    insert.bottom=NULL, first.hline.double=!(booktabs | ctable),
    where='!tbp', size=NULL,
    center=c('center','centering','none'),
    landscape=FALSE,
    multicol=TRUE,
    math.row.names=FALSE, math.col.names=FALSE,
    ...) # x is a matrix or data.frame

## S3 method for class 'function':
latex(
        object,
        title=first.word(deparse(substitute(object))),
        file=paste(title, ".tex", sep=""),
        append=FALSE,
        assignment=TRUE,  type=c('example','verbatim'), ...)

## S3 method for class 'list':
latex(
           object,
           title=first.word(deparse(substitute(object))),
           file=paste(title, ".tex", sep=""),
           append=FALSE,
           label,
           caption,
           caption.lot,
           caption.loc=c('top','bottom'),
           ...)

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

latexTranslate(object, inn=NULL, out=NULL, pb=FALSE, greek=FALSE, ...)

latexSN(x)
```

```
latexVerbatim(x, title=first.word(deparse(substitute(x))),
    file=paste(title, ".tex", sep=""),
    append=FALSE, size=NULL, hspace=NULL,
    width=.Options$width, length=.Options$length, ...)

dvi(object, ...)
## S3 method for class 'latex':
dvi(object, prlog=FALSE, nomargins=TRUE, width=5.5, height=7, ...)
## S3 method for class 'dvi':
print(x, ...)
dvips(object, ...)
## S3 method for class 'latex':
dvips(object, ...)
## S3 method for class 'dvi':
dvips(object, file, ...)
## S3 method for class 'latex':
show(object)  # or show.dvi(object) or just object
dvigv(object, ...)
## S3 method for class 'latex':
dvigv(object, ...)       # or gvdvi(dvi(object))
## S3 method for class 'dvi':
dvigv(object, ...)
```

Arguments

`object`	For `latex`, any S object. For `dvi` or `dvigv`, an object created by `latex`. For `latexTranslate` is a vector of character strings to translate.
`x`	any object to be `print`ed verbatim for `latexVerbatim`. For `latexSN` `x` is a numeric vector.
`title`	name of file to create without the `.tex` extension. If this option is not set, value/string of `x` (see above) is printed in the top left corner of the table. Set `title=''` to suppress this output.
`file`	name of the file to create. The default file name is `x.tex` where `x` is the first word in the name of the argument for `x`. Set `file=""` to have the generated LaTeX code just printed to standard output. This is especially useful when running under Sweave in R using its `results=tex` tag, to save having to manage many small external files. When `file=""`, `latex` keeps track of LaTeX styles that are called for by creating or modifying an object `latexStyles` (in `.GlobalTemp` in R or in frame 0 in S-Plus). `latexStyles` is a vector containing the base names of all the unique LaTeX styles called for so far in the current session. See the end of the examples section for a way to use this object to good effect. For `dvips`, `file` is the name of an output postscript file.
`append`	defaults to `FALSE`. Set to `TRUE` to append output to an existing file.
`label`	a text string representing a symbolic label for the table for referencing in the LaTeX `\label` and `\ref` commands. `label` is only used if `caption` is given.

`rowlabel`	If `x` has row dimnames, `rowlabel` is a character string containing the column heading for the row dimnames. The default is the name of the argument for `x`.
`rowlabel.just`	If `x` has row dimnames, specifies the justification for printing them. Possible values are `"l"`, `"r"`, `"c"`. The heading (`rowlabel`) itself is left justified if `rowlabel.just="l"`, otherwise it is centered.
`cgroup`	a vector of character strings defining major column headings. The default is to have none.
`n.cgroup`	a vector containing the number of columns for which each element in cgroup is a heading. For example, specify `cgroup=c("Major 1","Major 2")`, `n.cgroup=c(3,3)` if `"Major 1"` is to span columns 1-3 and `"Major 2"` is to span columns 4-6. `rowlabel` does not count in the column numbers. You can omit `n.cgroup` if all groups have the same number of columns.
`rgroup`	a vector of character strings containing headings for row groups. `n.rgroup` must be present when `rgroup` is given. The first `n.rgroup[1]` rows are sectioned off and `rgroup[1]` is used as a bold heading for them. The usual row dimnames (which must be present if `rgroup` is) are indented. The next `n.rgroup[2]` rows are treated likewise, etc.
`n.rgroup`	integer vector giving the number of rows in each grouping. If `rgroup` is not specified, `n.rgroup` is just used to divide off blocks of rows by horizontal lines. If `rgroup` is given but `n.rgroup` is omitted, `n.rgroup` will default so that each row group contains the same number of rows.
`cgroupTexCmd`	A character string specifying a LaTeX command to be used to format column group labels. The default, "bfseries", sets the current font to 'bold'. It is possible to supply a vector of strings so that each column group label is formatted differently. Please note that the first item of the vector is used to format the title (even if a title is not used), and that there is an empty column between each column group. Currently the user needs to handle these issues. Multiple effects can be achieved by creating custom LaTeX commands; for example, `\providecommand{\redscshape}{\color{red}\scshape}` creates a LaTeX command called `redscshape` that formats the text in red small-caps.
`rgroupTexCmd`	A character string specifying a LaTeX command to be used to format row group labels. The default, "bfseries", sets the current font to 'bold'. A vector of strings can be supplied to format each row group label differently. Normal recycling applies if the vector is shorter than `n.rgroups`. See also `cgroupTexCmd` above regarding multiple effects.
`rownamesTexCmd`	A character string specifying a LaTeX command to be used to format rownames. The default, NULL, applies no command. A vector of different commands can also be supplied. See also `cgroupTexCmd` above regarding multiple effects.
`colnamesTexCmd`	A character string specifying a LaTeX command to be used to format column labels. The default, NULL, applies no command. It is possible to supply a vector of strings to format each column label differently. If column groups are not used, the first item in the vector will be used to format the title. Please note that if column groups are used the first item of `cgroupTexCmd` and not

	`colnamesTexCmd` is used to format the title, and that there are empty columns between each group. The user needs to allow for these issues when supplying a vector of commands. See also `cgroupTexCmd` above regarding multiple effects.
`cellTexCmds`	A matrix of character strings which are LaTeX commands to be used to format each element, or cell, of the object. The matrix must have the same NROW() and NCOL() as the object. The default, NULL, applies no formats. Empty strings also apply no formats, and one way to start might be to create a matrix of empty strings with `matrix(rep("", NROW(x) * NCOL(x)), nrow=NROW(x))` and then selectively change appropriate elements of the matrix. Note that you might need to set `numeric.dollar=FALSE` (to disable math mode) for some effects to work. See also `cgroupTexCmd` above regarding multiple effects.
`na.blank`	Set to `TRUE` to use blanks rather than `NA` for missing values. This usually looks better in `latex`.
`insert.bottom`	an optional character string to typeset at the bottom of the table. For `"ctable"` style tables, this is placed in an unmarked footnote.
`first.hline.double`	set to `FALSE` to use single horizontal rules for styles other than `"bookmark"` or `"ctable"`
`rowname`	rownames for `tabular` environment. Default is rownames of matrix or data.frame. Specify `rowname=NULL` to suppress the use of row names.
`cgroup.just`	justification for labels for column groups. Defaults to `"c"`.
`colheads`	a character vector of column headings if you don't want to use `dimnames(object)[[2]]`. Specify `colheads=NULL` to suppress column headings.
`extracolheads`	an optional vector of extra column headings that will appear under the main headings (e.g., sample sizes). This character vector does not need to include an empty space for any `rowname` in effect, as this will be added automatically. You can also form subheadings by splitting character strings defining the column headings using the usual backslash `n` newline character.
`extracolsize`	size for `extracolheads` or for any second lines in column names; default is `"scriptsize"`
`dcolumn`	
`numeric.dollar`	logical, default `!dcolumn`. Set to `TRUE` to place dollar signs around numeric values when `dcolumn=FALSE`. This assures that `latex` will use minus signs rather than hyphens to indicate negative numbers. Set to `FALSE` when `dcolumn=TRUE`, as `dcolumn.sty` automatically uses minus signs.
`math.row.names`	logical, set true to place dollar signs around the row names.
`math.col.names`	logical, set true to place dollar signs around the column names.
`cdot`	see `format.df`

`longtable`	Set to `TRUE` to use David Carlisle's LaTeX `longtable` style, allowing long tables to be split over multiple pages with headers repeated on each page. The `"style"` element is set to `"longtable"`. The `latex \usepackage` must reference `[longtable]`. The file `longtable.sty` will need to be in a directory in your `$TEXINPUTS` path.
`draft.longtable`	I forgot what this does.
`ctable`	set to `TRUE` to use Wybo Dekker's `ctable` style from `CTAN`. Even though for historical reasons it is not the default, it is generally the preferred method. Thicker but not doubled `hline`s are used to start a table when `ctable` is in effect.
`booktabs`	set `booktabs=TRUE` to use the `booktabs` style of horizontal rules for better tables. In this case, double `hline`s are not used to start a table.
`table.env`	Set `table.env=FALSE` to suppress enclosing the table in a LaTeX `table` environment. `table.env` only applies when `longtable=FALSE`. You may not specify a `caption` if `table.env=FALSE`.
`here`	Set to `TRUE` if you are using `table.env=TRUE` with `longtable=FALSE` and you have installed David Carlisle's `here.sty` LaTeX style. This will cause the LaTeX `table` environment to be set up with option `H` to guarantee that the table will appear exactly where you think it will in the text. The `"style"` element is set to `"here"`. The `latex \usepackage` must reference `[here]`. The file `here.sty` will need to be in a directory in your `$TEXINPUTS` path. `here` is largely obsolete with LaTeX2e.
`lines.page`	Applies if `longtable=TRUE`. No more than `lines.page` lines in the body of a table will be placed on a single page. Page breaks will only occur at `rgroup` boundaries.
`caption`	a text string to use as a caption to print at the top of the first page of the table. Default is no caption.
`caption.lot`	a text string representing a short caption to be used in the "List of Tables". By default, LaTeX will use `caption`. If you get inexplicable `latex` errors, you may need to supply `caption.lot` to make the errors go away.
`caption.loc`	set to `"bottom"` to position a caption below the table instead of the default of `"top"`.
`double.slash`	set to `TRUE` to output `\` as `\\` in LaTeX commands. Useful when you are reading the output file back into an S vector for later output.
`vbar`	logical. When `vbar==TRUE`, columns in the tabular environment are separated with vertical bar characters. When `vbar==FALSE`, columns are separated with white space. The default, `vbar==FALSE`, produces tables consistent with the style sheet for the Journal of the American Statistical Association.
`collabel.just`	justification for column labels.
`assignment`	logical. When `TRUE`, the default, the name of the function and the assignment arrow are printed to the file.
`where`	specifies placement of floats if a table environment is used. Default is `"!tbp"`. To allow tables to appear in the middle of a page of text you might specify `where="!htbp"` to `latex.default`.

`size`	size of table text if a size change is needed (default is no change). For example you might specify `size="small"` to use LaTeX font size "small".
`center`	default is `"center"` to enclose the table in a `center` environment. Use `center="centering"` to instead use a LaTeX `centering` directive, or `center="none"` to use no centering. This option was implemented by Markus Jäntti ⟨markus.jantti@iki.fi⟩ of Abo Akademi University.
`landscape`	set to `TRUE` to enclose the table in a `landscape` environment. When `ctable` is `TRUE`, will use the `rotate` argument to `ctable`.
`type`	The default uses the S `Example` environment for `latex.function`, assuming you have installed `S.sty` in a location that the system latex command automatically accesses. Set `type="verbatim"` to instead use the LaTeX `verbatim` environment.
`...`	other arguments are accepted and ignored except that `latex` passes arguments to `format.df` (e.g., `col.just` and other formatting options like `dec`, `rdec`, and `cdec`). For `latexVerbatim` these arguments are passed to the `print` function. Ignored for `latexTranslate`.
`inn, out`	specify additional input and translated strings over the usual defaults
`pb`	If `pb=TRUE`, `latexTranslate` also translates `[()]` to math mode using `\left, \right`.
`greek`	set to `TRUE` to have `latexTranslate` put names for greek letters in math mode and add backslashes
`hspace`	horizontal space, e.g., extra left margin for verbatim text. Default is none. Use e.g. `hspace="10ex"` to add 10 extra spaces to the left of the text.
`length`	for S-Plus only; is the length of the output page for printing and capturing verbatim text
`width`	
`height`	are the `options( )` to have in effect only for when `print` is executed. Defaults are current `options`. For `dvi` these specify the paper width and height in inches if `nomargins=TRUE`, with defaults of 5.5 and 7, respectively.
`prlog`	set to `TRUE` to have `dvi` print, to the S-Plus session, the LaTeX .log file.
`multicol`	set to `FALSE` to not use `\multicolumn` in header of table
`nomargins`	set to `FALSE` to use default LaTeX margins when making the .dvi file

Details

If running under Windows and using MikTeX, `latex` and `yap` must be in your system path, and `yap` is used to browse `.dvi` files created by `latex`. You should install the `geometry` and `ctable` styles in MikTeX to make optimum use of `latex()`.

System options can be used to specify external commands to be used. Defaults are given by `options(xdvicmd='xdvi')` or `options(xdvicmd='yap')`, `options(dvipscmd='dvips')`, `options(latexcmd='latex')`. For MacOS specify `options(xdvicmd='MacdviX')`.

If running S-Plus and your directory for temporary files is not `/tmp` (Unix/Linux) or `\windows\temp` (Windows), add your own `tempdir` function such as `tempdir <- function() "/yourmaindirectory/your`

To prevent the latex file from being displayed store the result of `latex` in an object, e.g. `w <- latex(object, file='foo.tex')`.

Value

`latex` and `dvi` return a list of class `latex` or `dvi` containing character string elements `file` and `style`. `file` contains the name of the generated file, and `style` is a vector (possibly empty) of styles to be included using the LaTeX2e `\usepackage` command.

`latexTranslate` returns a vector of character strings

Side Effects

creates various system files and runs various Linux/UNIX system commands which are assumed to be in the system path.

Author(s)

Frank E. Harrell, Jr.,
Department of Biostatistics,
Vanderbilt University,
`f.harrell@vanderbilt.edu`

Richard M. Heiberger,
Department of Statistics,
Temple University, Philadelphia, PA.
`rmh@astro.ocis.temple.edu`

David R. Whiting,
School of Clinical Medical Sciences (Diabetes),
University of Newcastle upon Tyne, UK.
`david.whiting@ncl.ac.uk`

See Also

`html`, `format.df`, `texi2dvi`

Examples

```
x <- matrix(1:6, nrow=2, dimnames=list(c('a','b'),c('c','d','enLine 2')))
## Not run:
latex(x)   # creates x.tex in working directory
w <- latex(x, file='/tmp/my.tex')
d <- dvi(w)  # compile LaTeX document, make .dvi
             # latex assumed to be in path
d            # or show(d) : run xdvi (assumed in path) to display
w            # or show(w) : run dvi then xdvi
dvips(d)     # run dvips to print document
dvips(w)     # run dvi then dvips
library(tools)
texi2dvi('/tmp/my.tex')   # compile and produce pdf file in working dir.
## End(Not run)
latex(x, file="")   # just write out LaTeX code to screen

## Not run:
# After running latex( ) multiple times with different special styles in
# effect, make a file that will call for the needed LaTeX packages when
# latex is run (especially when using Sweave with R)
if(exists(latexStyles))
  cat(paste('\usepackage{',latexStyles,'}',sep=''),
      file='stylesused.tex', sep='\n')
# Then in the latex job have something like:
# \documentclass{article}
# \input{stylesused}
# \begin{document}
# ...
## End(Not run)
```

---

ldBands *Group Sequential Boundaries using the Lan-DeMets Approach*

Description

This function computes and plots group sequential stopping boundaries from the Lan-DeMets method with a variety of α-spending functions using the `ld98` program from the Department of Biostatistics, University of Wisconsin written by DM Reboussin, DL DeMets, KM Kim, and KKG Lan. Such stopping boundaries are useful for early termination of clinical trials for safety problems or for efficacy. Simple `plot` and `print` methods are implemented. Simple sample size and minimally detectable effect sizes given sample sizes may be obtained with a `summary` method if `power` was specified to `ldBands`. Alternatively, `summary` computes, for each look, the difference in means that must be achieved to cross a boundary if `n` and `sd` are specified, or the minimum difference in proportions and the odds ratios that must be achieved to cross a boundary if `n` and `pbar` are specified.

Usage

```
ldBands(n = length(times), times = NULL, alpha = 0.05, sided = 2,
        alphaLower=alpha/2, alphaUpper=alpha/2,
        information = NULL,
        spending=c('OBrien-Fleming','Pocock','alpha*t^phi',
                   'Hwang-Shih-DeCani'),
        phi=1,
        spending2=c('OBrien-Fleming','Pocock','alpha*t^phi',
                    'Hwang-Shih-DeCani'),
        phi2=phi,
        truncate = Inf, power = NULL, pr = TRUE)
## S3 method for class 'ldBands':
print(x, ...)
## S3 method for class 'ldBands':
plot(x, xlab='Time', ylab='Z', actual=NULL,
        type='b', labels=NULL, ...)

## S3 method for class 'ldBands':
summary(object, stdiff=NULL, n=NULL,
        p1=NULL, p2=NULL,  hr=NULL, events=NULL,
        pbar=NULL, sd=NULL, ...)
## S3 method for class 'summary.ldBands':
print(x, ...)
```

Arguments

`n`	number of interim analyses. If `times` is given, is automatically taken as the length of `times`. For `summary.ldBands`, `n` is the sample size, to obtain detectable standardized difference.
`times`	times at which interim analyses are done
`alpha`	overall α level for the multiple tests. Default is `0.05`. If `sided=3` is the α-level for the lower bounds, otherwise is the total α.
`sided`	set to `1` to use a one-sided test, `3` for asymmetric two-sided bounds
`alphaLower`	α-level for lower bound if `sided=3`. Defaults to $\alpha/2$. When `sided=3` `alpha` is recalculated from `alphaLower+alphaUpper`.
`alphaUpper`	α-level for upper bound if `sided=3`. Defaults to $\alpha/2$.
`information`	a vector of information times if different from `times`. Used for computing covariances of test statistics.
`spending`	an α spending function. Default is O'Brien-Fleming function. If `sided=3` is the spending function for the lower bounds.
`phi`	parameter for the third and fourth types of spending functions (exponent of time for the third, default is `1`)
`spending2`	spending function for the upper bounds if `sided=3`. Defaults to same spending function for lower bounds.
`phi2`	parameter for third and fourth spending functions if `sided=3`. Default is `phi`.
`truncate`	a value at which to truncate Z critical values so that early looks will have some power without really affecting the overall α level. Default is no truncation.
`power`	A power for which to compute a drift parameter; useful in sample size calculations
`pr`	set to `FALSE` to supress the actual output of the `ld98` program
`x`	an object created by `ldBands` or `summary.ldBands`
`xlab`	x-axis label
`ylab`	y-axis label
`actual`	an optional list with two components: times and z values to add as points to the first plot
`type`	default is `"b"` causing both points and lines to be drawn for boundaries. Use `type="l"` for example to suppress drawing circles at points.
`labels`	an optional character vector to be used to label time points corresponding to those generated by `ldBands`
`object`	an object created by `ldBands`
`stdiff`	standardized difference to detect

`p1`	probability of event in group 1
`p2`	probability of event in group 2, to compare with `p1` in order to obtain sample size for two-sample binomial
`hr`	hazard ratio to detect, to obtain needed number of events at end of study for either treatment arm using the logrank test
`events`	number of events per treatment arm at end of study, to obtain detectable hazard ratio
`pbar`	mean of actual probabilities of events in two treatment arms
`sd`	standard deviation of an observation
`...`	unused

Details

This function assumes that you have stored the `ld98` executable in a subdirectory that is in your system path. Obtain `ld98` program from the URL given in the reference below.

The `plot` method makes two plots if `power` is specified, the second containing exit and cumulative exit probabilities as a function of time. If `par(mfrow=c())` is not already set to something besides `c(1,1)`, `par(mfrow=c(2,1))` is set temporarily.

Value

A list of class `"ldBands"` with the following components. When `summary` is used and `n` and either `pbar` or `sd` are given, `summary` returns the same object returned by `ldBands` but with possible components `diff.lower,diff.lower` (needed difference means or proportions) and `or.lower,or.upper` (needed odds ratios).

`data`	a data frame containing the main calculations
`power`	power specified, if any
`drift`	drift calculated, if `power` specified
`type`	`"boundaries"` if `power` not specified, `"power"` otherwise
`n`	number of interim analyses, for `ldBands`. For `summary` is the number of observations per arm, and it must be a vector with length equal to the number of looks if `pbar` or `sd` are given.
`alpha`	input value of `alpha`
`sided`	`1-3`
`alphaLower`	lower α
`alphaUpper`	upper α
`spending`	name of α spending function used
`phi`	parameter for spending function
`spending2`	name of spending function for upper boundary. Defaults to `spending`
`phi2`	parameter for second spending function. Defaults to `phi` but is ignored if `spending2` is the first or second type of spending function.
`truncate`	value at which Z statistics truncated (default is `Inf`)

Author(s)

Frank E Harrell Jr

References

Reboussin DM, DeMets DL, Kim K-M, Lan KKG (1996): Programs for computing group sequential boundaries using the Lan-DeMets method. http://www.biostat.wisc.edu/landemets/

Reboussin DM, DeMets DL, Kim K, Lan KKG (2000): Computations for group sequential boundaries using the Lan-DeMets spending function method. Controlled Clinical Trials 21:190-207.

See Also

gbayes

Examples

```
## Not run:
# Get boundaries for O'Brien-Fleming spending function, 5 looks, alpha=.05
b <- ldBands(5, pr=FALSE)
plot(b)
# Same but times are irregular, and information times are different than
# test times.  Use Pocock spending function.
b <- ldBands(times=      c(.4, .6, .8, .9, .95),
             information=c(.42,.65,.83,.89,.94), spending='Pocock')

# Get power calculations
u <- ldBands(times=c(.4, .6, .8, .9, .95),  power=.9)
u$drift                   # standardize difference * sqrt(n per arm)
                          # needed to provide power=.9
summary(u, n=50)          # obtain detectable standardized difference
summary(u, p1=.4, p2=.5)  # get sample size per arm, two-sample binomial
summary(u, hr=1.5)        # get number of events per arm needed
                          # to detect a hazard ratio of 1.5

# Asymmetric boundaries with different spending functions, truncate
b <- ldBands(5, sided=3, spending='alpha*t^phi', phi=1, phi2=1.5,
             alphaLower=.01, alphaUpper=.04, truncate=4)
b
plot(b)
# Compute differences in proportions and odds ratios needed to cross
# the boundaries, given a mean probability in two treatment arms of 0.1
# and given a vector of sample sizes per arm corresponding to the looks
s <- summary(b, n=seq(200,1000,by=200), pbar=.1)
s
d <- s$data
plot(d$n, d$or.lower, xlab='N Per Arm',
     ylab='Critical Odds Ratio', type='b',
     ylim=range(d$or.lower, d$or.upper), log='y')
lines(d$n, d$or.upper, type='b')
abline(h=1, lty=2)
## End(Not run)
```

`list.tree` *Pretty-print the Structure of a Data Object*

Description

This is a function to pretty-print the structure of any data object (usually a list). It is similar to the R function `str`.

Usage

```
list.tree(struct, depth=-1, numbers=FALSE, maxlen=22, maxcomp=12,
          attr.print=TRUE, front="", fill=". ", name.of, size=TRUE)
```

Arguments

`struct`	The object to be displayed
`depth`	Maximum depth of recursion (of lists within lists . . .) to be printed; negative value means no limit on depth.
`numbers`	If TRUE, use numbers in leader instead of dots to represent position in structure.
`maxlen`	Approximate maximum length (in characters) allowed on each line to give the first few values of a vector. maxlen=0 suppresses printing any values.
`maxcomp`	Maximum number of components of any list that will be described.
`attr.print`	Logical flag, determining whether a description of attributes will be printed.
`front`	Front material of a line, for internal use.
`fill`	Fill character used for each level of indentation.
`name.of`	Name of object, for internal use (deparsed version of struct by default).
`size`	Logical flag, should the size of the object in bytes be printed?

A description of the structure of struct will be printed in outline form, with indentation for each level of recursion, showing the internal storage mode, length, class(es) if any, attributes, and first few elements of each data vector. By default each level of list recursion is indicated by a "." and attributes by "A".

Author(s)

Alan Zaslavsky, zaslavsk@hcp.med.harvard.edu

See Also

`str`

Examples

```
X <- list(a=ordered(c(1:30,30:1)),b=c("Rick","John","Allan"),
          c=diag(300),e=cbind(p=1008:1019,q=4))
list.tree(X)
# In R you can say str(X)
```

`mApply` *Apply a Function to Rows of a Matrix or Vector*

Description

`mApply` is like `tapply` except that the first argument can be a matrix or a vector, and the output is cleaned up if `simplify=TRUE`. It uses code adapted from Tony Plate (⟨tplate@blackmesacapital.com⟩) to operate on grouped submatrices.

As `mApply` can be much faster than using `by`, it is often worth the trouble of converting a data frame to a numeric matrix for processing by `mApply`. `asNumericMatrix` will do this, and `matrix2dataFrame` will convert a numeric matrix back into a data frame.

Usage

```
mApply(X, INDEX, FUN, ..., simplify=TRUE, keepmatrix=FALSE)
```

Arguments

`X`	a vector or matrix capable of being operated on by the function specified as the `FUN` argument
`INDEX`	list of factors, each of same number of rows as 'X' has.
`FUN`	the function to be applied. In the case of functions like '+', '
`...`	optional arguments to 'FUN'.
`simplify`	set to 'FALSE' to suppress simplification of the result in to an array, matrix, etc.
`keepmatrix`	set to `TRUE` to keep result as a matrix even if `simplify` is `TRUE`, in the case of only one stratum

Value

For `mApply`, the returned value is a vector, matrix, or list. If `FUN` returns more than one number, the result is an array if `simplify=TRUE` and is a list otherwise. If a matrix is returned, its rows correspond to unique combinations of `INDEX`. If `INDEX` is a list with more than one vector, `FUN` returns more than one number, and `simplify=FALSE`, the returned value is a list that is an array with the first dimension corresponding to the last vector in `INDEX`, the second dimension corresponding to the next to last vector in `INDEX`, etc., and the elements of the list-array correspond to the values computed by `FUN`. In this situation the returned value is a regular array if `simplify=TRUE`. The order of dimensions is as previously but the additional (last) dimension corresponds to values computed by `FUN`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`asNumericMatrix`, `matrix2dataFrame`, `tapply`, `sapply`, `lapply`, `mapply`, `by`.

Examples

```
require(datasets, TRUE)
a <- mApply(iris[,-5], iris$Species, mean)
```

`mChoice` *Methods for Storing and Analyzing Multiple Choice Variables*

Description

`mChoice` is a function that is useful for defining a group of variables on the right side of the formula. The variables can represent individual choices on a multiple choice question. These choices are typically factor or character values but may be of any type. Levels of component factor variables need not be the same; all unique levels (or unique character values) are collected over all of the multiple variables. Then a new character vector is formed with integer choice numbers separated by semicolons. Optimally, a database system would have exported the semicolon-separated character strings with a `levels` attribute containing strings defining value labels corresponding to the integer choice numbers. `mChoice` is a function for creating a multiple-choice variable after the fact. `mChoice` variables are explicitly handed by the `describe` and `summary.formula` functions. `NA`s or blanks in input variables are ignored.

`format.mChoice` will convert the multiple choice representation to text form by substituting `levels` for integer codes. `as.double.mChoice` converts the `mChoice` object to a binary numeric matrix, one column per used level (or all levels of `drop=FALSE`. This is called by the user by invoking `as.numeric`. There is a `print` method and a `summary` method, and a `print` method for the `summary.mChoice` object. The `summary` method computes frequencies of all two-way choice combinations, the frequencies of the top 5 combinations, information about which other choices are present when each given choice is present, and the frequency distribution of the number of choices per observation. This `summary` output is used in the `describe` function.

`inmChoice` creates a logical vector the same length as `x` whose elements are `TRUE` when the observation in `x` contains at least one of the codes or value labels in the second argument.

`is.mChoice` returns `TRUE` is the argument is a multiple choice variable.

Usage

```
mChoice(..., label='', sort.=TRUE,
        sort.levels=c('original','alphabetic'),
        add.none=FALSE, drop=TRUE)

## S3 method for class 'mChoice':
format(x, minlength=NULL, sep=";", ...)

## S3 method for class 'mChoice':
as.double(x, drop=FALSE, ...)
```

```
## S3 method for class 'mChoice':
print(x, long=FALSE, ...)

## S3 method for class 'mChoice':
summary(object, ncombos=5, minlength=NULL, drop=TRUE, ...)

## S3 method for class 'summary.mChoice':
print(x, prlabel=TRUE, ...)

## S3 method for class 'mChoice':
x[..., drop=FALSE]

inmChoice(x, values)

is.mChoice(x)
```

Arguments

`...`	a series of vectors
`sort.`	By default, choice codes are sorted in ascending numeric order. Set `sort=FALSE` to preserve the original left to right ordering from the input variables.
`label`	a character string `label` attribute to attach to the matrix created by `mChoice`
`sort.levels`	set `sort.levels="alphabetic"` to sort the columns of the matrix created by `mChoice` alphabetically by category rather than by the original order of levels in component factor variables (if there were any input variables that were factors)
`add.none`	Set `add.none` to `TRUE` to make a new category `'none'` if it doesn't already exist and if there is an observations with no choices selected.
`drop`	set `drop=FALSE` to keep unused factor levels as columns of the matrix produced by `mChoice`
`x`	an object of class `"mchoice"` such as that created by `mChoice`. For `is.mChoice` is any object.
`object`	an object of class `"mchoice"` such as that created by `mChoice`
`ncombos`	maximum number of combos.
`minlength`	By default no abbreviation of levels is done in `format` and `summary`. Specify a positive integer to use abbreviation in those functions. See `abbreviate`.
`sep`	character to use to separate levels when formatting
`long`	Set to `TRUE` to print the formatted levels. Otherwise integer codes are printed.
`prlabel`	set to `FALSE` to keep `print.summary.mChoice` from printing the variable label and number of unique values
`values`	a scalar or vector. If `values` is integer, it is the choice codes, and if it is a character vector, it is assumed to be value labels.

Value

`mChoice` returns a character vector of class `"mChoice"` plus attributes `"levels"` and `"label"`. `summary.mChoice` returns an object of class `"summary.mChoice"`. `inmChoice` returns a logical vector. `format.mChoice` returns a character vector, and `as.double.mChoice` returns a binary numeric matrix.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

`label`

Examples

```
options(digits=3)
set.seed(3)
n <- 20
sex <- factor(sample(c("m","f"), n, rep=TRUE))
age <- rnorm(n, 50, 5)
treatment <- factor(sample(c("Drug","Placebo"), n, rep=TRUE))

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
          'Muscle Ache','Depressed')
symptom1 <- sample(symp, n, TRUE)
symptom2 <- sample(symp, n, TRUE)
symptom3 <- sample(symp, n, TRUE)
cbind(symptom1, symptom2, symptom3)[1:5,]
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
Symptoms
print(Symptoms, long=TRUE)
format(Symptoms[1:5])
inmChoice(Symptoms,'Headache')
levels(Symptoms)
inmChoice(Symptoms, 3)
inmChoice(Symptoms, c('Headache','Hangnail'))
# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
# mChoice will ignore these redundant selections

meanage <- N <- numeric(5)
for(j in 1:5) {
 meanage[j] <- mean(age[inmChoice(Symptoms,j)])
 N[j] <- sum(inmChoice(Symptoms,j))
}
names(meanage) <- names(N) <- levels(Symptoms)
```

```
meanage
N

# Manually compute mean age for 2 symptoms
mean(age[symptom1=='Headache' | symptom2=='Headache' | symptom3=='Headache'])
mean(age[symptom1=='Hangnail' | symptom2=='Hangnail' | symptom3=='Hangnail'])

summary(Symptoms)

#Frequency table sex*treatment, sex*Symptoms
summary(sex ~ treatment + Symptoms, fun=table)
# Check:
ma <- inmChoice(Symptoms, 'Muscle Ache')
table(sex[ma])

# could also do:
# summary(sex ~ treatment + mChoice(symptom1,symptom2,symptom3), fun=table)

#Compute mean age, separately by 3 variables
summary(age ~ sex + treatment + Symptoms)

summary(age ~ sex + treatment + Symptoms, method="cross")

f <- summary(treatment ~ age + sex + Symptoms, method="reverse", test=TRUE)
f
# trio of numbers represent 25th, 50th, 75th percentile
print(f, long=TRUE)
```

`makeNstr` *creates a string that is a repeat of a substring*

Description

Takes a character and creates a string that is the character repeated `len` times.

Usage

```
makeNstr(char, len)
```

Arguments

`char`	character to be repeated
`len`	number of times to repeat `char`.

Value

A string that is `char` repeated `len` times.

Author(s)

Charles Dupont

See Also

`paste`, `rep`

Examples

```
makeNstr(" ", 5)
```

`mdb.get` *Read Tables in a Microsoft Access Database*

Description

Assuming the `mdb-tools` package has been installed on your system and is in the system path, `mdb.get` imports one or more tables in a Microsoft Access database. Date-time variables are converted to dates or `chron` package date-time variables. The `csv.get` function is used to import automatically exported csv files. If `tables` is unspecified all tables in the database are retrieved. If more than one table is imported, the result is a list of data frames.

Usage

```
mdb.get(file, tables=NULL, lowernames=FALSE, allow=NULL,
        dateformat='%m/%d/%y', ...)
```

Arguments

`file`	the file name containing the Access database
`tables`	character vector specifying the names of tables to import. Default is to import all tables. Specify `tables=TRUE` to return the list of available tables.
`lowernames`	set this to `TRUE` to change variable names to lower case
`allow`	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
`dateformat`	see `cleanup.import`. Default is the usual Access format used in the U.S.
`...`	arguments to pass to `csv.get`

Details

Uses the `mdb-tools` package executables `mdb-tables`, `mdb-schema`, and `mdb-export`. `cleanup.import` is invoked by `csv.get` to transform variables and store them as efficiently as possible.

Value

a new data frame or a list of data frames

Author(s)

Frank Harrell, Vanderbilt University

See Also

data.frame, cleanup.import, csv.get, Date, chron

Examples

```
## Not run:
# Read all tables in the Microsoft Access database Nwind.mdb
d <- mdb.get('Nwind.mdb')
contents(d)
for(z in d) print(contents(z))
# Just print the names of tables in the database
mdb.get('Nwind.mdb', tables=TRUE)
# Import one table
Orders <- mdb.get('Nwind.mdb', tables='Orders')
## End(Not run)
```

---

mgp.axis *Draw Axes With Side-Specific mgp Parameters*

Description

`mgp.axis` is a version of `axis` that uses the appropriate side-specific `mgp` parameter (see `par`) to account for different space requirements for axis labels vertical vs. horizontal tick marks. `mgp.axis` also fixes a bug in `axis(2,...)` that causes it to assume `las=1`.

`mgp.axis.labels` is used so that different spacing between tick marks and axis tick mark labels may be specified for x- and y-axes. Use `mgp.axis.labels('default')` to set defaults. Users can set values manually using `mgp.axis.labels(x,y)` where `x` and `y` are 2nd value of `par('mgp')` to use. Use `mgp.axis.labels(type=w)` to retrieve values, where `w='x'`, `'y'`, `'x and y'`, `'xy'`, to get 3 `mgp` values (first 3 types) or 2 `mgp.axis.labels`.

Usage

```
mgp.axis(side, at = NULL, ...,
          mgp = mgp.axis.labels(type = if (side == 1 | side == 3) "x"
                                else "y"),
          axistitle = NULL)

mgp.axis.labels(value,type=c('xy','x','y','x and y'))
```

Arguments

`side`	
`at`	see `par`
`...`	arguments passed through to `axis`
`mgp`	see `par`
`axistitle`	if specified will cause `axistitle` to be drawn on the appropriate axis as a title
`value`	vector of values to which to set system option `mgp.axis.labels`
`type`	see above

Value

`mgp.axis.labels` returns the value of `mgp` (only the second element of `mgp` if `type="xy"` or a list with elements `x` and `y` if `type="x or y"`, each list element being a 3-vector) for the appropriate axis if `value` is not specified, otherwise it returns nothing but the system option `mgp.axis.labels` is set.

`mgp.axis` returns nothing.

Side Effects

`mgp.axis.labels` stores the value in the system option `mgp.axis.labels`

Author(s)

Frank Harrell

See Also

`par`

Examples

```
## Not run:
mgp.axis.labels(type='x')  # get default value for x-axis
mgp.axis.labels(type='y')  # get value for y-axis
mgp.axis.labels(type='xy') # get 2nd element of both mgps
mgp.axis.labels(type='x and y')  # get a list with 2 elements
mgp.axis.labels(c(3,.5,0), type='x')  # set
options('mgp.axis.labels')            # retrieve

plot(..., axes=FALSE)
mgp.axis(1, "X Label")
mgp.axis(2, "Y Label")

## End(Not run)
```

`minor.tick` *Minor Tick Marks*

Description

Adds minor tick marks to an existing plot. All minor tick marks that will fit on the axes will be drawn.

Usage

```
minor.tick(nx=2, ny=2, tick.ratio=0.5)
```

Arguments

`nx`	number of intervals in which to divide the area between major tick marks on the X-axis. Set to 1 to suppress minor tick marks.
`ny`	same as `nx` but for the Y-axis
`tick.ratio`	ratio of lengths of minor tick marks to major tick marks. The length of major tick marks is retrieved from `par("tck")`.

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`axis`

Examples

```
plot(runif(20),runif(20))
minor.tick()
```

`mtitle` *Margin Titles*

Description

Writes overall titles and subtitles after a multiple image plot is drawn. If `par()$oma==c(0,0,0,0)`, `title` is used instead of `mtext`, to draw titles or subtitles that are inside the plotting region for a single plot.

Usage

```
mtitle(main, ll, lc,
       lr=if(.R.) format(Sys.time(),'%d%b%y') else
        if(under.unix)unix("date '+%d%h%y'") else date(),
       cex.m=1.75, cex.l=.5, ...)
none
```

Arguments

`main`	main title to be centered over entire figure, default is none
`ll`	subtitle for lower left of figure, default is none
`lc`	subtitle for lower center of figure, default is none
`lr`	subtitle for lower right of figure, default is today's date in format 23Jan91 for UNIX or R (Thu May 30 09:08:13 1996 format for Windows). Set to `""` to suppress lower right title.
`cex.m`	character size for main, default is 1.75
`cex.l`	character size for subtitles
`...`	other arguments passed to `mtext`

Value

nothing

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`par`, `mtext`, `title`, `unix`, `pstamp`

Examples

```
#Set up for 1 plot on figure, give a main title,
#use date for lr
plot(runif(20),runif(20))
mtitle("Main Title")

#Set up for 2 x 2 matrix of plots with a lower left subtitle and overall title
par(mfrow=c(2,2), oma=c(3,0,3,0))
plot(runif(20),runif(20))
plot(rnorm(20),rnorm(20))
plot(exp(rnorm(20)),exp(rnorm(20)))
mtitle("Main Title",ll="n=20")
```

`na.delete` *Row-wise Deletion na.action*

Description

Does row-wise deletion as `na.omit`, but adds frequency of missing values for each predictor to the `"na.action"` attribute of the returned model frame. Optionally stores further details if `options(na.detail.response=TRUE)`.

Usage

```
na.delete(frame)
```

Arguments

`frame`	a model frame

Value

a model frame with rows deleted and the `"na.action"` attribute added.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`na.omit`, `na.keep`, `na.detail.response`, `model.frame.default`, `naresid`, `naprint`

Examples

```
# options(na.action="na.delete")
# ols(y ~ x)
```

`na.detail.response` *Detailed Response Variable Information*

Description

This function is called by certain `na.action` functions if `options(na.detail.response=TRUE)` is set. By default, this function returns a matrix of counts of non-NAs and the mean of the response variable computed separately by whether or not each predictor is NA. The default action uses the last column of a `Surv` object, in effect computing the proportion of events. Other summary functions may be specified by using `options(na.fun.response="name of function")`.

Usage

```
na.detail.response(mf)
```

Arguments

`mf`	a model frame

Value

a matrix, with rows representing the different statistics that are computed for the response, and columns representing the different subsets for each predictor (NA and non-NA value subsets).

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`na.omit`, `na.delete`, `model.frame.default`, `naresid`, `naprint`, `describe`

Examples

```
# sex
# [1] m f f m f f m m m m m m m m f f f m f m
# age
# [1] NA 41 23 30 44 22 NA 32 37 34 38 36 36 50 40 43 34 22 42 30
# y
# [1] 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
# options(na.detail.response=TRUE, na.action="na.delete", digits=3)
# lrm(y ~ age*sex)
#
# Logistic Regression Model
#
# lrm(formula = y ~ age * sex)
```

```
#
#
# Frequencies of Responses
#   0 1
#  10 8
#
# Frequencies of Missing Values Due to Each Variable
#  y age sex
#  0   2   0
#
#
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#     age=NA age!=NA sex!=NA Any NA  No NA
#   N    2.0  18.000   20.00    2.0 18.000
# Mean    0.5   0.444    0.45    0.5  0.444
#
# ......
# options(na.action="na.keep")
# describe(y ~ age*sex)
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#      age=NA age!=NA sex!=NA Any NA  No NA
#    N    2.0  18.000   20.00    2.0 18.000
# Mean    0.5   0.444    0.45    0.5  0.444
#
# ...
# options(na.fun.response="table")  #built-in function table()
# describe(y ~ age*sex)
#
# Statistics on Response by Missing/Non-Missing Status of Predictors
#
#   age=NA age!=NA sex!=NA Any NA No NA
# 0      1      10      11      1    10
# 1      1       8       9      1     8
#
# ...
```

`na.keep` *Do-nothing na.action*

Description

Does not delete rows containing NAs, but does add details concerning the distribution of the response variable if `options(na.detail.response=TRUE)`. This `na.action` is primarily for use with `describe.formula`.

Usage

```
na.keep(mf)
```

Arguments

`mf`	a model frame

Value

the same model frame with the `"na.action"` attribute

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`na.omit`, `na.delete`, `model.frame.default`, `na.detail.response`, `naresid`, `naprint`, `describe`

Examples

```
options(na.action="na.keep", na.detail.response=TRUE)
x1 <- runif(20)
x2 <- runif(20)
x2[1:4] <- NA
y <- rnorm(20)
describe(y ~ x1*x2)
```

`%nin%` *Find Matching (or Non-Matching) Elements*

Description

%nin% is a binary operator, which returns a logical vector indicating if there is a match or not for its left operand. A true vector element indicates no match in left operand, false indicates a match.

Usage

```
a %nin% b
```

Arguments

`a`	a vector (numeric, character, factor)
`b`	a vector (numeric, character, factor), matching the mode of a

Value

vector of logical values with length equal to length of `a`.

See Also

match %in%

Examples

```
c('a','b','c') %nin% c('a','b')
```

---

panel.bpplot *Box-Percentile Panel Function for Trellis*

Description

For all their good points, box plots have a high ink/information ratio in that they mainly display 3 quartiles. Many practitioners have found that the "outer values" are difficult to explain to non-statisticians and many feel that the notion of "outliers" is too dependent on (false) expectations that data distributions should be Gaussian.

`panel.bpplot` is a `panel` function for use with `trellis`, especially for `bwplot`. It draws box plots (without the whiskers) with any number of user-specified "corners" (corresponding to different quantiles), but it also draws box-percentile plots similar to those drawn by Jeffrey Banfield's (umsfjban@bill.oscs.montana.edu) `bpplot` function. To quote from Banfield, "box-percentile plots supply more information about the univariate distributions. At any height the width of the irregular 'box' is proportional to the percentile of that height, up to the 50th percentile, and above the 50th percentile the width is proportional to 100 minus the percentile. Thus, the width at any given height is proportional to the percent of observations that are more extreme in that direction. As in boxplots, the median, 25th and 75th percentiles are marked with line segments across the box."

`panel.bpplot` is a generalization of `bpplot` and `panel.bwplot` in that it works with `trellis` (making the plots horizontal so that category labels are more visable), it allows the user to specify the quantiles to connect and those for which to draw reference lines, and it displays means (by default using dots).

`bpplt` draws horizontal box-percentile plot much like those drawn by `panel.bpplot` but taking as the starting point a matrix containing quantiles summarizing the data. `bpplt` is primarily intended to be used internally by `plot.summary.formula.reverse` but when used with no arguments has a general purpose: to draw an annotated example box-percentile plot with the default quantiles used and with the mean drawn with a solid dot. This schematic plot is rendered nicely in postscript with an image height of 3.5 inches.

Usage

```
panel.bpplot(x, y, box.ratio=1, means=TRUE, qref=c(.5,.25,.75),
             probs=c(.05,.125,.25,.375), nout=0,
             datadensity=FALSE, scat1d.opts=NULL,
             font=box.dot$font, pch=box.dot$pch,
             cex =box.dot$cex,  col=box.dot$col, ...)

# E.g. bwplot(formula, panel=panel.bpplot, panel.bpplot.parameters)
```

```
bpplt(stats, xlim, xlab='', box.ratio = 1, means=TRUE,
      qref=c(.5,.25,.75), qomit=c(.025,.975),
      pch=16, cex.labels=par('cex'), cex.points=if(prototype)1 else 0.5,
      grid=FALSE)
```

Arguments

`x`	continuous variable whose distribution is to be examined
`y`	grouping variable
`box.ratio`	see `panel.bwplot`
`means`	set to `FALSE` to suppress drawing a character at the mean value
`qref`	vector of quantiles for which to draw reference lines. These do not need to be included in `probs`.
`probs`	vector of quantiles to display in the box plot. These should all be less than 0.5; the mirror-image quantiles are added automatically. By default, `probs` is set to `c(.05,.125,.25,.375)` so that intervals contain 0.9, 0.75, 0.5, and 0.25 of the data. To draw all 99 percentiles, i.e., to draw a box-percentile plot, set `probs=seq(.01,.49,by=.01)`. To make a more traditional box plot, use `probs=.25`.
`nout`	tells the function to use `scat1d` to draw tick marks showing the `nout` smallest and `nout` largest values if `nout >= 1`, or to show all values less than the `nout` quantile or greater than the `1-nout` quantile if `0 < nout <= 0.5`. If `nout` is a whole number, only the first `n/2` observations are shown on either side of the median, where `n` is the total number of observations.
`datadensity`	set to `FALSE` to invoke `scat1d` to draw a data density (one-dimensional scatter diagram or rug plot) inside each box plot.
`scat1d.opts`	a list containing named arguments (without abbreviations) to pass to `scat1d` when `datadensity=TRUE` or `nout > 0`
`font`	
`pch`	
`cex`	
`col`	see `panel.bwplot`
`...`	arguments passed to `points`
`stats`	
`xlim`	
`xlab`	
`qomit`	
`cex.labels`	
`cex.points`	
`grid`	undocumented arguments to `bpplt`

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Esty, W. W. and Banfield, J. D. (1992) "The Box-Percentile Plot," Technical Report (May 15, 1992), Department of Mathematical Sciences, Montana State University.

See Also

`bpplot`, `panel.bwplot`, `scat1d`, `quantile`, `Ecdf`

Examples

```
set.seed(13)
x <- rnorm(1000)
g <- sample(1:6, 1000, replace=TRUE)
x[g==1][1:20] <- rnorm(20)+3   # contaminate 20 x's for group 1

# default trellis box plot
require(lattice)
bwplot(g ~ x)

# box-percentile plot with data density (rug plot)
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.01,.49,by=.01), datadensity=TRUE)
# add ,scat1d.opts=list(tfrac=1) to make all tick marks the same size
# when a group has > 125 observations

# small dot for means, show only .05,.125,.25,.375,.625,.75,.875,.95 quantiles
bwplot(g ~ x, panel=panel.bpplot, cex=.3)

# suppress means and reference lines for lower and upper quartiles
bwplot(g ~ x, panel=panel.bpplot, probs=c(.025,.1,.25), means=FALSE, qref=FALSE)

# continuous plot up until quartiles ("Tootsie Roll plot")
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.01,.25,by=.01))

# start at quartiles then make it continuous ("coffin plot")
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.25,.49,by=.01))

# same as previous but add a spike to give 0.95 interval
bwplot(g ~ x, panel=panel.bpplot, probs=c(.025,seq(.25,.49,by=.01)))

# decile plot with reference lines at outer quintiles and median
bwplot(g ~ x, panel=panel.bpplot, probs=c(.1,.2,.3,.4), qref=c(.5,.2,.8))

# default plot with tick marks showing all observations outside the outer
# box (.05 and .95 quantiles), with very small ticks
```

```
bwplot(g ~ x, panel=panel.bpplot, nout=.05, scat1d.opts=list(frac=.01))

# show 5 smallest and 5 largest observations
bwplot(g ~ x, panel=panel.bpplot, nout=5)

# Use a scat1d option (preserve=TRUE) to ensure that the right peak extends
# to the same position as the extreme scat1d
bwplot(~x , panel=panel.bpplot, probs=seq(.00,.5,by=.001),
       datadensity=TRUE, scat1d.opt=list(preserve=TRUE))

# Draw a prototype showing how to interpret the plots
bpplt()

# make a local copy of bwplot that always uses panel.bpplot (S-Plus only)
# bwplot$panel <- panel.bpplot
# bwplot(g ~ x, nout=.05)
```

`partition` *Patitions an object into different sets*

Description

Partitions an object into subsets of length defined in the `sep` argument.

Usage

```
partition.vector(x, sep, ...)
partition.matrix(x, rowsep, colsep, ...)
```

Arguments

`x`	object to be partitioned.
`sep`	determines how many elements should go into each set. The sum of `sep` should be equal to the length of `x`.
`rowsep`	determins how many rows should go into each set. The sum of `rowsep` must equal the number of rows in `x`.
`colsep`	determins how many columns should go into each set. The sum of `colsep` must equal the number of columns in `x`.
`...`	arguments used in other methods of `partition`.

Value

A list of equal length as `sep` containing the partitioned objects.

Author(s)

Charles Dupont

See Also

`split`

Examples

```
a <- 1:7
partition.vector(a, sep=c(1,3,2,1))
```

`pc1` *First Principal Component*

Description

Given a numeric matrix which may or may not contain `NA`s, `pc1` standardizes the columns to have mean 0 and variance 1 and computes the first principal component using `prcomp`. The proportion of variance explained by this component is printed, and so are the coefficients of the original (not scaled) variables. These coefficients may be applied to the raw data to obtain the first PC.

Usage

```
pc1(x, hi)
```

Arguments

`x`	numeric matrix
`hi`	if specified, the first PC is scaled so that its maximum value is `hi` and its minimum value is zero

Value

The vector of observations with the first PC. An attribute `"coef"` is attached to this vector. `"coef"` contains the raw-variable coefficients.

Author(s)

Frank Harrell

See Also

`prcomp`

Examples

```
set.seed(1)
x1 <- rnorm(100)
x2 <- x1 + rnorm(100)
w <- pc1(cbind(x1,x2))
attr(w,'coef')
```

plotCorrPrecision *Plot Precision of Estimate of Pearson Correlation Coefficient*

Description

This function plots the precision (margin of error) of the product-moment linear correlation coefficient r vs. sample size, for a given vector of correlation coefficients `rho`. Precision is defined as the larger of the upper confidence limit minus rho and rho minus the lower confidence limit. `labcurve` is used to automatically label the curves.

Usage

```
plotCorrPrecision(rho = c(0, 0.5), n = seq(10, 400, length = 100),
                  conf.int = 0.95)
```

Arguments

`rho`	single or vector of true correlations. A worst-case precision graph results from rho=0
`n`	vector of sample sizes to use on the x-axis
`conf.int`	confidence coefficient; default uses 0.95 confidence limits

Author(s)

Xing Wang and Frank Harrell

See Also

`rcorr`,`cor`,`cor.test`

Examples

```
plotCorrPrecision()
plotCorrPrecision(rho=0)
```

plsmo *Plot smoothed estimates*

Description

Plot smoothed estimates of x vs. y, handling missing data for lowess or supsmu, and adding axis labels. Optionally suppresses plotting extrapolated estimates. An optional `group` variable can be specified to compute and plot the smooth curves by levels of `group`. When `group` is present, the `datadensity` option will draw tick marks showing the location of the raw `x`-values, separately for each curve. `plsmo` has an option to plot connected points for raw data, with no smoothing.

`panel.plsmo` is a `panel` function for `trellis` for the `xyplot` function that uses `plsmo` and its options to draw one or more nonparametric function estimates on each panel. This has advantages over using `xyplot` with `panel.xyplot` and `panel.loess`: (1) by default it will invoke `labcurve` to label the curves where they are most separated, (2) the `datadensity` option will put rug plots on each curve (instead of a single rug plot at the bottom of the graph), and (3) when `panel.plsmo` invokes `plsmo` it can use the "super smoother" (`supsmu` function) instead of `lowess`. `panel.plsmo` senses when a `group` variable is specified to `xyplot` so that it can invoke `panel.superpose` instead of `panel.xyplot`. Using `panel.plsmo` through `trellis` has some advantages over calling `plsmo` directly in that conditioning variables are allowed and `trellis` uses nicer fonts etc.

When a `group` variable was used, `panel.plsmo` creates a function `Key` in the session frame that the user can invoke to draw a key for individual data point symbols used for the `groups`. By default, the key is positioned at the upper right corner of the graph. If `Key(locator(1))` is specified, the key will appear so that its upper left corner is at the coordinates of the mouse click.

Usage

```
plsmo(x, y, method=c("lowess","supsmu","raw"), xlab, ylab,
      add=FALSE, lty=1:nlev, col=par("col"), lwd=par("lwd"),
      iter=if(length(unique(y))>2) 3 else 0, bass=0, trim,
      fun, group, prefix, xlim, ylim,
      label.curves=TRUE, datadensity=FALSE, lines.=TRUE, subset=TRUE,
      grid=FALSE, ...)

#To use panel function:
#xyplot(formula=y ~ x | conditioningvars, groups,
#       panel=panel.plsmo, type='b',
#       label.curves=TRUE,
#       lwd = superpose.line$lwd,
#       lty = superpose.line$lty,
#       pch = superpose.symbol$pch,
#       cex = superpose.symbol$cex,
#       font = superpose.symbol$font,
#       col = NULL, ...)
```

Arguments

`x`	vector of x-values, NAs allowed
`y`	vector of y-values, NAs allowed
`method`	"lowess" (the default), "supsmu", or "raw" to not smooth at all
`xlab`	x-axis label iff add=F. Defaults of label(x) or argument name.

`ylab`	y-axis label, like xlab.
`add`	Set to T to call lines instead of plot. Assumes axes already labeled.
`lty`	line type, default=1,2,3,. . . , corresponding to `group`
`col`	color for each curve, corresponding to `group`. Default is current `par("col")`.
`lwd`	vector of line widths for the curves, corresponding to `group`. Default is current `par("lwd")`. `lwd` can also be specified as an element of `label.curves` if `label.curves` is a list.
`iter`	iter parameter if method="lowess", default=0 if `y` is binary, and 3 otherwise.
`bass`	bass parameter if method="supsmu", default=0.
`trim`	only plots smoothed estimates between trim and 1-trim quantiles of x. Default is to use 10th smallest to 10th largest x in the group if the number of observations in the group exceeds 200 (0 otherwise). Specify trim=0 to plot over entire range.
`fun`	after computing the smoothed estimates, if `fun` is given the y-values are transformed by `fun()`
`group`	a variable, either a `factor` vector or one that will be converted to `factor` by `plsmo`, that is used to stratify the data so that separate smooths may be computed
`prefix`	a character string to appear in group of group labels. The presence of `prefix` ensures that `labcurve` will be called even when `add=TRUE`.
`xlim`	a vector of 2 x-axis limits. Default is observed range.
`ylim`	a vector of 2 y-axis limits. Default is observed range.
`label.curves`	set to `FALSE` to prevent `labcurve` from being called to label multiple curves corresponding to `groups`. Set to a list to pass options to `labcurve`. `lty` and `col` are passed to `labcurve` automatically.
`datadensity`	set to `TRUE` to draw tick marks on each curve, using x-coordinates of the raw data `x` values. This is done using `scat1d`.
`lines.`	set to `FALSE` to suppress smoothed curves from being drawn. This can make sense if `datadensity=TRUE`.
`subset`	a logical or integer vector specifying a subset to use for processing, with respect too all variables being analyzed
`grid`	set to `TRUE` if the R `grid` package drew the current plot
`...`	optional arguments that are passed to `scat1d`, or optional parameters to pass to `plsmo` from `panel.plsmo`. See optional arguments for `plsmo` above.
`type`	set to `p` to have `panel.plsmo` plot points (and not call `plsmo`), `l` to call `plsmo` and not plot points, or use the default `b` to plot both.
`pch`	
`cex`	
`font`	vectors of graphical parameters corresponding to the `groups` (scalars if `group` is absent). By default, the parameters set up by `trellis` will be used.

Value

`plsmo` returns a list of curves (x and y coordinates) that was passed to `labcurve`

Side Effects

plots, and `panel.plsmo` creates the `Key` function in the session frame.

See Also

`lowess`, `supsmu`, `label`, `quantile`, `labcurve`, `scat1d`, `xyplot`, `panel.superpose`, `panel.xyplot`

Examples

```
set.seed(1)
x <- 1:100
y <- x + runif(100, -10, 10)
plsmo(x,y,"supsmu",xlab="Time of Entry")
#Use label(y) or "y" for ylab

plsmo(x,y,add=TRUE,lty=2)
#Add lowess smooth to existing plot, with different line type

age <- rnorm(500, 50, 15)
survival.time <- rexp(500)
sex <- sample(c('female','male'), 500, TRUE)
race <- sample(c('black','non-black'), 500, TRUE)
plsmo(age, survival.time < 1, fun=qlogis, group=sex) # plot logit by sex

#Plot points and smooth trend line using trellis
# (add type='l' to suppress points or type='p' to suppress trend lines)
require(lattice)
xyplot(survival.time ~ age, panel=panel.plsmo)

#Do this for multiple panels
xyplot(survival.time ~ age | sex, panel=panel.plsmo)

#Do this for subgroups of points on each panel, show the data
#density on each curve, and draw a key at the default location
xyplot(survival.time ~ age | sex, groups=race, panel=panel.plsmo,
       datadensity=TRUE)
Key()

#Use wloess.noiter to do a fast weighted smooth
plot(x, y)
lines(wtd.loess.noiter(x, y))
lines(wtd.loess.noiter(x, y, weights=c(rep(1,50), 100, rep(1,49))), col=2)
points(51, y[51], pch=18)   # show overly weighted point
#Try to duplicate this smooth by replicating 51st observation 100 times
lines(wtd.loess.noiter(c(x,rep(x[51],99)),c(y,rep(y[51],99)),
      type='ordered all'), col=3)
#Note: These two don't agree exactly
```

popower — *Power and Sample Size for Ordinal Response*

Description

popower computes the power for a two-tailed two sample comparison of ordinal outcomes under the proportional odds ordinal logistic model. The power is the same as that of the Wilcoxon test but with ties handled properly. posamsize computes the total sample size needed to achieve a given power. Both functions compute the efficiency of the design compared with a design in which the response variable is continuous. print methods exist for both functions. Any of the input arguments may be vectors, in which case a vector of powers or sample sizes is returned. These functions use the methods of Whitehead (1993).

Usage

```
popower(p, odds.ratio, n, n1, n2, alpha=0.05)
## S3 method for class 'popower':
print(x, ...)
posamsize(p, odds.ratio, fraction=.5, alpha=0.05, power=0.8)
## S3 method for class 'posamsize':
print(x, ...)
```

Arguments

p	a vector of marginal cell probabilities which must add up to one. The ith element specifies the probability that a patient will be in response level i, averaged over the two treatment groups.
odds.ratio	the odds ratio to be able to detect. It doesn't matter which group is in the numerator.
n	total sample size for popower. You must specify either n or n1 and n2. If you specify n, n1 and n2 are set to n/2.
n1	for popower, the number of subjects in treatment group 1
n2	for popower, the number of subjects in group 2
alpha	type I error
x	an object created by popower or posamsize
fraction	for posamsize, the fraction of subjects that will be allocated to group 1
power	for posamsize, the desired power (default is 0.8)
...	unused

Value

a list containing power and eff (relative efficiency) for popower, or containing n and eff for posamsize.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Whitehead J (1993): Sample size calculations for ordered categorical data. Stat in Med 12:2257–2271.

Julious SA, Campbell MJ (1996): Letter to the Editor. Stat in Med 15: 1065–1066. Shows accuracy of formula for binary response case.

See Also

`bpower`, `cpower`

Examples

```
#For a study of back pain (none, mild, moderate, severe) here are the
#expected proportions (averaged over 2 treatments) that will be in
#each of the 4 categories:

p <- c(.1,.2,.4,.3)
popower(p, 1.2, 1000)   # OR=1.2, total n=1000
posamsize(p, 1.2)
popower(p, 1.2, 3148)
```

`print.char.list` *prints a list of lists in a visually readable format.*

Description

`print.char.list` Takes a list that is composed of other lists and matrixs and prints it in a visually readable format.

Usage

```
print.char.list(x, ..., hsep = c("|"), vsep = c("-"), csep = c("+"), print.it = TRU
```

Arguments

`x`	list object to be printed
`...`	place for extra arguments to reside.
`hsep`	character used to separate horizontal fields
`vsep`	character used to separate veritcal feilds

`csep`	character used where horizontal and veritcal separators meet.
`print.it`	should the value be `cat`ed out or returned as a string.
`rowname.halign`	horizontal justification of row names.
`rowname.valign`	verical justification of row names.
`colname.halign`	horizontal justification of column names.
`colname.valign`	verical justification of column names.
`text.halign`	horizontal justification of cell text.
`text.valign`	vertical justification of cell text.
`rowname.width`	minimumwidth of row name strings.
`rowname.height`	minimum height of row name strings.
`min.colwidth`	minimum column width.
`max.rowheight`	maximum row height.
`abbreviate.dimnames`	should the row and column names be abbreviated.
`page.width`	width of the page being printed on.
`colname.width`	minimum width of the column names.
`colname.height`	minimum height of the column names
`prefix.width`	maximum width of the rowname columns
`superprefix.width`	maximum width of the super rowname columns

Value

String that formated table of the list object.

Author(s)

Charles Dupont

`print.char.matrix` *Function to print a matrix with stacked cells*

Description

Prints a dataframe or matrix in stacked cells. Line break charcters in a matrix element will result in a line break in that cell, but tab characters are not supported.

Usage

```
print.char.matrix(x, file = "", col.name.align = "cen", col.txt.align = "right",
    cell.align = "cen", hsep = "|", vsep = "-", csep = "+", row.names = TRUE,
    col.names = FALSE, append = FALSE,
    top.border = TRUE, left.border = TRUE, ...)
```

Arguments

`x`	a matrix or dataframe
`file`	name of file if file output is desired. If left empty, output will be to the screen
`col.name.align`	if column names are used, they can be aligned right, left or centre. Default `"cen"` results in names centred between the sides of the columns they name. If the width of the text in the columns is less than the width of the name, `col.name.align` will have no effect. Other options are `"right"` and `"left"`.
`col.txt.align`	how character columns are aligned. Options are the same as for `col.name.align` with no effect when the width of the column is greater than its name.
`cell.align`	how numbers are displayed in columns
`hsep`	character string to use as horizontal separator, i.e. what separates columns
`vsep`	character string to use as vertical separator, i.e. what separates rows. Length cannot be more than one.
`csep`	character string to use where vertical and horizontal separators cross. If `hsep` is more than one character, `csep` will need to be the same length. There is no provision for multiple vertical separators
`row.names`	logical: are we printing the names of the rows?
`col.names`	logical: are we printing the names of the columns?
`append`	logical: if file is not `""`, are we appending to the file or overwriting?
`top.border`	logical: do we want a border along the top above the columns?
`left.border`	logical: do we want a border along the left of the first column?
`...`	unused

Details

If any column of `x` is a mixture of character and numeric, the distinction between character and numeric columns will be lost. This is especially so if the matrix is of a form where you would not want to print the column names, the column information being in the rows at the beginning of the matrix.

Row names, if not specified in the making of the matrix will simply be numbers. To prevent printing them, set `row.names = FALSE`.

Value

No value is returned. The matrix or dataframe will be printed to file or to the screen.

Author(s)

Patrick Connolly ⟨p.connolly@hortresearch.co.nz⟩

See Also

`write`, `write.table`

Examples

```
data(HairEyeColor)
print.char.matrix(HairEyeColor[ , , "Male"], col.names = TRUE)
print.char.matrix(HairEyeColor[ , , "Female"], col.txt.align = "left", col.names = TRUE)

z <- rbind(c("", "N", "y"),
           c("[ 1.34,40.3)\n[40.30,48.5)\n[48.49,58.4)\n[58.44,87.8]",
             " 50\n 50\n 50\n 50",
             "0.530\n0.489\n0.514\n0.507"),
           c("female\nmale", " 94\n106", "0.552\n0.473"  ),
           c("", "200", "0.510"))
dimnames(z) <- list(c("", "age", "sex", "Overall"),NULL)

print.char.matrix(z)
```

`prnz` *Print and Object with its Name*

Description

Prints an object with its name and with an optional descriptive text string. This is useful for annotating analysis output files and for debugging.

Usage

```
prn(x, txt)
```

Arguments

`x`	any object
`txt`	optional text string

Side Effects

prints

See Also

`print`, `cat`

Examples

```
x <- 1:5
prn(x)
# prn(fit, 'Full Model Fit')
```

`ps.slide` *Postscript and Adobe PDF Setup for 35mm Slides and Other Formats*

Description

The `ps.slide` function has nice defaults to create postscript images with larger font, thicker lines, and better axis labeling. These images can be used to make nice slides. There is an option to view the constructed postscript file using `ghostview`, and an option to initiate a background process to convert the postscript file to a PC Paintbrush `.pcx` file for importing into various PC presentation graphics packages although with a significant loss in resolution. This option assumes you have installed various public-domain unix image conversion programs. You can preview `.pcx` files using e.g. `xli file.pcx &`. Specify `type=1` to make nice fullsize graphs or `type=3` for making 5 x 7" landscape graphs using 14-point type (useful for submitting to journals). `type=2` (the default) is for color 35mm slides. Use `type=4` to make nice black and white overhead projection transparancies (portrait mode). This uses line thickness 4, pointsize 14, height 8, width 7. For `type=3`, numbers on the y-axis are written horizontally (`las` defaults to `1` for `type=3`).

`ps.slide` calls `mgp.axis.labels` in Hmisc set up axis-specific defaults for the 2nd `mgp` graphical parameter. See `Overview` for Hmisc for help. This is only used automatically for select high-level graphics functions in Hmisc and Design, as S-Plus only supports a single distance between tick marks and tick mark labels using `par`, and when `las=1` a larger distance is needed for the y-axis.

See the body of the function for `type`-specific default values for many of the parameters. This function has not been tested for color output on Windows systems.

`setps` is a function that makes small postscript plots with minimal surrounding white space, suitable for inclusion in books and reports. Internally `setps` uses (and defines) the `psfig` function by Antonio Possolo (antonio@atc.boeing.com). `setps` is especially good for including plots in LaTeX. `setps` creates a temporary function in the session database that when invoked will convert a completed postscript graphics file to a Adobe Acrobat .pdf if you have Ghostscript installed and in your path (so that the `gs` command is available in UNIX or `gswin32c` is available for Windows/NT). Invoke `topdf` by the command `topdf()`, or, if you want to convert a graphic other than the last one created, run `topdf(filename)` to convert `filename.ps` to `filename.pdf`. If `trellis=TRUE`, `setps` invokes `trellis.device` with a `postscript` device argument, and it does not set any of the `par` parameters. Arguments 3, 4, 5, 7, 9, and 10 to `setps` are ignored if `trellis=TRUE`. If `options(setpsPrefix="name")` is defined, the `"name"` string will be prefixed to the file name used by `setps`. `setpdf` uses a similar option `setpdfPrefix`. `setps` and `setpdf` set `par(mgp=c(2,0.4,0))` if `trellis=FALSE`.

`setpdf` is similar to `setps` but for making Adobe Acrobat PDF graphics files directly. There are a few problems with the S-Plus `pdf.graph` function used by `setpdf`, though: (1) the default for points (open circle) is too large, (2) graphs are not centered properly, (3) gray scale does not work, and (4) there is some wasted space at the bottom of the graph. When drawing points, the user may want to specify `cex=0.7`. It may be better to use `setps` followed by `topdf()`.

`tex` is a little function to save typing when including `\tex` commands in graphs that are used with the psfrag package in LaTeX to typeset any LaTeX text inside a postscript graphic. `tex` surrounds the input character string with \tex[options]{}. This is especially useful for getting Greek letters and math symbols in postscript graphs. By default `tex` returns a string with `psfrag` commands specifying that the string be centered, not rotated, and not specially enlarged or shrunk.

`showPsfrag` is used to display (using ghostview) a postscript image that contained psfrag LaTeX strings, by building a small LaTeX script and running latex and dvips.

Usage

```
ps.slide(file, background = if (type != 2) "white" else "navy blue",
         foreground = if (type == 2) "yellow" else
                      (if(background == "white") "black" else "white"),
         font = "Helvetica", pointsize = c(24, 28, 14, 14)[type],
         hor = type != 4, lwd = c(2, 5, 2, 4)[type],
         mgp = if(under.unix) list(c(1.8, 0.4, 0), c(1.5, 0.2, 0),
                 c(2, 0.4, 0), c(1.5, 0.2, 0))[[type]] else
                 list(c(1.8, 0.5, 0), c(1.5, 0.4, 0), c(2, 0.5, 0),
                 c(1.5, 0.4, 0))[[type]],
         mar = list(c(4, 3, 2, 1) + 0.1, c(5, 4, 2.25, 2) + 0.1,
                 c(3, 3, 1, 1) + 0.1, c(5, 4, 2.25, 2) + 0.1)[[type]],
         pch = 202, view = FALSE, pcx = FALSE, tiff = FALSE,
         close = view | pcx | tiff, bty = "l",
         type = 2, height = switch(type, NULL, NULL, 5, 8),
         width = switch(type, NULL, NULL, 7, 7),
         tck = if (type == 3 || !under.unix) -0.013 else par("tck"),
         las = if (type == 3) 1 else 0, eps =  FALSE, ...)

setps(filename, w=0, h=3, pointsize=10, sublines=0, toplines=0,
      type="symbol", lwd=2, font="Helvetica",
      leftlines=0, las=1,
      trellis=!(missing(setTrellis.) & missing(strip.blank) &
                missing(lty.dot.line) & missing(lwd.dot.line)),
      setTrellis.=TRUE,
      strip.blank =TRUE, lty.dot.line = 1, lwd.dot.line = 1,
      seqno=NULL, color=FALSE)

setpdf(filename, w=0, h=4, pointsize=10, sublines=0, toplines=0,
       type="symbol", lwd=1.5, font=if(.R.)"Helvetica" else 1,
       ratio= if(.R.) 4/3 else (1 + sqrt(5))/2,
       leftlines=0, las=1, bty='l', hor=FALSE,
       trellis=!(missing(setTrellis.) & missing(strip.blank) &
                 missing(lty.dot.line) & missing(lwd.dot.line)),
       setTrellis.=TRUE,
       strip.blank =TRUE, lty.dot.line = 1, lwd.dot.line =1,
       region=c(0, 0, h, w), color=FALSE, seqno=NULL, ...)

tex(string, lref='c', psref='c', scale=1, srt=0)

showPsfrag(filename)
```

Arguments

`file`	
`filename`	name or character string or character vector specifying file prefix. For `setps` or `setpdf` specify `type="char"` if this is a character vector or a quote-delimited character string.
`string`	a character string to be processed by `psfrag` in LaTeX.
`background`	default is yellow on navy blue background (black on white for `type=1,3`. `background` may also be set to any legitimate background color listed in the S-supplied object `ps.colors.rgb`.
`foreground`	foreground color. See `background` for allowable values.
`font`	font for text. Replaces the first font in the standard list of fonts in `ps.options("fonts")`. If `font="Times-Roman"`, the fifth font (normally `Helvetica-Bold`) is set to `Times-Bold`. For `setpdf`, `font` is a number, and the default is `1` for Helvetica. All default fonts are Helvetica for `setps`, `psfig`, and `ps.slide`.
`pointsize`	postscript point size. Set to a larger number if using multiple plots via `par(mfrow=)`.
`hor`	default is `TRUE` to make a horizontal graph
`lwd`	line width
`mgp`	see `par`. Defaults are chosen according to `type`.
`mar`	margins (see `par`)
`pch`	see `par`
`view`	set to `TRUE` to initiate a `ghostview` run to view the postscript file. This option will also close out the postscript file (this is done before viewing). If you have an active `ghostview` window for this file already, you can just type `graphics.off()` or `dev.off()` to re-create the `.ps` file. `ghostview` will then update the image automatically.
`pcx`	set to `TRUE` to initiate conversion to `pcx` format. Also implies `close=TRUE`.
`tiff`	set to `TRUE` to initiate conversion to `tiff` format. Also implies `close=TRUE`.

`close`	set to `TRUE` to finish construction of the postscript file.
`bty`	box type surrounding graph. Default is `"l"` for `"L"` shape. Use `"c"` for complete box.
`type`	For `ps.slide`, `type` is an integer. In this case, set `type=1` to use black on white background, smaller pointsize, and other settings that are good for making overhead transparencies and graphs to include in reports. Set `type=3` for 5" x 7" landscape plots, and `type=4` for overheads. For `setps` and `setpdf`, `type="char"` specifies that the `filename` argument is a character string or vector, and any other value indicates that it is an unquoted name.
`height`	defaults to 5 if `type=3`, otherwise no default (except for type=4)
`width`	defaults to 7 if `type=3`, otherwise no default (except for type=4)
`tck`	length of tick marks. See `par`.
`las`	set to `0` to have axis labels always parallel to the axis, `1` for always horizontal, `2` for perpendicular to axis
`eps`	set to `TRUE` if you are going to be importing the postscript file to a system that really cares that it is marked to officially be encapsulated postscript. If you set `eps=TRUE`, you may put only one figure in the file (see the `onefile` argument in `postscript`). This applies to UNIX systems only.
`...`	other arguments to `ps.options` (or `postscript` for Windows or `pdf.graph` for `setpdf`)
`w`	width of plot. Default is chosen to scale nicely to `h` for a landscape plot
`h`	height of plot (default is 3in)
`sublines`	number of lines to reserve for subtitles
`toplines`	number of lines to reserve for main title
`leftlines`	number of lines to reserve for left margin
`trellis`	set to `TRUE` to set up for postscript output for Trellis graphics. This makes `trellis.device("postscript", ...)` be called instead of `postscript(...)` directly, and leaves `par` parameters at defaults.
`setTrellis.`	set to `FALSE` to prevent `setTrellis` from being called to set the strip panel background and to set characteristics for dot plot reference lines
`strip.blank`	set to `FALSE` to keep shading in conditioning variable panel titles, if `setTrellis.=TRUE`
`lty.dot.line`	if `setTrellis.=TRUE`, the line type for dot plot reference lines (default = solid line)
`lwd.dot.line`	if `setTrellis.=TRUE`, the line width for dot plot reference lines (default = 1)
`seqno`	if non-null, pastes the value of `seqno` at the end of the base of the file name, for `setps` and `setpdf`
`color`	set `color=TRUE` to use a color Trellis device instead of default of black and white, for `setps`. For `setpdf` set to `TRUE` to get color pdf graphics.
`region`	see `pdf.graph`. Default is to use an image region that is just large enough to contain the graphic.

`ratio`	ratio of width to height of the plot when only one of those is specified. Defaults depend on whether S-Plus or R are being used.
`lref`	LaTeX reference point for `string`. See the `psfrag` documentation referenced below. Default is `"c"` for centered (this is also the default for `psref`).
`psref`	PostScript reference point.
`scale`	scall factor, default is 1
`srt`	rotation for `string` in degrees (default is zero)

Value

nothing, for most of the functions. `tex` returns a modified character string.

Side Effects

Starts a postscript file or a process to convert it to pcx format, or starts a Trellis postscript device. `ps.slide` Stores a system option `ps.slide.file`. `pdf.graph` opens a graphics file using `pdf.graph`. `setps` creates a function `topdf` in frame 0 (the session database).

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Grant MC, Carlisle (1998): The PSfrag System, Version 3. Full documentation is obtained by searching www.ctan.org for pfgguide.ps.

See Also

`postscript`, `par`, `ps.options`, `mgp.axis.labels`, `pdf`, `trellis.device`, `setTrellis`

Examples

```
## Not run:
ps.slide("myslide")   # myslide is file name prefix
# use ps.slide("myslide",back="green") to use e.g. green background
plot(x, y)
title("My Title")

ps.slide(view=TRUE)   # makes myslide.ps file
                     # use ps.slide(close=TRUE) to close file without viewing with
                     # ghostview.
ps.slide(view=TRUE, pcx=TRUE)
                     # converts myslide.ps into myslide.pcx (PC Paintbrush
                     # format suitable for importing in PC graphics packages)
mgp.axis.labels(c(.4,1.2))  # override 2nd mgp parameters for x- and y axes
mgp.axis.labels(type='x')   # retrieve 3 mgp parameters for x-axis
```

```
setps(myfile)       # equiv. to setps('myfile', type='char')
                    # setps(myfile, trellis=TRUE, other args) for Trellis
# plotting commands
dev.off()
topdf()             # topdf created by setps
                    # makes Ghostscript create "myfile.pdf"
setpdf(myfile)
# plotting commands
dev.off()

# Put math and Greek symbols in a graph
setps(test)
x <- seq(0,15,length=100)
plot(x, dchisq(x, 5), xlab=tex('$x$'),
        ylab=tex('$f(x)$'), type='l')
title(tex('Density Function of the $\chi_{5}^{2}$ Distribution'))
dev.off()
# To process this file in LaTeX do something like
#\documentclass{article}
#\usepackage[scanall]{psfrag}
#\begin{document}
#\begin{figure}
#\includegraphics{test.ps}
#\caption{This is an example}
#\end{figure}
#\end{document}
## End(Not run)
```

pstamp *Date/Time/Directory Stamp the Current Plot*

Description

Date-time stamp the current plot in the extreme lower right corner. Optionally add the current working directory and arbitrary other text to the stamp.

Usage

```
pstamp(txt, pwd = FALSE, time. = TRUE)
```

Arguments

`txt`	an optional single text string
`pwd`	set to `TRUE` to add the current working directory name to the stamp
`time.`	set to `FALSE` to use the date without the time

Details

Certain functions are not supported for S-Plus under Windows. For R, results may not be satisfactory if `par(mfrow=)` is in effect.

Author(s)

Frank Harrell

Examples

```
plot(1:20)
pstamp(pwd=TRUE, time=FALSE)
```

`rMultinom` *Generate Multinomial Random Variables with Varying Probabilities*

Description

Given a matrix of multinomial probabilities where rows correspond to observations and columns to categories (and each row sums to 1), generates a matrix with the same number of rows as has `probs` and with `m` columns. The columns represent multinomial cell numbers, and within a row the columns are all samples from the same multinomial distribution. The code is a modification of that in the `impute.polyreg` function in the `MICE` package.

Usage

```
rMultinom(probs, m)
```

Arguments

`probs`	matrix of probabilities
`m`	number of samples for each row of `probs`

Value

an integer matrix having `m` columns

See Also

`rbinom`

Examples

```
set.seed(1)
w <- rMultinom(rbind(c(.1,.2,.3,.4),c(.4,.3,.2,.1)),200)
t(apply(w, 1, table)/200)
```

rcorr *Matrix of Correlations and P-values*

Description

`rcorr` Computes a matrix of Pearson's `r` or Spearman's `rho` rank correlation coefficients for all possible pairs of columns of a matrix. Missing values are deleted in pairs rather than deleting all rows of `x` having any missing variables. Ranks are computed using efficient algorithms (see reference 2), using midranks for ties.

Usage

```
rcorr(x, y, type=c("pearson","spearman"))

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

Arguments

`x`	a numeric matrix with at least 5 rows and at least 2 columns (if `y` is absent). For `print`, `x` is an object produced by `rcorr`.
`y`	a numeric vector or matrix which will be concatenated to `x`. If `y` is omitted for `rcorr`, `x` must be a matrix.
`type`	specifies the type of correlations to compute. Spearman correlations are the Pearson linear correlations computed on the ranks of non-missing elements, using midranks for ties.
`...`	argument for method compatiblity.

Details

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values are approximated by using the `t` or `F` distributions.

Value

`rcorr` returns a list with elements `r`, the matrix of correlations, `n` the matrix of number of observations used in analyzing each pair of variables, and `P`, the asymptotic P-values. Pairs with fewer than 2 non-missing values have the r values set to NA. The diagonals of `n` are the number of non-NAs for the single variable corresponding to that row and column.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Hollander M. and Wolfe D.A. (1973). Nonparametric Statistical Methods. New York: Wiley.

Press WH, Flannery BP, Teukolsky SA, Vetterling, WT (1988): Numerical Recipes in C. Cambridge: Cambridge University Press.

See Also

`hoeffd`, `cor`, `combine.levels`, `varclus`, `dotchart2`, `impute`, `chisq.test`, `cut2`.

Examples

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4,   1, 0, 1, 4)
z <- c(1,   2, 3, 4, NA)
v <- c(1,   2, 3, 4, 5)
rcorr(cbind(x,y,z,v))
```

`rcorr.cens` *Rank Correlation for Censored Data*

Description

Computes the `c` index and the corresponding generalization of Somers' Dxy rank correlation for a censored response variable. Also works for uncensored and binary responses, although its use of all possible pairings makes it slow for this purpose.

Usage

```
rcorr.cens(x, S, outx=FALSE)
```

Arguments

`x`	a numeric predictor variable
`S`	an `Surv` object or a vector. If a vector, assumes that every observation is uncensored.
`outx`	set to `TRUE` to not count pairs of observations tied on `x` as a relevant pair. This results in a Goodman–Kruskal gamma type rank correlation.

Value

a vector with the following named elements: `C Index`, `Dxy`, `S.D.`, `n`, `missing`, `uncensored`, `Relevant Pairs`, `Concordant`, and `Uncertain`

`n`	number of observations not missing on any input variables
`missing`	number of observations missing on `x` or `S`
`relevant`	number of pairs of non-missing observations for which `S` could be ordered

`concordant`	number of relevant pairs for which `x` and `S` are concordant.
`uncertain`	number of pairs of non-missing observations for which censoring prevents classification of concordance of `x` and `S`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Newson R: Confidence intervals for rank statistics: Somers' D and extensions. Stata Journal 6:309-334; 2006.

See Also

`somers2`

Examples

```
set.seed(1)
x <- round(rnorm(200))
y <- rnorm(200)
rcorr.cens(x, y, outx=TRUE)   # can correlate non-censored variables
library(survival)
age <- rnorm(400, 50, 10)
d.time <- rexp(400)
cens   <- runif(400,.5,2)
death  <- d.time <= cens
d.time <- pmin(d.time, cens)
rcorr.cens(age, Surv(d.time, death))

# Show typical 0.95 confidence limits for ROC areas for a sample size
# with 24 events and 62 non-events, for varying population ROC areas
# Repeat for 138 events and 102 non-events
set.seed(8)
par(mfrow=c(2,1))
for(i in 1:2) {
 n1 <- c(24,138)[i]
 n0 <- c(62,102)[i]
 y <- c(rep(0,n0), rep(1,n1))
 deltas <- seq(-3, 3, by=.25)
 C <- se <- deltas
 j <- 0
 for(d in deltas) {
  j <- j + 1
  x <- c(rnorm(n0, 0), rnorm(n1, d))
  w <- rcorr.cens(x, y)
  C[j]  <- w['C Index']
  se[j] <- w['S.D.']/2
```

```
   }
   low <- C-1.96*se; hi <- C+1.96*se
   print(cbind(C, low, hi))
   errbar(deltas, C, C+1.96*se, C-1.96*se,
          xlab='True Difference in Mean X',
          ylab='ROC Area and Approx. 0.95 CI')
   title(paste('n1=',n1,'  n0=',n0,sep=''))
   abline(h=.5, v=0, col='gray')
   true <- 1 - pnorm(0, deltas, sqrt(2))
   lines(deltas, true, col='blue')
  }
  par(mfrow=c(1,1))
```

rcorrp.cens *Rank Correlation for Paired Predictors with a Censored Response*

Description

Computes U-statistics to test for whether predictor X1 is more concordant than predictor X2, extending rcorr.cens. For method=1, estimates the fraction of pairs for which the x1 difference is more impressive than the x2 difference. For method=2, estimates the fraction of pairs for which x1 is concordant with S but x2 is not.

Usage

```
rcorrp.cens(x1, x2, S, outx=FALSE, method=1)
```

Arguments

`x1`	first predictor
`x2`	second predictor
`S`	a possibly right-censored `Surv` object. If `S` is a vector instead, it is converted to a `Surv` object and it is assumed that no observations are censored.
`outx`	set to T to exclude pairs tied on x1 or x2 from consideration
`method`	see above

Value

a vector of statistics

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

See Also

rcorr.cens, somers2, Surv

Examples

```
set.seed(1)
library(survival)

x1 <- rnorm(400)
x2 <- x1 + rnorm(400)
d.time <- rexp(400) + (x1 - min(x1))
cens   <- runif(400,.5,2)
death  <- d.time <= cens
d.time <- pmin(d.time, cens)
rcorrp.cens(x1, x2, Surv(d.time, death))
#rcorrp.cens(x1, x2, y) ## no censoring
```

---

rcspline.eval *Restricted Cubic Spline Design Matrix*

Description

Computes matrix that expands a single variable into the terms needed to fit a restricted cubic spline (natural spline) function using the truncated power basis. Two normalization options are given for somewhat reducing problems of ill-conditioning. The antiderivative function can be optionally created. If knot locations are not given, they will be estimated from the marginal distribution of `x`.

Usage

```
rcspline.eval(x, knots, nk=5, inclx=FALSE, knots.only=FALSE,
              type="ordinary", norm=2, rpm=NULL)
```

Arguments

`x`	a vector representing a predictor variable
`knots`	knot locations. If not given, knots will be estimated using default quantiles of `x`. For 3 knots, the outer quantiles used are .10 and .90. For 4-6 knots, the outer quantiles used are .05 and .95. For `nk>6`, the outer quantiles are .025 and .975. The knots are equally spaced between these on the quantile scale. For fewer than 100 non-missing values of `x`, the outer knots are the 5th smallest and largest `x`.
`nk`	number of knots. Default is 5. The minimum value is 3.
`inclx`	set to `TRUE` to add `x` as the first column of the returned matrix
`knots.only`	return the estimated knot locations but not the expanded matrix
`type`	`"ordinary"` to fit the function, `"integral"` to fit its anti-derivative.

`norm`	`0` to use the terms as originally given by Devlin and Weeks (1986), `1` to normalize non-linear terms by the cube of the spacing between the last two knots, `2` to normalize by the square of the spacing between the first and last knots (the default). `norm=2` has the advantage of making all nonlinear terms be on the `x`-scale.
`rpm`	If given, any NAs in `x` will be replaced with the value `rpm` after estimating any knot locations.

Value

If `knots.only=TRUE`, returns a vector of knot locations. Otherwise returns a matrix with `x` (if `inclx=TRUE`) followed by `nk-2` nonlinear terms. The matrix has an attribute `knots` which is the vector of knots used.

References

Devlin TF and Weeks BJ (1986): Spline functions for logistic regression modeling. Proc 11th Annual SAS Users Group Intnl Conf, p. 646–651. Cary NC: SAS Institute, Inc.

See Also

`ns`, `rcspline.restate`, `rcs`

Examples

```
x <- 1:100
rcspline.eval(x, nk=4, inclx=TRUE)
#lrm.fit(rcspline.eval(age,nk=4,inclx=TRUE), death)
```

`rcspline.plot` *Plot Restricted Cubic Spline Function*

Description

Provides plots of the estimated restricted cubic spline function relating a single predictor to the response for a logistic or Cox model. The `rcspline.plot` function does not allow for interactions as do `lrm` and `cph`, but it can provide detailed output for checking spline fits. This function uses the `rcspline.eval`, `lrm.fit`, and Therneau's `coxph.fit` functions and plots the estimated spline regression and confidence limits, placing summary statistics on the graph. If there are no adjustment variables, `rcspline.plot` can also plot two alternative estimates of the regression function when `model="logistic"`: proportions or logit proportions on grouped data, and a nonparametric estimate. The nonparametric regression estimate is based on smoothing the binary responses and taking the logit transformation of the smoothed estimates, if desired. The smoothing uses `supsmu`.

Usage

```
rcspline.plot(x,y,model=c("logistic", "cox", "ols"),xrange,event,nk=5,knots=NULL,
              show=c("xbeta","prob"),adj=NULL,xlab,ylab,ylim,plim=c(0,1),plotcl=TRUE
              showknots=TRUE,add=FALSE,subset,lty=1,noprint=FALSE,m,smooth=FALSE,bas
              main="auto",statloc)
```

Arguments

`x`	a numeric predictor
`y`	a numeric response. For binary logistic regression, `y` should be `0-1`.
`model`	`"logistic"` or `"cox"`. For `"cox"`, uses the `coxph.fit` with `method="efron"`. function.
`xrange`	range for evaluating `x`, default is `f` and `1-f` quantiles of `x`, where `f=10/max(n,200)`
`event`	event/censoring indicator if `model="cox"`. If `event` is present, `model` is assumed to be `"cox"`
`nk`	number of knots
`knots`	knot locations, default based on quantiles of `x` (by `rcspline.eval`)
`show`	`"xbeta"` or `"prob"` - what is plotted on `y`-axis
`adj`	optional matrix of adjustment variables
`xlab`	`x`-axis label, default is `"label"` attribute of `x`
`ylab`	same for `y`
`ylim`	`y`-axis limits for logit or log hazard
`plim`	`y`-axis limits for probability scale
`plotcl`	plot confidence limits
`showknots`	show knot locations with arrows
`add`	add this plot to an already existing plot
`subset`	subset of observations to process, e.g. `subset=sex=="male"`
`lty`	line type for plotting estimated spline function
`noprint`	suppress printing regression coefficients and standard errors
`m`	for `model="logistic"`, plot grouped estimates with triangles. Each group contains `m` ordered observations on `x`.
`smooth`	plot nonparametric estimate if `model="logistic"` and `adj` is not specified
`bass`	smoothing parameter (see `supsmu`)
`main`	main title, default is e.g. `"Estimated Spline Transformation"`
`statloc`	location of summary statistics. Default positioning by clicking left mouse button where upper left corner of statistics should appear. Alternative is `"ll"` to place below the graph on the lower left, or the actual `x` and `y` coordinates. Use `"none"` to suppress statistics.

Value

list with components `knots, x, xbeta, lower, upper` which are respectively the knot locations, design matrix, linear predictor, and lower and upper confidence limits

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`lrm`, `cph`, `rcspline.eval`, `plot`, `supsmu`, `coxph.fit`, `lrm.fit`

Examples

```
#rcspline.plot(cad.dur, tvdlm, m=150)
#rcspline.plot(log10(cad.dur+1), tvdlm, m=150)
```

`rcspline.restate` *Re-state Restricted Cubic Spline Function*

Description

This function re-states a restricted cubic spline function in the un-linearly-restricted form. Coefficients for that form are returned, along with an S functional representation of this function and a LaTeX character representation of the function. `rcsplineFunction` is a fast function that creates a function to compute a restricted cubic spline function with given coefficients and knots, without reformatting the function to be pretty (i.e., into unrestricted form).

Usage

```
rcspline.restate(knots, coef,
                 type=c("ordinary","integral"),
                 x="X", lx=nchar(x),
                 norm=2, columns=65, before="& &", after="\",
                 begin="", nbegin=0, digits=max(8, .Options$digits))

rcsplineFunction(knots, coef, norm=2)
```

Arguments

`knots`	vector of knots used in the regression fit
`coef`	vector of coefficients from the fit. If the length of `coef` is `k-1`, where `k=length(knots)`, the first coefficient must be for the linear term and remaining `k-2` coefficients must be for the constructed terms (e.g., from `rcspline.eval`). If the length of `coef` is `k`, an intercept is assumed to be in the first element (or a zero is prepended to `coef` for `rcsplineFunction`).
`type`	The default is to represent the cubic spline function corresponding to the coefficients and knots. Set `type="integral"` to instead represent its anti-derivative.

`x`	a character string to use as the variable name in the LaTeX expression for the formula.
`lx`	length of `x` to count with respect to `columns`. Default is length of character string contained by `x`. You may want to set `lx` smaller than this if it includes non-printable LaTeX commands.
`norm`	normalization that was used in deriving the original nonlinear terms used in the fit. See `rcspline.eval` for definitions.
`columns`	maximum number of symbols in the LaTeX expression to allow before inserting a newline (\\\\) command. Set to a very large number to keep text all on one line.
`before`	text to place before each line of LaTeX output. Use `"& &"` for an equation array environment in LaTeX where you want to have a left-hand prefix e.g. `f(X) & = &` or using `\lefteqn`.
`after`	text to place at the end of each line of output.
`begin`	text with which to start the first line of output. Useful when adding LaTeX output to part of an existing formula
`nbegin`	number of columns of printable text in `begin`
`digits`	number of significant digits to write for coefficients and knots

Value

`rcspline.restate` returns a vector of coefficients. The coefficients are un-normalized and two coefficients are added that are linearly dependent on the other coefficients and knots. The vector of coefficients has four attributes. `knots` is a vector of knots, `latex` is a vector of text strings with the LaTeX representation of the formula. `columns.used` is the number of columns used in the output string since the last newline command. `function` is an S function, which is also return in character string format as the `text` attribute. `rcsplineFunction` returns an S function with arguments `x` (a user-supplied numeric vector at which to evaluate the function), and some automatically-supplied other arguments.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`rcspline.eval`, `ns`, `rcs`, `latex`, `Function.transcan`

Examples

```
set.seed(1)
x <- 1:100
y <- (x - 50)^2 + rnorm(100, 0, 50)
plot(x, y)
xx <- rcspline.eval(x, inclx=TRUE, nk=4)
knots <- attr(xx, "knots")
```

```
coef <- lsfit(xx, y)$coef
options(digits=4)
# rcspline.restate must ignore intercept
w <- rcspline.restate(knots, coef[-1], x="{\\rm BP}")
# could also have used coef instead of coef[-1], to include intercept
cat(attr(w,"latex"), sep="\n")

xtrans <- eval(attr(w, "function"))
# This is an S function of a single argument
lines(x, coef[1] + xtrans(x), type="l")
# Plots fitted transformation

xtrans <- rcsplineFunction(knots, coef)
xtrans
lines(x, xtrans(x), col='blue')

#x <- blood.pressure
xx.simple <- cbind(x, pmax(x-knots[1],0)^3, pmax(x-knots[2],0)^3,
                       pmax(x-knots[3],0)^3, pmax(x-knots[4],0)^3)
pred.value <- coef[1] + xx.simple %*% w
plot(x, pred.value, type='l')   # same as above
```

---

reShape *Reshape Matrices and Serial Data*

Description

If the first argument is a matrix, `reShape` strings out its values and creates row and column vectors specifying the row and column each element came from. This is useful for sending matrices to Trellis functions, for analyzing or plotting results of `table` or `crosstabs`, or for reformatting serial data stored in a matrix (with rows representing multiple time points) into vectors. The number of observations in the new variables will be the product of the number of rows and number of columns in the input matrix. If the first argument is a vector, the `id` and `colvar` variables are used to restructure it into a matrix, with NAs for elements that corresponded to combinations of `id` and `colvar` values that did not exist in the data. When more than one vector is given, multiple matrices are created. This is useful for restructuring irregular serial data into regular matrices. It is also useful for converting data produced by `expand.grid` into a matrix (see the last example). The number of rows of the new matrices equals the number of unique values of `id`, and the number of columns equals the number of unique values of `colvar`.

When the first argument is a vector and the `id` is a data frame (even with only one variable), `reShape` will produce a data frame, and the unique groups are identified by combinations of the values of all variables in `id`. If a data frame `constant` is specified, the variables in this data frame are assumed to be constant within combinations of `id` variables (if not, an arbitrary observation in `constant` will be selected for each group). A row of `constant` corresponding to the target `id` combination is then carried along when creating the data frame result.

A different behavior of `reShape` is achieved when `base` and `reps` are specified. In that case `x` must be a list or data frame, and those data are assumed to contain one or more non-repeating measurements (e.g., baseline measurements) and one or more repeated measurements represented by variables named by pasting together the character strings in the vector `base` with the integers 1, 2, . . ., `reps`. The input data are rearranged by repeating each value of the baseline variables `reps` times and by transposing each observation's values of one of the set of repeated measurements as `reps` observations under the variable whose name does not have an integer pasted to the end. if `x` has a `row.names` attribute, those observation identifiers are each repeated `reps` times in the output object. See the last example.

Usage

```
reShape(x, ..., id, colvar, base, reps, times=1:reps,
        timevar='seqno', constant=NULL)
```

Arguments

`x`	a matrix or vector, or, when `base` is specified, a list or data frame
`...`	other optional vectors, if `x` is a vector
`id`	A numeric, character, category, or factor variable containing subject identifiers, or a data frame of such variables that in combination form groups of interest. Required if `x` is a vector, ignored otherwise.
`colvar`	A numeric, character, category, or factor variable containing column identifiers. `colvar` is using a "time of data collection" variable. Required if `x` is a vector, ignored otherwise.
`base`	vector of character strings containing base names of repeated measurements
`reps`	number of times variables named in `base` are repeated. This must be a constant.
`times`	when `base` is given, `times` is the vector of times to create if you do not want to use consecutive integers beginning with 1.
`timevar`	specifies the name of the time variable to create if `times` is given, if you do not want to use `seqno`
`constant`	a data frame with the same number of rows in `id` and `x`, containing auxiliary information to be merged into the resulting data frame. Logically, the rows of `constant` within each group should have the same value of all of its variables.

Details

In converting `dimnames` to vectors, the resulting variables are numeric if all elements of the matrix dimnames can be converted to numeric, otherwise the corresponding row or column variable remains character. When the `dimnames` if `x` have a `names` attribute, those two names become the new variable names. If `x` is a vector and another vector is also given (in `...`), the matrices in the resulting list are named the same as the input vector calling arguments. You can specify customized names for these on-the-fly by using e.g. `reShape(X=x, Y=y, id= , colvar= )`. The new names will then be `X` and `Y` instead of `x` and `y`. A new variable named `seqnno` is also added to the resulting object. `seqno` indicates the sequential repeated measurement number. When `base` and `times` are specified, this new variable is named the character value of `timevar` and the values are given by a table lookup into the vector `times`.

Value

If `x` is a matrix, returns a list containing the row variable, the column variable, and the `as.vector(x)` vector, named the same as the calling argument was called for `x`. If `x` is a vector and no other vectors were specified as `...`, the result is a matrix. If at least one vector was given to `...`, the result is a list containing `k` matrices, where `k` one plus the number of vectors in `...`. If `x` is a list or data frame, the same type of object is returned. If `x` is a vector and `id` is a data frame, a data frame will be the result.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
⟨f.harrell@vanderbilt.edu⟩

See Also

`reshape`, `as.vector`, `matrix`, `dimnames`, `outer`, `table`

Examples

```
set.seed(1)
Solder  <- factor(sample(c('Thin','Thick'),200,TRUE),c('Thin','Thick'))
Opening <- factor(sample(c('S','M','L'),  200,TRUE),c('S','M','L'))

tab <- table(Opening, Solder)
tab
reShape(tab)
# attach(tab)  # do further processing

#if(!.R.) {
# g <- crosstabs( ~ Solder + Opening, data = solder, subset = skips > 10)
# rowpct <- 100*attr(g,'marginals')$"N/RowTotal"   # compute row pcts
# rowpct
#
#
# r <- reShape(rowpct)
# # note names "Solder" and "Opening" came originally from formula
# # given to crosstabs
# r
# dotplot(Solder ~ rowpct, groups=Opening, panel=panel.superpose, data=r)
#}

# An example where a matrix is created from irregular vectors
follow <- data.frame(id=c('a','a','b','b','b','d'),
                     month=c(1, 2,  1,  2,  3,  2),
                     cholesterol=c(225,226, 320,319,318, 270))
follow
attach(follow)
reShape(cholesterol, id=id, colvar=month)
detach('follow')
```

```
# Could have done :
# reShape(cholesterol, triglyceride=trig, id=id, colvar=month)

# Create a data frame, reshaping a long dataset in which groups are
# formed not just by subject id but by combinations of subject id and
# visit number.  Also carry forward a variable that is supposed to be
# constant within subject-visit number combinations.  In this example,
# it is not constant, so an arbitrary visit number will be selected.
w <- data.frame(id=c('a','a','a','a','b','b','b','d','d','d'),
             visit=c(  1,  1,  2,  2,  1,  1,  2,  2,  2,  2),
                 k=c('A','A','B','B','C','C','D','E','F','G'),
               var=c('x','y','x','y','x','y','y','x','y','z'),
               val=1:10)
with(w,
     reShape(val, id=data.frame(id,visit),
             constant=data.frame(k), colvar=var))

# Get predictions from a regression model for 2 systematically
# varying predictors.  Convert the predictions into a matrix, with
# rows corresponding to the predictor having the most values, and
# columns corresponding to the other predictor
# d <- expand.grid(x2=0:1, x1=1:100)
# pred <- predict(fit, d)
# reShape(pred, id=d$x1, colvar=d$x2)  # makes 100 x 2 matrix

# Reshape a wide data frame containing multiple variables representing
# repeated measurements (3 repeats on 2 variables; 4 subjects)
set.seed(33)
n <- 4
w <- data.frame(age=rnorm(n, 40, 10),
                sex=sample(c('female','male'), n,TRUE),
                sbp1=rnorm(n, 120, 15),
                sbp2=rnorm(n, 120, 15),
                sbp3=rnorm(n, 120, 15),
                dbp1=rnorm(n,  80, 15),
                dbp2=rnorm(n,  80, 15),
                dbp3=rnorm(n,  80, 15), row.names=letters[1:n])
options(digits=3)
w

u <- reShape(w, base=c('sbp','dbp'), reps=3)
u
reShape(w, base=c('sbp','dbp'), reps=3, timevar='week', times=c(0,3,12))
```

redun *Redundancy Analysis*

Description

Uses flexible parametric additive models (see areg and its use of regression splines) to determine how well each variable can be predicted from the remaining variables. Variables are dropped in a stepwise fashion, removing the most predictable variable at each step. The remaining variables are used to predict. The process continues until no variable still in the list of predictors can be predicted with an R^2 or adjusted R^2 of at least `r2` or until dropping the variable with the highest R^2 (adjusted or ordinary) would cause a variable that was dropped earlier to no longer be predicted at least at the `r2` level from the now smaller list of predictors.

Usage

```
redun(formula, data=NULL, subset=NULL, r2 = 0.9,
      type = c("ordinary", "adjusted"), nk = 3, tlinear = TRUE,
      allcat=FALSE, minfreq=0, pr = FALSE, ...)
## S3 method for class 'redun':
print(x, digits=3, long=TRUE, ...)
```

Arguments

`formula`	a formula. Enclose a variable in `I()` to force linearity.
`data`	a data frame
`subset`	usual subsetting expression
`r2`	ordinary or adjusted R^2 cutoff for redundancy
`type`	specify `"adjusted"` to use adjusted R^2
`nk`	number of knots to use for continuous variables. Use `nk=0` to force linearity for all variables.
`tlinear`	set to `FALSE` to allow a variable to be automatically nonlinearly transformed (see `areg`) while being predicted. By default, only continuous variables on the right hand side (i.e., while they are being predictors) are automatically transformed, using regression splines. Estimating transformations for target (dependent) variables causes more overfitting than doing so for predictors.
`allcat`	set to `TRUE` to ensure that all categories of categorical variables having more than two categories are redundant (see details below)
`minfreq`	For a binary or categorical variable, there must be at least two categories with at least `minfreq` observations or the variable will be dropped and not checked for redundancy against other variables. `minfreq` also specifies the minimum frequency of a category or its complement before that category is considered when `allcat=TRUE`.
`pr`	set to `TRUE` to monitor progress of the stepwise algorithm
`...`	arguments to pass to `dataframeReduce` to remove "difficult" variables from `data` if `formula` is `~.` to use all variables in `data` (`data` must be specified when these arguments are used). Ignored for `print`.
`x`	an object created by `redun`
`digits`	number of digits to which to round R^2 values when printing
`long`	set to `FALSE` to prevent the `print` method from printing the R^2 history and the original R^2 with which each variable can be predicted from ALL other variables.

Details

A categorical variable is deemed redundant if a linear combination of dummy variables representing it can be predicted from a linear combination of other variables. For example, if there were 4 cities in the data and each city's rainfall was also present as a variable, with virtually the same rainfall reported for all observations for a city, city would be redundant given rainfall (or vice-versa; the one declared redundant would be the first one in the formula). If two cities had the same rainfall, `city` might be declared redundant even though tied cities might be deemed non-redundant in another setting. To ensure that all categories may be predicted well from other variables, use the `allcat` option. To ignore categories that are too infrequent or too frequent, set `minfreq` to a nonzero integer. When the number of observations in the category is below this number or the number of observations not in the category is below this number, no attempt is made to predict observations being in that category individually for the purpose of redundancy detection.

Value

an object of class `"redun"`

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`areg`, `dataframeReduce`, `transcan`, `varclus`

Examples

```
set.seed(1)
n <- 100
x1 <- runif(n)
x2 <- runif(n)
x3 <- x1 + x2 + runif(n)/10
x4 <- x1 + x2 + x3 + runif(n)/10
x5 <- factor(sample(c('a','b','c'),n,replace=TRUE))
x6 <- 1*(x5=='a' | x5=='c')
redun(~x1+x2+x3+x4+x5+x6, r2=.8)
redun(~x1+x2+x3+x4+x5+x6, r2=.8, minfreq=40)
redun(~x1+x2+x3+x4+x5+x6, r2=.8, allcat=TRUE)
# x5 is no longer redundant but x6 is
```

reorder.factor *Reorder Factor Levels*

Description

Reorders the levels of a factor variable by the values or the summarized values of another variable

Usage

```
reorder.factor(x, v, FUN = mean, ...)
```

Arguments

`x`	a factor variable
`v`	a numeric variable the same length as `x`
`FUN`	a statistical summarization function applied to `v` by levels of `x`
`...`	other arguments passed to `FUN`

Value

a new factor vector

See Also

`factor`

Examples

```
x <- factor(c('a','b','b','c'))
v <- c(3,-1,1,-5)
w <- reorder.factor(x, v)  # uses FUN=mean
w
levels(w)
class(w)
```

requirePackage *require a packge and error if package is not installed*

Description

require a package. If package doesn't exist then throw an error.

Usage

```
requirePackage(package, character.only = FALSE, ...)
```

Arguments

`package`	character vector containing the names of packages to load.
`character.only`	a logical indicating whether 'package' can be assumed to be character string.
`...`	arguments to be passed to require.

Author(s)

Charles Dupont

See Also

`library`, `require`

Examples

```
## Not run:
requirePackage(methods)
## End(Not run)
```

`rlegend` *Special Version of legend for R*

Description

`rlegend` is a version of `legend` for R that implements `plot=FALSE`, adds `grid=TRUE`, and defaults `lty, lwd, pch` to `NULL` and checks for `length>0` rather than `missing()`, so it's easier to deal with non-applicable parameters. But when `grid` is in effect, the preferred function to use is `rlegendg`, which calls the `lattice` `draw.key` function.

Usage

```
rlegend(x, y, legend, fill, col = "black", lty = NULL, lwd = NULL, pch = NULL, angl

rlegendg(x, y, legend, col=pr$col[1], lty=NULL,
         lwd=NULL, pch=NULL, cex=pr$cex[1], other=NULL)
```

Arguments

`x`

`y`

`legend`

`fill`

`col`

`lty`

`lwd`	
`pch`	
`angle`	
`density`	
`bty`	
`bg`	
`pt.bg`	
`cex`	
`xjust`	
`yjust`	
`x.intersp`	
`y.intersp`	
`adj`	
`text.width`	
`merge`	
`trace`	
`ncol`	
`horiz`	see `legend`
`plot`	set to `FALSE` to suppress drawing the legend. This is used the compute the size needed for when the legend is drawn with a later call to `rlegend`.
`grid`	set to `TRUE` if the `grid` package is in effect
`...`	see `legend`
`other`	a list containing other arguments to pass to `draw.key`. See the help file for `xyplot`.

Value

a list with elements `rect` and `text`. `rect` has elements `w, h, left, top` with size/position information.

Author(s)

Frank Harrell and R-Core

See Also

`legend`,`draw.key`

rm.boot	*Bootstrap Repeated Measurements Model*

Description

For a dataset containing a time variable, a scalar response variable, and an optional subject identification variable, obtains least squares estimates of the coefficients of a restricted cubic spline function or a linear regression in time after adjusting for subject effects through the use of subject dummy variables. Then the fit is bootstrapped `B` times, either by treating time and subject ID as fixed (i.e., conditioning the analysis on them) or as random variables. For the former, the residuals from the original model fit are used as the basis of the bootstrap distribution. For the latter, samples are taken jointly from the time, subject ID, and response vectors to obtain unconditional distributions.

If a subject `id` variable is given, the bootstrap sampling will be based on samples with replacement from subjects rather than from individual data points. In other words, either none or all of a given subject's data will appear in a bootstrap sample. This cluster sampling takes into account any correlation structure that might exist within subjects, so that confidence limits are corrected for within-subject correlation. Assuming that ordinary least squares estimates, which ignore the correlation structure, are consistent (which is almost always true) and efficient (which would not be true for certain correlation structures or for datasets in which the number of observation times vary greatly from subject to subject), the resulting analysis will be a robust, efficient repeated measures analysis for the one-sample problem.

Predicted values of the fitted models are evaluated by default at a grid of 100 equally spaced time points ranging from the minimum to maximum observed time points. Predictions are for the average subject effect. Pointwise confidence intervals are optionally computed separately for each of the points on the time grid. However, simultaneous confidence regions that control the level of confidence for the entire regression curve lying within a band are often more appropriate, as they allow the analyst to draw conclusions about nuances in the mean time response profile that were not stated apriori. The method of Tibshirani (1997) is used to easily obtain simultaneous confidence sets for the set of coefficients of the spline or linear regression function as well as the average intercept parameter (over subjects). Here one computes the objective criterion (here both the -2 log likelihood evaluated at the bootstrap estimate of beta but with respect to the original design matrix and response vector, and the sum of squared errors in predicting the original response vector) for the original fit as well as for all of the bootstrap fits. The confidence set of the regression coefficients is the set of all coefficients that are associated with objective function values that are less than or equal to say the 0.95 quantile of the vector of `B + 1` objective function values. For the coefficients satisfying this condition, predicted curves are computed at the time grid, and minima and maxima of these curves are computed separately at each time point to derive the final simultaneous confidence band.

By default, the log likelihoods that are computed for obtaining the simultaneous confidence band assume independence within subject. This will cause problems unless such log likelihoods have very high rank correlation with the log likelihood allowing for dependence. To allow for correlation or to estimate the correlation function, see the `cor.pattern` argument below.

Usage

```
rm.boot(time, y, id=seq(along=time), subset,
        plot.individual=FALSE,
        bootstrap.type=c('x fixed','x random'),
        nk=6, knots, B=500, smoother=supsmu,
        xlab, xlim, ylim=range(y),
        times=seq(min(time),max(time),length=100),
        absorb.subject.effects=FALSE,
        rho=0, cor.pattern=c('independent','estimate'), ncor=10000,
        ...)

## S3 method for class 'rm.boot':
plot(x, obj2, conf.int=.95,
     xlab=x$xlab, ylab=x$ylab,
     xlim, ylim=x$ylim,
     individual.boot=FALSE,
     pointwise.band=FALSE,
     curves.in.simultaneous.band=FALSE,
     col.pointwise.band=2,
     objective=c('-2 log L','sse','dep -2 log L'), add=FALSE, ncurves,
     multi=FALSE, multi.method=c('color','density'),
     multi.conf   =c(.05,.1,.2,.3,.4,.5,.6,.7,.8,.9,.95,.99),
     multi.density=c( -1,90,80,70,60,50,40,30,20,10,  7,  4),
     multi.col    =c(  1, 8,20, 5, 2, 7,15,13,10,11,  9, 14),
     subtitles=TRUE, ...)
```

Arguments

`time`	numeric time vector
`y`	continuous numeric response vector of length the same as `time`. Subjects having multiple measurements have the measurements strung out.
`x`	an object returned from `rm.boot`
`id`	subject ID variable. If omitted, it is assumed that each time-response pair is measured on a different subject.
`subset`	subset of observations to process if not all the data
`plot.individual`	set to `TRUE` to plot nonparametrically smoothed time-response curves for each subject
`bootstrap.type`	specifies whether to treat the time and subject ID variables as fixed or random
`nk`	number of knots in the restricted cubic spline function fit. The number of knots may be 0 (denoting linear regression) or an integer greater than 2 in which `k` knots results in `k-1` regression coefficients excluding the intercept. The default is 6 knots.
`knots`	vector of knot locations. May be specified if `nk` is omitted.
`B`	number of bootstrap repetitions. Default is 500.

`smoother`	a smoothing function that is used if `plot.individual=TRUE`. Default is `supsmu`.
`xlab`	label for x-axis. Default is `"units"` attribute of the original `time` variable, or `"Time"` if no such attribute was defined using the `units` function.
`xlim`	specifies x-axis plotting limits. Default is to use range of times specified to `rm.boot`.
`ylim`	for `rm.boot` this is a vector of y-axis limits used if `plot.individual=TRUE`. It is also passed along for later use by `plot.rm.boot`. For `plot.rm.boot`, `ylim` can be specified, to override the value stored in the object stored by `rm.boot`. The default is the actual range of `y` in the input data.
`times`	a sequence of times at which to evaluated fitted values and confidence limits. Default is 100 equally spaced points in the observed range of `time`.
`absorb.subject.effects`	If `TRUE`, adjusts the response vector `y` before re-sampling so that the subject-specific effects in the initial model fit are all zero. Then in re-sampling, subject effects are not used in the models. This will downplay one of the sources of variation. This option is used mainly for checking for consistency of results, as the re-sampling analyses are simpler when `absort.subject.effects=TRUE`.
`rho`	The log-likelihood function that is used as the basis of simultaneous confidence bands assumes normality with independence within subject. To check the robustness of this assumption, if `rho` is not zero, the log-likelihood under multivariate normality within subject, with constant correlation `rho` between any two time points, is also computed. If the two log-likelihoods have the same ranks across re-samples, alllowing the correlation structure does not matter. The agreement in ranks is quantified using the Spearman rank correlation coefficient. The `plot` method allows the non-zero intra-subject correlation log-likelihood to be used in deriving the simultaneous confidence band. Note that this approach does assume homoscedasticity.
`cor.pattern`	More generally than using an equal-correlation structure, you can specify a function of two time vectors that generates as many correlations as the length of these vectors. For example, `cor.pattern=function(time1,time2).2^(abs(time1-time2)/10)` would specify a dampening serial correlation pattern. `cor.pattern` can also be a list containing vectors `x` (a vector of absolute time differences) and `y` (a corresponding vector of correlations). To estimate the correlation function as a function of absolute time differences within subjects, specify `cor.pattern="estimate"`. The products of all possible pairs of residuals (or at least up to `ncor` of them) within subjects will be related to the absolute time difference. The correlation function is estimated by computing the sample mean of the products of standardized residuals, stratified by absolute time difference. The correlation for a zero time difference is set to 1 regardless of the `lowess` estimate. NOTE: This approach fails in the presence of large subject effects; correcting for such effects removes too much of the correlation structure in the residuals.
`ncor`	the maximum number of pairs of time values used in estimating the correlation function if `cor.pattern="estimate"`
`...`	other arguments to pass to `smoother` if `plot.individual=TRUE`

`obj2`	a second object created by `rm.boot` that can also be passed to `plot.rm.boot`. This is used for two-sample problems for which the time profiles are allowed to differ between the two groups. The bootstrapped predicted y values for the second fit are subtracted from the fitted values for the first fit so that the predicted mean response for group 1 minus the predicted mean response for group 2 is what is plotted. The confidence bands that are plotted are also for this difference. For the simultaneous confidence band, the objective criterion is taken to be the sum of the objective criteria (-2 log L or sum of squared errors) for the separate fits for the two groups. The `times` vectors must have been identical for both calls to `rm.boot`, although `NA`s can be inserted by the user of one or both of the time vectors in the `rm.boot` objects so as to suppress certain sections of the difference curve from being plotted.
`conf.int`	the confidence level to use in constructing simultaneous, and optionally pointwise, bands. Default is `0.95`.
`ylab`	label for y-axis. Default is the `"label"` attribute of the original `y` variable, or `"y"` if no label was assigned to `y` (using the `label` function, for example).
`individual.boot`	set to `TRUE` to plot the first 100 bootstrap regression fits
`pointwise.band`	set to `TRUE` to draw a pointwise confidence band in addition to the simultaneous band
`curves.in.simultaneous.band`	set to `TRUE` to draw all bootstrap regression fits that had a sum of squared errors (obtained by predicting the original `y` vector from the original `time` vector and `id` vector) that was less that or equal to the `conf.int` quantile of all bootstrapped models (plus the original model). This will show how the point by point max and min were computed to form the simultaneous confidence band.
`col.pointwise.band`	color for the pointwise confidence band. Default is `2`, which defaults to red for default Windows S-PLUS setups.
`objective`	the default is to use the -2 log of the Gaussian likelihood for computing the simultaneous confidence region. If neither `cor.pattern` nor `rho` was specified to `rm.boot`, the independent homoscedastic Gaussian likelihood is used. Otherwise the dependent homoscedastic likelihood is used according to the specified or estimated correlation pattern. Specify `objective="sse"` to instead use the sum of squared errors.
`add`	set to `TRUE` to add curves to an existing plot. If you do this, titles and subtitles are omitted.
`ncurves`	when using `individual.boot=TRUE` or `curves.in.simultaneous.band=TRUE`, you can plot a random sample of `ncurves` of the fitted curves instead of plotting up to `B` of them.
`multi`	set to `TRUE` to draw multiple simultaneous confidence bands shaded with different colors. Confidence levels vary over the values in the `multi.conf` vector.
`multi.method`	specifies the method of shading when `multi=TRUE`. Default is to use colors, with the default colors chosen so that when the graph is printed under S-Plus for Windows 4.0 to an HP LaserJet printer, the confidence regions are naturally

	ordered by darkness of gray-scale. Regions closer to the point estimates (i.e., the center) are darker. Specify `multi.method="density"` to instead use densities of lines drawn per inch in the confidence regions, with all regions drawn with the default color. The `polygon` function is used to shade the regions.
`multi.conf`	vector of confidence levels, in ascending order. Default is to use 12 confidence levels ranging from 0.05 to 0.99.
`multi.density`	vector of densities in lines per inch corresponding to `multi.conf`. As is the convention in the `polygon` function, a density of `-1` indicates a solid region.
`multi.col`	vector of colors corresponding to `multi.conf`. See `multi.method` for rationale.
`subtitles`	set to `FALSE` to suppress drawing subtitles for the plot

Details

Observations having missing `time` or `y` are excluded from the analysis.

As most repeated measurement studies consider the times as design points, the fixed covariable case is the default. Bootstrapping the residuals from the initial fit assumes that the model is correctly specified. Even if the covariables are fixed, doing an unconditional bootstrap is still appropriate, and for large sample sizes unconditional confidence intervals are only slightly wider than conditional ones. For moderate to small sample sizes, the `"x random"` method can be fairly conservative.

If not all subjects have the same number of observations (after deleting observations containing missing values) and if `bootstrap.type="x fixed"`, bootstrapped residual vectors may have a length `m` that is different from the number of original observations `n`. If `m > n` for a bootstrap repetition, the first `n` elements of the randomly drawn residuals are used. If `m < n`, the residual vector is appended with a random sample with replacement of length `n - m` from itself. A warning message is issued if this happens. If the number of time points per subject varies, the bootstrap results for `"x fixed"` can still be invalid, as this method assumes that a vector (over subjects) of all residuals can be added to the original yhats, and varying number of points will cause misalignment.

For `bootstrap.type="x random"` in the presence of significant subject effects, the analysis is approximate as the subjects used in any one bootstrap fit will not be the entire list of subjects. The average (over subjects used in the bootstrap sample) intercept is used from that bootstrap sample as a predictor of average subject effects in the overall sample.

Once the bootstrap coefficient matrix is stored by `rm.boot`, `plot.rm.boot` can be run multiple times with different options (e.g, different confidence levels).

See `bootcov` in the `Design` library for a general approach to handling repeated measurement data for ordinary linear models, binary and ordinal models, and survival models, using the unconditional bootstrap. `bootcov` does not handle bootstrapping residuals.

Value

an object of class `rm.boot` is returned by `rm.boot`. The principal object stored in the returned object is a matrix of regression coefficients for the original fit and all of the bootstrap repetitions (object `Coef`), along with vectors of the corresponding -2 log likelihoods are sums of squared errors. The original fit object from `lm.fit.qr` is stored in `fit`. For this fit, a cell means model is used for the `id` effects.

`plot.rm.boot` returns a list containing the vector of times used for plotting along with the overall fitted values, lower and upper simultaneous confidence limits, and optionally the pointwise confidence limits.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Feng Z, McLerran D, Grizzle J (1996): A comparison of statistical methods for clustered data analysis with Gaussian error. Stat in Med 15:1793–1806.

Tibshirani R, Knight K (1997):Model search and inference by bootstrap "bumping". Technical Report, Department of Statistics, University of Toronto.
http://www-stat.stanford.edu/ tibs. Presented at the Joint Statistical Meetings, Chicago, August 1996.

Efron B, Tibshirani R (1993): An Introduction to the Bootstrap. New York: Chapman and Hall.

Diggle PJ, Verbyla AP (1998): Nonparametric estimation of covariance structure in logitudinal data. Biometrics 54:401–415.

Chapman IM, Hartman ML, et al (1997): Effect of aging on the sensitivity of growth hormone secretion to insulin-like growth factor-I negative feedback. J Clin Endocrinol Metab 82:2996–3004.

See Also

`rcspline.eval`, `lm`, `lowess`, `supsmu`, `bootcov`, `units`, `label`, `polygon`, `reShape`

Examples

```
# Generate multivariate normal responses with equal correlations (.7)
# within subjects and no correlation between subjects
# Simulate realizations from a piecewise linear population time-response
# profile with large subject effects, and fit using a 6-knot spline
# Estimate the correlation structure from the residuals, as a function
# of the absolute time difference

# Function to generate n p-variate normal variates with mean vector u and
# covariance matrix S
# Slight modification of function written by Bill Venables
# See also the built-in function rmvnorm
mvrnorm <- function(n, p = 1, u = rep(0, p), S = diag(p)) {
  Z <- matrix(rnorm(n * p), p, n)
  t(u + t(chol(S)) %*% Z)
}

n      <- 20          # Number of subjects
sub    <- .5*(1:n)    # Subject effects
```

```
# Specify functional form for time trend and compute non-stochastic component
times <- seq(0, 1, by=.1)
g     <- function(times) 5*pmax(abs(times-.5),.3)
ey    <- g(times)

# Generate multivariate normal errors for 20 subjects at 11 times
# Assume equal correlations of rho=.7, independent subjects

nt    <- length(times)
rho   <- .7


set.seed(19)
errors <- mvrnorm(n, p=nt, S=diag(rep(1-rho,nt))+rho)
# Note:  first random number seed used gave rise to mean(errors)=0.24!

# Add E[Y], error components, and subject effects
y      <- matrix(rep(ey,n), ncol=nt, byrow=TRUE) + errors +
          matrix(rep(sub,nt), ncol=nt)

# String out data into long vectors for times, responses, and subject ID
y      <- as.vector(t(y))
times  <- rep(times, n)
id     <- sort(rep(1:n, nt))

# Show lowess estimates of time profiles for individual subjects
f <- rm.boot(times, y, id, plot.individual=TRUE, B=25, cor.pattern='estimate',
             smoother=lowess, bootstrap.type='x fixed', nk=6)
# In practice use B=400 or 500
# This will compute a dependent-structure log-likelihood in addition
# to one assuming independence.  By default, the dep. structure
# objective will be used by the plot method  (could have specified rho=.7)
# NOTE: Estimating the correlation pattern from the residual does not
# work in cases such as this one where there are large subject effects

# Plot fits for a random sample of 10 of the 25 bootstrap fits
plot(f, individual.boot=TRUE, ncurves=10, ylim=c(6,8.5))

# Plot pointwise and simultaneous confidence regions
plot(f, pointwise.band=TRUE, col.pointwise=1, ylim=c(6,8.5))

# Plot population response curve at average subject effect
ts <- seq(0, 1, length=100)
lines(ts, g(ts)+mean(sub), lwd=3)

## Not run:
#
# Handle a 2-sample problem in which curves are fitted
# separately for males and females and we wish to estimate the
# difference in the time-response curves for the two sexes.
# The objective criterion will be taken by plot.rm.boot as the
# total of the two sums of squared errors for the two models
#
```

```
knots <- rcspline.eval(c(time.f,time.m), nk=6, knots.only=TRUE)
# Use same knots for both sexes, and use a times vector that
# uses a range of times that is included in the measurement
# times for both sexes
#
tm <- seq(max(min(time.f),min(time.m)),
          min(max(time.f),max(time.m)),length=100)

f.female <- rm.boot(time.f, bp.f, id.f, knots=knots, times=tm)
f.male   <- rm.boot(time.m, bp.m, id.m, knots=knots, times=tm)
plot(f.female)
plot(f.male)
# The following plots female minus male response, with
# a sequence of shaded confidence band for the difference
plot(f.female,f.male,multi=TRUE)

# Do 1000 simulated analyses to check simultaneous coverage
# probability.  Use a null regression model with Gaussian errors

n.per.pt <- 30
n.pt     <- 10

null.in.region <- 0

for(i in 1:1000) {
  y    <- rnorm(n.pt*n.per.pt)
  time <- rep(1:n.per.pt, n.pt)
#  Add the following line and add ,id=id to rm.boot to use clustering
#  id   <- sort(rep(1:n.pt, n.per.pt))
#  Because we are ignoring patient id, this simulation is effectively
#  using 1 point from each of 300 patients, with times 1,2,3,,,30

  f <- rm.boot(time, y, B=500, nk=5, bootstrap.type='x fixed')
  g <- plot(f, ylim=c(-1,1), pointwise=FALSE)
  null.in.region <- null.in.region + all(g$lower<=0 & g$upper>=0)
  prn(c(i=i,null.in.region=null.in.region))
}

# Simulation Results: 905/1000 simultaneous confidence bands
# fully contained the horizontal line at zero
## End(Not run)
```

`samplesize.bin` *Sample Size for 2-sample Binomial*

Description

Computes sample size(s) for 2-sample binomial problem given vector or scalar probabilities in the two groups.

Usage

```
samplesize.bin(alpha, beta, pit, pic, rho=0.5)
```

Arguments

`alpha`	scalar ONE-SIDED test size, or two-sided size/2
`beta`	scalar or vector of powers
`pit`	hypothesized treatment probability of success
`pic`	hypothesized control probability of success
`rho`	proportion of the sample devoted to treated group (0 <rho < 1)

Value

TOTAL sample size(s)

AUTHOR

Rick Chappell

Dept. of Statistics and Human Oncology

University of Wisconsin at Madison

chappell@stat.wisc.edu

Examples

```
alpha <- .05
beta <- c(.70,.80,.90,.95)

# N1 is a matrix of total sample sizes whose
# rows vary by hypothesized treatment success probability and
# columns vary by power
# See Meinert's book for formulae.

N1 <- samplesize.bin(alpha, beta, pit=.55, pic=.5)
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.60, pic=.5))
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.65, pic=.5))
N1 <- rbind(N1, samplesize.bin(alpha, beta, pit=.70, pic=.5))
attr(N1,"dimnames") <- NULL

#Accounting for 5% noncompliance in the treated group
inflation <- (1/.95)**2
print(round(N1*inflation+.5,0))
```

`sasxport.get` *Enhanced Importing of SAS Transport Files using read.xport*

Description

Uses the `read.xport` and `lookup.xport` functions in the `foreign` library to import SAS datasets. SAS date, time, and date/time variables are converted respectively to `Date`, POSIX, or `chron` objects in R, variable names are converted to lower case, SAS labels are associated with variables, and (by default) integer-valued variables are converted from storage mode `double` to `integer`. If the user ran `PROC FORMAT CNTLOUT=` in SAS and included the resulting dataset in the SAS version 5 transport file, variables having customized formats that do not include any ranges (i.e., variables having standard `PROC FORMAT; VALUE` label formats) will have their format labels looked up, and these variables are converted to S `factors`.

For those users having access to SAS, `method='csv'` is preferred when importing several SAS datasets. Run SAS macro `exportlib.sas` available from `http://biostat.mc.vanderbilt.edu/twiki/pub/Main/Hmisc/exportlib.sas` to convert all SAS datasets in a SAS data library (from any engine supported by your system) into `CSV` files. If any customized formats are used, it is assumed that the `PROC FORMAT CNTLOUT=` dataset is in the data library as a regular SAS dataset, as above.

`SASdsLabels` reads a file containing `PROC CONTENTS` printed output to parse dataset labels, assuming that `PROC CONTENTS` was run on an entire library.

Usage

```
sasxport.get(file, force.single = TRUE,
             method=c('read.xport','dataload','csv'), formats=NULL, allow=NULL,
             out=NULL, keep=NULL, drop=NULL, as.is=0.5, FUN=NULL)
sasdsLabels(file)
```

Arguments

`file`	name of a file containing the SAS transport file. `file` may be a URL beginning with `http://`. For `sasdsLabels`, `file` is the name of a file containing a `PROC CONTENTS` output listing. For `method='csv'`, `file` is the name of the directory containing all the `CSV` files created by running the `exportlib` SAS macro.
`force.single`	set to `FALSE` to keep integer-valued variables not exceeding 2^31-1 in value from being converted to `integer` storage mode
`method`	set to `"dataload"` if you have the `dataload` executable installed and want to use it instead of `read.xport`. This seems to correct some errors in which rarely some factor variables are always missing when read by `read.xport` when in fact they have some non-missing values.
`formats`	a data frame or list (like that created by `read.xport`) containing `PROC FORMAT` output, if such output is not stored in the main transport file.

`allow`	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
`out`	a character string specifying a directory in which to write separate R `save` files (`.rda` files) for each regular dataset. Each file and the data frame inside it is named with the SAS dataset name translated to lower case and with underscores changed to periods. The default `NULL` value of `out` results in a data frame or a list of data frames being returned. When `out` is given, `sasxport.get` returns only metadata (see below), invisibly. `out` only works with `methods='csv'`. `out` should not have a trailing slash.
`keep`	a vector of names of SAS datasets to process (original SAS upper case names). Must include `PROC FORMAT` dataset if it exists, and if the kept datasets use any of its value label formats.
`drop`	a vector of names of SAS datasets to ignore (original SAS upper case names)
`as.is`	SAS character variables are converted to S factor objects if `as.is=FALSE` or if `as.is` is a number between 0 and 1 inclusive and the number of unique values of the variable is less than the number of observations (`n`) times `as.is`. The default if `as.is` is .5, so character variables are converted to factors only if they have fewer than `n/2` unique values. The primary purpose of this is to keep unique identification variables as character values in the data frame instead of using more space to store both the integer factor codes and the factor labels.
`FUN`	an optional function that will be run on each data frame created, when `method='csv'` and `out` are specified. The result of all the `FUN` calls is made into a list corresponding to the SAS datasets that are read. This list is the `FUN` attribute of the result returned by `sasxport.get`.

Details

See `contents.list` for a way to print the directory of SAS datasets when more than one was imported.

Value

If there is more than one dataset in the transport file other than the `PROC FORMAT` file, the result is a list of data frames containing all the non-`PROC FORMAT` datasets. Otherwise the result is the single data frame. There is an exception if `out` is specified; that causes separate R `save` files to be written and the returned value to be a list corresponding to the SAS datasets, with key `PROC CONTENTS` information in a data frame making up each part of the list. `sasdsLabels` returns a named vector of dataset labels, with names equal to the dataset names.

Author(s)

Frank E Harrell Jr

See Also

`read.xport`,`label`,`sas.get`, `Dates`,`DateTimeClasses`,`chron`, `lookup.xport`,`contents`,`describe`

Examples

```
## Not run:
# SAS code to generate test dataset:
# libname y SASV5XPT "test2.xpt";
#
# PROC FORMAT; VALUE race 1=green 2=blue 3=purple; RUN;
# PROC FORMAT CNTLOUT=format;RUN;  * Name, e.g. 'format', unimportant;
# data test;
# LENGTH race 3 age 4;
# age=30; label age="Age at Beginning of Study";
# race=2;
# d1='3mar2002'd ;
# dt1='3mar2002 9:31:02'dt;
# t1='11:13:45't;
# output;
#
# age=31;
# race=4;
# d1='3jun2002'd ;
# dt1='3jun2002 9:42:07'dt;
# t1='11:14:13't;
# output;
# format d1 mmddyy10. dt1 datetime. t1 time. race race.;
# run;
# data z; LENGTH x3 3 x4 4 x5 5 x6 6 x7 7 x8 8;
#    DO i=1 TO 100;
#        x3=ranuni(3);
#        x4=ranuni(5);
#        x5=ranuni(7);
#        x6=ranuni(9);
#        x7=ranuni(11);
#        x8=ranuni(13);
#        output;
#        END;
#    DROP i;
#    RUN;
# PROC MEANS; RUN;
# PROC COPY IN=work OUT=y;SELECT test format z;RUN; *Creates test2.xpt;
w <- sasxport.get('test2.xpt')
# To use an existing copy of test2.xpt available on the web:
w <- sasxport.get('http://hesweb1.med.virginia.edu/biostat/s/data/sas/test2.xpt')

describe(w$test)   # see labels, format names for dataset test
# Note: if only one dataset (other than format) had been exported,
# just do describe(w) as sasxport.get would not create a list for that
lapply(w, describe)# see descriptive stats for both datasets
contents(w$test)   # another way to see variable attributes
lapply(w, contents)# show contents of both datasets
options(digits=7)  # compare the following matrix with PROC MEANS output
t(sapply(w$z, function(x)
 c(Mean=mean(x),SD=sqrt(var(x)),Min=min(x),Max=max(x))))
## End(Not run)
```

scat1d *One-Dimensional Scatter Diagram, Spike Histogram, or Density*

Description

scat1d adds tick marks (bar codes. rug plot) on any of the four sides of an existing plot, corresponding with non-missing values of a vector x. This is used to show the data density. Can also place the tick marks along a curve by specifying y-coordinates to go along with the x values.

If any two values of x are within eps*w of each other, where eps defaults to .001 and w is the span of the intended axis, values of x are jittered by adding a value uniformly distributed in [-jitfrac*w, jitfrac*w], where jitfrac defaults to .008. Specifying preserve=TRUE invokes jitter2 with a different logic of jittering. Allows plotting random sub-segments to handle very large x vectors (see tfrac).

jitter2 is a generic method for jittering, which does not add random noise. It retains unique values and ranks, and randomly spreads duplicate values at equidistant positions within limits of enclosing values. jitter2 is especially useful for numeric variables with discrete values, like rating scales. Missing values are allowed and are returned. Currently implemented methods are jitter2.default for vectors and jitter2.data.frame which returns a data.frame with each numeric column jittered.

datadensity is a generic method used to show data densities in more complex situations. In the Design library there is a datadensity method for use with plot.Design. Here, another datadensity method is defined for data frames. Depending on the which argument, some or all of the variables in a data frame will be displayed, with scat1d used to display continuous variables and, by default, bars used to display frequencies of categorical, character, or discrete numeric variables. For such variables, when the total length of value labels exceeds 200, only the first few characters from each level are used. By default, datadensity.data.frame will construct one axis (i.e., one strip) per variable in the data frame. Variable names appear to the left of the axes, and the number of missing values (if greater than zero) appear to the right of the axes. An optional group variable can be used for stratification, where the different strata are depicted using different colors. If the q vector is specified, the desired quantiles (over all groups) are displayed with solid triangles below each axis.

When the sample size exceeds 2000 (this value may be modified using the nhistSpike argument, datadensity calls histSpike instead of scat1d to show the data density for numeric variables. This results in a histogram-like display that makes the resulting graphics file much smaller. In this case, datadensity uses the minf argument (see below) so that very infrequent data values will not be lost on the variable's axis, although this will slightly distort the histogram.

histSpike is another method for showing a high-resolution data distribution that is particularly good for very large datasets (say n > 1000). By default, histSpike bins the continuous x variable into 100 equal-width bins and then computes the frequency counts within bins (if n does not exceed 10, no binning is done). If add=FALSE (the default), the function displays either proportions or frequencies as in a vertical histogram. Instead of bars, spikes are used to depict the frequencies. If add=FALSE, the function assumes you are adding small density displays that are intended to take up a small amount of space in the margins of the overall plot. The frac argument is used as with scat1d to determine the relative length of the whole plot that is used to represent the maximum frequency. No jittering is done by histSpike.

histSpike can also graph a kernel density estimate for x, or add a small density curve to any of 4 sides of an existing plot. When y or curve is specified, the density or spikes are drawn with respect to the curve rather than the x-axis.

Usage

```
scat1d(x, side=3, frac=0.02, jitfrac=0.008, tfrac,
       eps=ifelse(preserve,0,.001),
       lwd=0.1, col=par("col"),
       y=NULL, curve=NULL,
       bottom.align=FALSE,
       preserve=FALSE, fill=1/3, limit=TRUE, nhistSpike=2000, nint=100,
       type=c('proportion','count','density'), grid=FALSE, ...)

jitter2(x, ...)

## Default S3 method:
jitter2(x, fill=1/3, limit=TRUE, eps=0, presorted=FALSE, ...)

## S3 method for class 'data.frame':
jitter2(x, ...)

datadensity(object, ...)

## S3 method for class 'data.frame':
datadensity(object, group,
            which=c("all","continuous","categorical"),
            method.cat=c("bar","freq"),
            col.group=1:10,
            n.unique=10, show.na=TRUE, nint=1, naxes,
            q, bottom.align=nint>1,
            cex.axis=sc(.5,.3), cex.var=sc(.8,.3),
            lmgp=NULL, tck=sc(-.009,-.002),
            ranges=NULL, labels=NULL, ...)
# sc(a,b) means default to a if number of axes <= 3, b if >=50, use
# linear interpolation within 3-50

histSpike(x, side=1, nint=100, frac=.05, minf=NULL, mult.width=1,
          type=c('proportion','count','density'),
          xlim=range(x), ylim=c(0,max(f)), xlab=deparse(substitute(x)),
          ylab=switch(type,proportion='Proportion',
                           count     ='Frequency',
                           density   ='Density'),
          y=NULL, curve=NULL, add=FALSE,
          bottom.align=type=='density', col=par('col'), lwd=par('lwd'),
          grid=FALSE, ...)
```

Arguments

`x`	a vector of numeric data, or a data frame (for `jitter2`)
`object`	a data frame or list (even with unequal number of observations per variable, as long as `group` is not specified)
`side`	axis side to use (1=bottom (default for `histSpike`), 2=left, 3=top (default for `scat1d`), 4=right)
`frac`	fraction of smaller of vertical and horizontal axes for tick mark lengths. Can be negative to move tick marks outside of plot. For `histSpike`, this is the relative length to be used for the largest frequency. When `scat1d` calls `histSpike`, it multiplies its `frac` argument by 2.5.
`jitfrac`	fraction of axis for jittering. If <=0, no jittering is done. If `preserve=TRUE`, the amount of jittering is independent of jitfrac.
`tfrac`	fraction of tick mark to actually draw. If `tfrac<1`, will draw a random fraction `tfrac` of the line segment at each point. This is useful for very large samples or ones with some very dense points. The default value is 1 if the number of non-missing observations `n` is less than 125, and `max(.1, 125/n)` otherwise.
`eps`	fraction of axis for determining overlapping points in `x`. For `preserve=TRUE` the default is 0 and original unique values are retained, bigger values of eps tends to bias observations from dense to sparse regions, but ranks are still preserved.
`lwd`	line width for tick marks, passed to `segments`
`col`	color for tick marks, passed to `segments`
`y`	specify a vector the same length as `x` to draw tick marks along a curve instead of by one of the axes. The `y` values are often predicted values from a model. The `side` argument is ignored when `y` is given. If the curve is already represented as a table look-up, you may specify it using the `curve` argument instead. `y` may be a scalar to use a constant vertical placement.
`curve`	a list containing elements `x` and `y` for which linear interpolation is used to derive `y` values corresponding to values of `x`. This results in tick marks being drawn along the curve. For `histSpike`, interpolated `y` values are derived for bin midpoints.
`bottom.align`	set to `TRUE` to have the bottoms of tick marks (for `side=1` or `side=3`) aligned at the y-coordinate. The default behavior is to center the tick marks. For `datadensity.data.frame`, `bottom.align` defaults to `TRUE` if `nint>1`. In other words, if you are only labeling the first and last axis tick mark, the `scat1d` tick marks are centered on the variable's axis.
`preserve`	set to `TRUE` to invoke `jitter2`
`fill`	maximum fraction of the axis filled by jittered values. If `d` are duplicated values between a lower value `l` and upper value `u`, then `d` will be spread within `+/- fill*min(u-d,d-l)/2`.
`limit`	specifies a limit for maximum shift in jittered values. Duplicate values will be spread within `+/- fill*min(limit,min(u-d,d-l)/2)`. The default `TRUE` restricts jittering to the smallest min(u-d,d-l)/2 observed and results in equal amount of jittering for all d. Setting to `FALSE` allows for locally different amount of jittering, using maximum space available.

`nhistSpike`	If the number of observations exceeds or equals `nhistSpike`, `scat1d` will automatically call `histSpike` to draw the data density, to prevent the graphics file from being too large.
`type`	used by or passed to `histSpike`. Set to `"count"` to display frequency counts rather than relative frequencies, or `"density"` to display a kernel density estimate computed using the `density` function.
`grid`	set to `TRUE` if the R `grid` package is in effect for the current plot
`nint`	number of intervals to divide each continuous variable's axis for `datadensity`. For `histSpike`, is the number of equal-width intervals for which to bin `x`, and if instead `nint` is a character string (e.g., `nint="all"`), the frequency tabulation is done with no binning. In other words, frequencies for all unique values of `x` are derived and plotted.
`...`	optional arguments passed to `scat1d` from `datadensity` or to `histSpike` from `scat1d`
`presorted`	set to `TRUE` to prevent from sorting for determining the order l<d<u. This is usefull if an existing meaningfull local order would be destroyed by sorting, as in sin(pi*sort(round(runif(1000,0,10),1))).
`group`	an optional stratification variable, which is converted to a `factor` vector if it is not one already
`which`	set `which="continuous"` to only plot continuous variables, or `which="categorical"` to only plot categorical, character, or discrete numeric ones. By default, all types of variables are depicted.
`method.cat`	set `method.cat="freq"` to depict frequencies of categorical variables with digits representing the cell frequencies, with size proportional to the square root of the frequency. By default, vertical bars are used.
`col.group`	colors representing the `group` strata. The vector of colors is recycled to be the same length as the levels of `group`.
`n.unique`	number of unique values a numeric variable must have before it is considered to be a continuous variable
`show.na`	set to `FALSE` to suppress drawing the number of `NA`s to the right of each axis
`naxes`	number of axes to draw on each page before starting a new plot. You can set `naxes` larger than the number of variables in the data frame if you want to compress the plot vertically.
`q`	a vector of quantiles to display. By default, quantiles are not shown.
`cex.axis`	character size for draw labels for axis tick marks
`cex.var`	character size for variable names and frequence of `NA`s
`lmgp`	spacing between numeric axis labels and axis (see `par` for `mgp`)
`tck`	see `tck` under `par`
`ranges`	a list containing ranges for some or all of the numeric variables. If `ranges` is not given or if a certain variable is not found in the list, the empirical range, modified by `pretty`, is used. Example: `ranges=list(age=c(10,100), pressure=c(50,150))`.

`labels`	a vector of labels to use in labeling the axes for `datadensity.data.frame`. Default is to use the names of the variables in the input data frame. Note: margin widths computed for setting aside names of variables use the names, and not these labels.
`minf`	For `histSpike`, if `minf` is specified low bin frequencies are set to a minimum value of `minf` times the maximum bin frequency, so that rare data points will remain visible. A good choice of `minf` is 0.075. `datadensity.data.frame` passes `minf=0.075` to `scat1d` to pass to `histSpike`. Note that specifying `minf` will cause the shape of the histogram to be distorted somewhat.
`mult.width`	multiplier for the smoothing window width computed by `histSpike` when `type="density"`
`xlim`	a 2-vector specifying the outer limits of `x` for binning (and plotting, if `add=FALSE` and `nint` is a number)
`ylim`	y-axis range for plotting (if `add=FALSE`)
`xlab`	x-axis label (`add=FALSE`); default is name of input argument `x`
`ylab`	y-axis label (`add=FALSE`)
`add`	set to `TRUE` to add the spike-histogram to an existing plot, to show marginal data densities

Details

For `scat1d` the length of line segments used is `frac*min(par()$pin) / par()$uin[opp]` data units, where `opp` is the index of the opposite axis and `frac` defaults to .02. Assumes that `plot` has already been called. Current `par("usr")` is used to determine the range of data for the axis of the current plot. This range is used in jittering and in constructing line segments.

Value

`histSpike` returns the actual range of `x` used in its binning

Side Effects

`scat1d` adds line segments to plot. `datadensity.data.frame` draws a complete plot. `histSpike` draws a complete plot or adds to an existing plot.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
Charlottesville VA, USA
f.harrell@vanderbilt.edu

Martin Maechler (improved `scat1d`)
Seminar fuer Statistik
ETH Zurich SWITZERLAND
maechler@stat.math.ethz.ch

Jens Oehlschlaegel-Akiyoshi (wrote `jitter2`)
Center for Psychotherapy Research
Christian-Belser-Strasse 79a
D-70597 Stuttgart Germany
oehl@psyres-stuttgart.de

See Also

`segments`, `jitter`, `rug`, `plsmo`, `stripplot`, `hist.data.frame`,Ecdf, `hist`, `histogram`, `table`, `density`

Examples

```
plot(x <- rnorm(50), y <- 3*x + rnorm(50)/2 )
scat1d(x)                 # density bars on top of graph
scat1d(y, 4)              # density bars at right
histSpike(x, add=TRUE)       # histogram instead, 100 bins
histSpike(y, 4, add=TRUE)
histSpike(x, type='density', add=TRUE)  # smooth density at bottom
histSpike(y, 4, type='density', add=TRUE)

smooth <- lowess(x, y)    # add nonparametric regression curve
lines(smooth)             # Note: plsmo() does this
scat1d(x, y=approx(smooth, xout=x)$y) # data density on curve
scat1d(x, curve=smooth)   # same effect as previous command
histSpike(x, curve=smooth, add=TRUE) # same as previous but with histogram
histSpike(x, curve=smooth, type='density', add=TRUE)
# same but smooth density over curve

plot(x <- rnorm(250), y <- 3*x + rnorm(250)/2)
scat1d(x, tfrac=0)        # dots randomly spaced from axis
scat1d(y, 4, frac=-.03)   # bars outside axis
scat1d(y, 2, tfrac=.2)    # same bars with smaller random fraction

x <- c(0:3,rep(4,3),5,rep(7,10),9)
plot(x, jitter2(x))       # original versus jittered values
abline(0,1)               # unique values unjittered on abline
points(x+0.1, jitter2(x, limit=FALSE), col=2)
                          # allow locally maximum jittering
points(x+0.2, jitter2(x, fill=1), col=3); abline(h=seq(0.5,9,1), lty=2)
                          # fill 3/3 instead of 1/3
x <- rnorm(200,0,2)+1; y <- x^2
x2 <- round((x+rnorm(200))/2)*2
x3 <- round((x+rnorm(200))/4)*4
dfram <- data.frame(y,x,x2,x3)
plot(dfram$x2, dfram$y)   # jitter2 via scat1d
scat1d(dfram$x2, y=dfram$y, preserve=TRUE, col=2)
scat1d(dfram$x2, preserve=TRUE, frac=-0.02, col=2)
scat1d(dfram$y, 4, preserve=TRUE, frac=-0.02, col=2)

pairs(jitter2(dfram))     # pairs for jittered data.frame
# This gets reasonable pairwise scatter plots for all combinations of
# variables where
#
# - continuous variables (with unique values) are not jittered at all, thus
#   all relations between continuous variables are shown as they are,
#   extreme values have exact positions.
#
# - discrete variables get a reasonable amount of jittering, whether they
#   have 2, 3, 5, 10, 20 ... levels
#
# - different from adding noise, jitter2() will use the available space
#   optimally and no value will randomly mask another
#
# If you want a scatterplot with lowess smooths on the *exact* values and
# the point clouds shown jittered, you just need
#
pairs( dfram ,panel=function(x,y) { points(jitter2(x),jitter2(y))
                                    lines(lowess(x,y)) } )



datadensity(dfram)     # graphical snapshot of entire data frame
datadensity(dfram, group=cut2(dfram$x2,g=3))
                          # stratify points and frequencies by
                          # x2 tertiles and use 3 colors

# datadensity.data.frame(split(x, grouping.variable))
# need to explicitly invoke datadensity.data.frame when the
# first argument is a list
```

`score.binary` *Score a Series of Binary Variables*

Description

Creates a new variable from a series of logical conditions. The new variable can be a hierarchical category or score derived from considering the rightmost `TRUE` value among the input variables, an additive point score, a union, or any of several others by specifying a function using the `fun` argument.

Usage

```
score.binary(..., fun=max, points=1:p,
              na.rm=funtext == "max", retfactor=TRUE)
```

Arguments

`...`	a list of variables or expressions which are considered to be binary or logical
`fun`	a function to compute on each row of the matrix represented by a specific observation of all the variables in `...`

`points`	points to assign to successive elements of `...` . The default is `1, 2, ..., p`, where `p` is the number of elements. If you specify one number for `points`, that number will be duplicated (i.e., equal weights are assumed).
`na.rm`	set to `TRUE` to remove `NA`s from consideration when processing each row of the matrix of variables in `...` . For `fun=max`, `na.rm=TRUE` is the default since `score.binary` assumes that a hierarchical scale is based on available information. Otherwise, `na.rm=FALSE` is assumed. For `fun=mean` you may want to specify `na.rm=TRUE`.
`retfactor`	applies if `fun=max`, in which case `retfactor=TRUE` makes `score.binary` return a `factor` object since a hierarchical scale implies a unique choice.

Value

a `factor` object if `retfactor=TRUE` and `fun=max` or a numeric vector otherwise. Will not contain NAs if `na.rm=TRUE` unless every variable in a row is `NA`. If a `factor` object is returned, it has levels `"none"` followed by character string versions of the arguments given in `...` .

See Also

`any`, `sum`, `max`, `factor`

Examples

```
set.seed(1)
age <- rnorm(25, 70, 15)
previous.disease <- sample(0:1, 25, TRUE)
#Hierarchical scale, highest of 1:age>70  2:previous.disease
score.binary(age>70, previous.disease, retfactor=FALSE)
#Same as above but return factor variable with levels "none" "age>70"
# "previous.disease"
score.binary(age>70, previous.disease)

#Additive scale with weights 1:age>70  2:previous.disease
score.binary(age>70, previous.disease, fun=sum)
#Additive scale, equal weights
score.binary(age>70, previous.disease, fun=sum, points=c(1,1))
#Same as saying points=1

#Union of variables, to create a new binary variable
score.binary(age>70, previous.disease, fun=any)
```

sedit *Character String Editing and Miscellaneous Character Handling Functions*

Description

This suite of functions was written to implement many of the features of the UNIX `sed` program entirely within S-PLUS (function `sedit`). The `substring.location` function returns the first and last position numbers that a sub-string occupies in a larger string. The `substring2<-` function does the opposite of the builtin function `substring`. It is named `substring2` because for S-Plus 5.x there is a built-in function `substring`, but it does not handle multiple replacements in a single string. `replace.substring.wild` edits character strings in the fashion of "change xxxxANYTHINGyyyy to aaaaANYTHINGbbbb", if the "ANYTHING" passes an optional user-specified `test` function. Here, the "yyyy" string is searched for from right to left to handle balancing parentheses, etc. `numeric.string` and `all.digits` are two examples of `test` functions, to check, respectively if each of a vector of strings is a legal numeric or if it contains only the digits 0-9. For the case where `old="*$" or "^*"`, or for `replace.substring.wild` with the same values of `old` or with `front=TRUE` or `back=TRUE`, `sedit` (if `wild.literal=FALSE`) and `replace.substring.wild` will edit the largest substring satisfying `test`.

`substring2` is just a copy of `substring` so that `substring2<-` will work.

Usage

```
sedit(text, from, to, test, wild.literal=FALSE)
substring.location(text, string, restrict)
# substring(text, first, last) <- setto   # S-Plus only
replace.substring.wild(text, old, new, test, front=FALSE, back=FALSE)
numeric.string(string)
all.digits(string)
substring2(text, first, last=1e6)
substring2(text, first, last) <- value
```

Arguments

`text`	a vector of character strings for `sedit, substring2, substring2<-` or a single character string for `substring.location, replace.substring.wild`.
`from`	a vector of character strings to translate from, for `sedit`. A single asterisk wild card, meaning allow any sequence of characters (subject to the `test` function, if any) in place of the `"*"`. An element of `from` may begin with `"^"` to force the match to begin at the beginning of `text`, and an element of `from` can end with `"$"` to force the match to end at the end of `text`.
`to`	a vector of character strings to translate to, for `sedit`. If a corresponding element in `from` had an `"*"`, the element in `to` may also have an `"*"`. Only single asterisks are allowed. If `to` is not the same length as `from`, the `rep` function is used to make it the same length.
`string`	a single character string, for `substring.location`, `numeric.string`, `all.digits`
`first`	a vector of integers specifying the first position to replace for `substring2<-`. `first` may also be a vector of character strings that are passed to `sedit` to use as patterns for replacing substrings with `setto`. See one of the last examples below.

`last`	a vector of integers specifying the ending positions of the character substrings to be replaced. The default is to go to the end of the string. When `first` is character, `last` must be omitted.
`setto`	a character string or vector of character strings used as replacements, in `substring2<-`
`old`	a character string to translate from for `replace.substring.wild`. May be `"*$"` or `"^*"` or any string containing a single `"*"` but not beginning with `"^"` or ending with `"$"`.
`new`	a character string to translate to for `replace.substring.wild`
`test`	a function of a vector of character strings returning a logical vector whose elements are `TRUE` or `FALSE` according to whether that string element qualifies as the wild card string for `sedit, replace.substring.wild`
`wild.literal`	set to `TRUE` to not treat asterisks as wild cards and to not look for `"^"` or `"$"` in `old`
`restrict`	a vector of two integers for `substring.location` which specifies a range to which the search for matches should be restricted
`front`	specifying `front=TRUE` and `old="*"` is the same as specifying `old="^*"`
`back`	specifying `back=TRUE` and `old="*"` is the same as specifying `old="*$"`
`value`	a character vector

Value

`sedit` returns a vector of character strings the same length as `text`. `substring.location` returns a list with components named `first` and `last`, each specifying a vector of character positions corresponding to matches. `replace.substring.wild` returns a single character string. `numeric.string` and `all.digits` return a single logical value.

Side Effects

`substring2<-` modifies its first argument

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

See Also

`grep`, `substring`

Examples

```
x <- 'this string'
substring2(x, 3, 4) <- 'IS'
x
substring2(x, 7) <- ''
x

substring.location('abcdefgabc', 'ab')
substring.location('abcdefgabc', 'ab', restrict=c(3,999))

replace.substring.wild('this is a cat','this*cat','that*dog')
replace.substring.wild('there is a cat','is a*', 'is not a*')
replace.substring.wild('this is a cat','is a*', 'Z')

qualify <- function(x) x==' 1.5 ' | x==' 2.5 '
replace.substring.wild('He won 1.5 million $','won*million',
                       'lost*million', test=qualify)
replace.substring.wild('He won 1 million $','won*million',
                       'lost*million', test=qualify)
replace.substring.wild('He won 1.2 million $','won*million',
                       'lost*million', test=numeric.string)

x <- c('a = b','c < d','hello')
sedit(x, c('=','he*o'),c('==','he*'))

sedit('x23', '*$', '[*]', test=numeric.string)
sedit('23xx', '^*', 'Y_{*} ', test=all.digits)

replace.substring.wild("abcdefabcdef", "d*f", "xy")

x <- "abcd"
substring2(x, "bc") <- "BCX"
x
substring2(x, "B*d") <- "B*D"
x
```

`show.pch` *Display Colors, Plotting Symbols, and Symbol Numeric Equivalents*

Description

`show.pch` plots the definitions of the `pch` parameters. `show.col` plots definitions of integer-valued colors. `character.table` draws numeric equivalents of all latin characters; the character on line `xy` and column `z` of the table has numeric code `"xyz"`, which you would surround in quotes and preceed by a backslash.

Usage

```
show.pch(object = par("font"))
show.col(object=NULL)
character.table(font=1)
```

Arguments

`object`	font for `show.pch`, ignored for `show.col`.
`font`	font

Author(s)

Pierre Joyet ⟨pierre.joyet@bluewin.ch⟩, Frank Harrell

See Also

`points`, `text`

Examples

```
## Not run:
show.pch()
show.col()
character.table()
## End(Not run)
```

`simplifyDims` *List Simplification*

Description

Takes a list where each element is a group of rows that have been spanned by a multirow row and combines it into one large matrix.

Usage

```
simplifyDims(x)
```

Arguments

`x`	list of spanned rows

Details

All rows must have the same number of columns. This is used to format the list for printing.

Value

a matrix that contains all of the spanned rows.

Author(s)

Charles Dupont

See Also

`rbind`

Examples

```
a <- list(a = matrix(1:25, ncol=5), b = matrix(1:10, ncol=5), c = 1:5)

simplifyDims(a)
```

`smean.sd` *Compute Summary Statistics on a Vector*

Description

A number of statistical summary functions is provided for use with `summary.formula` and `summarize` (as well as `tapply` and by themselves). `smean.cl.normal` computes 3 summary variables: the sample mean and lower and upper Gaussian confidence limits based on the t-distribution. `smean.sd` computes the mean and standard deviation. `smean.sdl` computes the mean plus or minus a constant times the standard deviation. `smean.cl.boot` is a very fast implementation of the basic nonparametric bootstrap for obtaining confidence limits for the population mean without assuming normality. These functions all delete NAs automatically. `smedian.hilow` computes the sample median and a selected pair of outer quantiles having equal tail areas.

Usage

```
smean.cl.normal(x, mult=qt((1+conf.int)/2,n-1), conf.int=.95, na.rm=TRUE)

smean.sd(x, na.rm=TRUE)

smean.sdl(x, mult=2, na.rm=TRUE)

smean.cl.boot(x, conf.int=.95, B=1000, na.rm=TRUE, reps=FALSE)

smedian.hilow(x, conf.int=.95, na.rm=TRUE)
```

Arguments

`x`	for summary functions `smean.*`, `smedian.hilow`, a numeric vector from which NAs will be removed automatically
`na.rm`	defaults to `TRUE` unlike built-in S-Plus functions, so that by default `NA`s are automatically removed

`mult`	for `smean.cl.normal` is the multiplier of the standard error of the mean to use in obtaining confidence limits of the population mean (default is appropriate quantile of the t distribution). For `smean.sdl`, `mult` is the multiplier of the standard deviation used in obtaining a coverage interval about the sample mean. The default is `mult=2` to use plus or minus 2 standard deviations.
`conf.int`	for `smean.cl.normal` and `smean.cl.boot` specifies the confidence level (0-1) for interval estimation of the population mean. For `smedian.hilow`, `conf.int` is the coverage probability the outer quantiles should target. When the default, 0.95, is used, the lower and upper quantiles computed are 0.025 and 0.975.
`B`	number of bootstrap resamples for `smean.cl.boot`
`reps`	set to `TRUE` to have `smean.cl.boot` return the vector of bootstrapped means as the `reps` attribute of the returned object

Value

a vector of summary statistics

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`summarize`, `summary.formula`

Examples

```
set.seed(1)
x <- rnorm(100)
smean.sd(x)
smean.sdl(x)
smean.cl.normal(x)
smean.cl.boot(x)
smedian.hilow(x, conf.int=.5)  # 25th and 75th percentiles

# Function to compute 0.95 confidence interval for the difference in two means
# g is grouping variable
bootdif <- function(y, g) {
 g <- as.factor(g)
 a <- attr(smean.cl.boot(y[g==levels(g)[1]], B=2000, reps=TRUE),'reps')
 b <- attr(smean.cl.boot(y[g==levels(g)[2]], B=2000, reps=TRUE),'reps')
 meandif <- diff(tapply(y, g, mean, na.rm=TRUE))
 a.b <- quantile(b-a, c(.025,.975))
 res <- c(meandif, a.b)
 names(res) <- c('Mean Difference','.025','.975')
 res
}
```

solvet *solve Function with tol argument*

Description

A slightly modified version of `solve` that allows a tolerance argument for singularity (`tol`) which is passed to `qr`.

Usage

```
solvet(a, b, tol=1e-09)
```

Arguments

`a`	a square numeric matrix
`b`	a numeric vector or matrix
`tol`	tolerance for detecting linear dependencies in columns of `a`

See Also

`solve`

somers2 *Somers' Dxy Rank Correlation*

Description

Computes Somers' Dxy rank correlation between a variable `x` and a binary (0-1) variable `y`, and the corresponding receiver operating characteristic curve area `c`. Note that `Dxy = 2(c-0.5)`. `somers` allows for a `weights` variable, which specifies frequencies to associate with each observation.

Usage

```
somers2(x, y, weights=NULL, normwt=FALSE, na.rm=TRUE)
```

Arguments

`x`	typically a predictor variable. `NA`s are allowed.
`y`	a numeric outcome variable coded `0-1`. `NA`s are allowed.
`weights`	a numeric vector of observation weights (usually frequencies). Omit or specify a zero-length vector to do an unweighted analysis.
`normwt`	set to `TRUE` to make `weights` sum to the actual number of non-missing observations.
`na.rm`	set to `FALSE` to suppress checking for NAs.

Details

The `rcorr.cens` function, which although slower than `somers2` for large sample sizes, can also be used to obtain Dxy for non-censored binary `y`, and it has the advantage of computing the standard deviation of the correlation index.

Value

a vector with the named elements `C`, `Dxy`, `n` (number of non-missing pairs), and `Missing`. Uses the formula `C = (mean(rank(x)[y == 1]) - (n1 + 1)/2)/(n - n1)`, where `n1` is the frequency of `y=1`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

See Also

`rcorr.cens`, `rank`, `wtd.rank`,

Examples

```
set.seed(1)
predicted <- runif(200)
dead      <- sample(0:1, 200, TRUE)
roc.area <- somers2(predicted, dead)["C"]
```

`spower` *Simulate Power of 2-Sample Test for Survival under Complex Conditions*

Description

Given functions to generate random variables for survival times and censoring times, `spower` simulates the power of a user-given 2-sample test for censored data. By default, the logrank (Cox 2-sample) test is used, and a `logrank` function for comparing 2 groups is provided. For composing S-Plus functions to generate random survival times under complex conditions, the `Quantile2` function allows the user to specify the intervention:control hazard ratio as a function of time, the probability of a control subject actually receiving the intervention (dropin) as a function of time, and the probability that an intervention subject receives only the control agent as a function of time (non-compliance, dropout). `Quantile2` returns a function that generates either control or intervention uncensored survival times subject to non-constant treatment effect, dropin, and dropout. There is a `plot` method for plotting the results of `Quantile2`, which will aid in understanding the effects of the two types of non-compliance and non-constant treatment effects. `Quantile2` assumes that the hazard function for either treatment group is a mixture of the control and intervention hazard functions, with mixing proportions defined by the dropin and dropout probabilities. It computes hazards and survival distributions by numerical differentiation and integration using a grid of (by default) 7500 equally-spaced time points.

The `logrank` function is intended to be used with `spower` but it can be used by itself as long as the `group` variable has only the values `1` and `2` and there are no missing data. It returns the 1 degree of freedom chi-square statistic.

The `Weibull2` function accepts as input two vectors, one containing two times and one containing two survival probabilities, and it solves for the scale and shape parameters of the Weibull distribution (`S(t)=exp(-alpha*t^ gamma)`) which will yield those estimates. It creates an S-Plus function to evaluate survival probabilities from this Weibull distribution. `Weibull2` is useful in creating functions to pass as the first argument to `Quantile2`.

The `Lognorm2` and `Gompertz2` functions are similar to `Weibull2` except that they produce survival functions for the log-normal and Gompertz distributions.

Usage

```
spower(rcontrol, rinterv, rcens, nc, ni,
       test=logrank, nsim=500, alpha=0.05, pr=TRUE)

Quantile2(scontrol, hratio,
          dropin=function(times)0, dropout=function(times)0,
          m=7500, tmax, qtmax=.001, mplot=200, pr=TRUE, ...)

## S3 method for class 'Quantile2':
print(x, ...)

## S3 method for class 'Quantile2':
plot(x,
     what=c('survival','hazard','both','drop','hratio','all'),
     dropsep=FALSE, lty=1:4, col=1, xlim, ylim=NULL,
     label.curves=NULL, ...)

logrank(S, group)

Gompertz2(times, surv)
Lognorm2(times, surv)
Weibull2(times, surv)
```

Arguments

`rcontrol`	a function of `n` which returns `n` random uncensored failure times for the control group. `spower` assumes that non-compliance (dropin) has been taken into account by this function.
`rinterv`	similar to `rcontrol` but for the intervention group
`rcens`	a function of `n` which returns `n` random censoring times. It is assumed that both treatment groups have the same censoring distribution.

`nc`	number of subjects in the control group
`ni`	number in the intervention group
`scontrol`	a function of a time vector which returns the survival probabilities for the control group at those times assuming that all patients are compliant
`hratio`	a function of time which specifies the intervention:control hazard ratio (treatment effect)
`x`	an object of class `"Quantile2"` created by `Quantile2`
`S`	a `Surv` object or other two-column matrix for right-censored survival times
`group`	group indicators have length equal to the number of rows in `S`. Only values allowed are 1 and 2.
`times`	a vector of two times
`surv`	a vector of two survival probabilities
`test`	any function of a `Surv` object and a grouping variable which computes a chi-square for a two-sample censored data test. The default is `logrank`.
`nsim`	number of simulations to perform (default=500)
`alpha`	type I error (default=.05)
`pr`	set to `FALSE` to cause `spower` to suppress progress notes for simulations. Set to `FALSE` to prevent `Quantile2` from printing `tmax` when it calculates `tmax`.
`dropin`	a function of time specifying the probability that a control subject actually becomes an intervention subject at the corresponding time
`dropout`	a function of time specifying the probability of an intervention subject dropping out to control conditions
`m`	number of time points used for approximating functions (default is 7500)
`tmax`	maximum time point to use in the grid of `m` times. Default is the time such that `scontrol(time)` is `qtmax`.
`qtmax`	survival probability corresponding to the last time point used for approximating survival and hazard functions. Default is `.001`. For `qtmax` of the time for which a simulated time is needed which corresponds to a survival probability of less than `qtmax`, the simulated value will be `tmax`.
`mplot`	number of points used for approximating functions for use in plotting (default is 200 equally spaced points)
`...`	optional arguments passed to the `scontrol` function when it's evaluated by `Quantile2`
`what`	a single character constant (may be abbreviated) specifying which functions to plot. The default is `"both"` meaning both survival and hazard functions. Specify `what="drop"` to just plot the dropin and dropout functions, `what="hratio"` to plot the hazard ratio functions, or `"all"` to make 4 separate plots showing all functions (6 plots if `dropsep=TRUE`).
`dropsep`	set `dropsep=TRUE` to make `plot.Quantile2` separate pure and contaminated functions onto separate plots
`lty`	vector of line types

`col`	vector of colors
`xlim`	optional x-axis limits
`ylim`	optional y-axis limits
`label.curves`	optional list which is passed as the `opts` argument to `labcurve`.

Value

`spower` returns the power estimate (fraction of simulated chi-squares greater than the alpha-critical value). `Quantile2` returns an S-Plus function of class `"Quantile2"` with attributes that drive the `plot` method. The major attribute is a list containing several lists. Each of these sub-lists contains a `Time` vector along with one of the following: survival probabilities for either treatment group and with or without contamination caused by non-compliance, hazard rates in a similar way, intervention:control hazard ratio function with and without contamination, and dropin and dropout functions. `logrank` returns a single chi-square statistic, and `Weibull2`, `Lognorm2` and `Gompertz2` return an S function with three arguments, only the first of which (the vector of `times`) is intended to be specified by the user.

Side Effects

`spower` prints the interation number every 10 iterations if `pr=TRUE`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Lakatos E (1988): Sample sizes based on the log-rank statistic in complex clinical trials. Biometrics 44:229–241 (Correction 44:923).

Cuzick J, Edwards R, Segnan N (1997): Adjusting for non-compliance and contamination in randomized clinical trials. Stat in Med 16:1017–1029.

Cook, T (2003): Methods for mid-course corrections in clinical trials with survival outcomes. Stat in Med 22:3431–3447.

Barthel FMS, Babiker A et al (2006): Evaluation of sample size and power for multi-arm survival trials allowing for non-uniform accrual, non-proportional hazards, loss to follow-up and cross-over. Stat in Med 25:2521–2542.

See Also

`cpower`, `ciapower`, `bpower`, `cph`, `coxph`, `labcurve`

Examples

```
# Simulate a simple 2-arm clinical trial with exponential survival so
# we can compare power simulations of logrank-Cox test with cpower()
# Hazard ratio is constant and patients enter the study uniformly
# with follow-up ranging from 1 to 3 years
# Drop-in probability is constant at .1 and drop-out probability is
# constant at .175.  Two-year survival of control patients in absence
# of drop-in is .8 (mortality=.2).  Note that hazard rate is -log(.8)/2
# Total sample size (both groups combined) is 1000
# % mortality reduction by intervention (if no dropin or dropout) is 25
# This corresponds to a hazard ratio of 0.7283 (computed by cpower)

cpower(2, 1000, .2, 25, accrual=2, tmin=1,
       noncomp.c=10, noncomp.i=17.5)

ranfun <- Quantile2(function(x)exp(log(.8)/2*x),
                    hratio=function(x)0.7283156,
                    dropin=function(x).1,
                    dropout=function(x).175)

rcontrol <- function(n) ranfun(n, what='control')
rinterv  <- function(n) ranfun(n, what='int')
rcens    <- function(n) runif(n, 1, 3)

set.seed(11)   # So can reproduce results
spower(rcontrol, rinterv, rcens, nc=500, ni=500,
       test=logrank, nsim=50)  # normally use nsim=500 or 1000

# Simulate a 2-arm 5-year follow-up study for which the control group's
# survival distribution is Weibull with 1-year survival of .95 and
# 3-year survival of .7.  All subjects are followed at least one year,
# and patients enter the study with linearly increasing probability  after that
# Assume there is no chance of dropin for the first 6 months, then the
# probability increases linearly up to .15 at 5 years
# Assume there is a linearly increasing chance of dropout up to .3 at 5 years
# Assume that the treatment has no effect for the first 9 months, then
# it has a constant effect (hazard ratio of .75)

# First find the right Weibull distribution for compliant control patients
sc <- Weibull2(c(1,3), c(.95,.7))
sc

# Inverse cumulative distribution for case where all subjects are followed
# at least a years and then between a and b years the density rises
# as (time - a) ^ d is a + (b-a) * u ^ (1/(d+1))

rcens <- function(n) 1 + (5-1) * (runif(n) ^ .5)
# To check this, type hist(rcens(10000), nclass=50)

# Put it all together

f <- Quantile2(sc,
       hratio=function(x)ifelse(x<=.75, 1, .75),
       dropin=function(x)ifelse(x<=.5, 0, .15*(x-.5)/(5-.5)),
       dropout=function(x).3*x/5)

par(mfrow=c(2,2))
# par(mfrow=c(1,1)) to make legends fit
plot(f, 'all', label.curves=list(keys='lines'))

rcontrol <- function(n) f(n, 'control')
rinterv  <- function(n) f(n, 'intervention')

set.seed(211)
spower(rcontrol, rinterv, rcens, nc=350, ni=350,
       test=logrank, nsim=50)  # normally nsim=500 or more
par(mfrow=c(1,1))
```

---

spss.get *Enhanced Importing of SPSS Files*

Description

spss.get invokes the read.spss function in the **foreign** package to read an SPSS file, with a default output format of "data.frame". The label function is used to attach labels to individual variables instead of to the data frame as done by read.spss. By default, integer-valued variables are converted to a storage mode of integer unless force.single=FALSE. Date variables are converted to R Date variables. By default, underscores in names are converted to periods.

Usage

```
spss.get(file, lowernames=FALSE, datevars = NULL,
         use.value.labels = TRUE, to.data.frame = TRUE,
         max.value.labels = Inf, force.single=TRUE,
         allow=NULL, charfactor=FALSE)
```

Arguments

file	input SPSS save file. May be a file on the WWW, indicated by file starting with 'http://'.
lowernames	set to TRUE to convert variable names to lower case
datevars	vector of variable names containing dates to be converted to R internal format
use.value.labels	see read.spss
to.data.frame	see read.spss; default is TRUE for spss.get
max.value.labels	see read.spss

`force.single`	set to `FALSE` to prevent integer-valued variables from being converted from storage mode `double` to `integer`
`allow`	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.
`charfactor`	set to `TRUE` to change character variables to factors if they have at least two characters in an observation but have fewer than n/2 unique values

Value

a data frame or list

Author(s)

Frank Harrell

See Also

`read.spss`,`cleanup.import`,`sas.get`

Examples

```
## Not run:
w <- spss.get('/tmp/my.sav', datevars=c('birthdate','deathdate'))

## End(Not run)
```

`src` *Source a File from the Current Working Directory*

Description

`src` concatenates `".s"` to its argument, quotes the result, and `source`s in the file. It sets `options(last.source)` to this file name so that `src()` can be issued to re-`source` the file when it is edited.

Usage

```
src(x)
```

Arguments

`x`	an unquoted file name aside from `".s"`. This base file name must be a legal S name.

Side Effects

Sets system option `last.source`

Author(s)

Frank Harrell

See Also

`source`

Examples

```
## Not run:
src(myfile)   # source("myfile.s")
src()         # re-source myfile.s
## End(Not run)
```

`stata.get` *Enhanced Importing of STATA Files*

Description

Reads a file in Stata version 5-8 or 7/SE binary format into a data frame.

Usage

```
stata.get(file, lowernames = FALSE, convert.dates = TRUE,
          convert.factors = TRUE, missing.type = FALSE,
          convert.underscore = TRUE, warn.missing.labels = TRUE,
          force.single = TRUE, allow=NULL, charfactor=FALSE, ...)
```

Arguments

`file`	input SPSS save file. May be a file on the WWW, indicated by `file` starting with `'http://'`.
`lowernames`	set to `TRUE` to convert variable names to lower case
`convert.dates`	see `read.dta`
`convert.factors`	see `read.dta`
`missing.type`	see `read.dta`
`convert.underscore`	see `read.dta`
`warn.missing.labels`	see `read.dta`
`force.single`	set to `FALSE` to prevent integer-valued variables from being converted from storage mode `double` to `integer`
`allow`	a vector of characters allowed by R that should not be converted to periods in variable names. By default, underscores in variable names are converted to periods as with R before version 1.9.

`charfactor`	set to `TRUE` to change character variables to factors if they have at least two characters in an observation but have fewer than n/2 unique values
`...`	arguments passed to `read.dta`.

Details

`stata.get` invokes the `read.dta` function in the **foreign** package to read an STATA file, with a default output format of `data.frame`. The `label` function is used to attach labels to individual variables instead of to the data frame as done by `read.dta`. By default, integer-valued variables are converted to a storage mode of integer unless `force.single=FALSE`. Date variables are converted to R `Date` variables. By default, underscores in names are converted to periods.

Value

A data frame

Author(s)

Charles Dupont

See Also

`read.dta`,`cleanup.import`,`label`,`\label{data.frame}`,`Date`

Examples

```
## Not run:
w <- stata.get(\sQuote{/tmp/my.dta})
## End(Not run)
```

`store` *Store an Object Permanently*

Description

By default, `store` will copy the object to `.Data` under the same name. This function is most useful when you have attached a data frame or a temporary directory in position 1. `store` is also useful for setting up to store later objects in a temporary work area (`.Data.tempnnnn`, where `nnnn` is a number computed by the system) so that they are not stored on disk. For this usage, just invoke `store` with no arguments, i.e., `store()`. After that, you can still invoke `store` with arguments so that the object is copied to permanent storage. Another function, `stores` is useful for storing a series of temporary objects in `.Data` with one call. `store` and `stores` are not available For R. See Details below for a method of approximating the use of `store` in R.

`storeTemp` stores an object in frame 0 for S-Plus or in a temporary environment `.GlobalTemp` in R, attaching that environment if it is not already attached, so that the objects are easily available.

Usage

```
store(object, name=as.character(substitute(object)),
      where=if (under.unix || .SV4.) ".Data" else "_Data")
stores(...)
storeTemp(object, name=deparse(substitute(object)))
```

Arguments

`object`	object to store (omit to set search list position one to a temporary directory created by `store`)
`name`	name under which to store the object. Default is name of object in call to `store()`.
`where`	directory in which to store object. Default is `.Data` underneath current directory (for UNIX) or position 2 in the search list (for Windows). For R the default is `.GlobalEnv`.
`...`	a list of objects to store in `.Data` or `.GlobalEnv` permanently, using names which are the same as the argument names

Details

To almost mimic the functionality of `store` or `stores` in R, you can do the following. Use `save(x,y,z,file="Permdata")` to save permanent objects in `"permdata"`. When you exit R, do not save the workspace. Then all temporary objects will disappear. In your `.Rprofile` put the command `load("Permdata")` so that the next time you invoke R the permanent objects will be available.

Side Effects

uses `assign` and `attach` functions. `store` with no arguments also stores a function `.Last` in `.Data.tempnnnn`, which will cause `.Data.tempnnnn` to be removed when the S session ends. For S-Plus, `store()` causes creation of a system option named `.store.temp` which contains the name of the temporary directory created.

See Also

`assign`, `.Last`, `attach`, `search`

Examples

```
## Not run:
attach(database, 1)     #this database takes precedence
store()                 #new objects to go under database in memory
                        #this doesn't work in R
f <- lm(y ~ x)
store(f)                #store f under name "f" in .Data or .GlobalEnv
                        #uses assign() with immediate=T
store(f, "final.model") #immediately store f under "final.model" in .Data
store()                 #store future objects in .Data.tempnnnn
x <- runif(1000)        #x will disappear at session end unless
```

```
store(x)                #this statement appears -> store in .Data
stores(x, y, z)         #store x,y,z in .Data under names x,y,z
storeTemp(x)            #put x as name 'x' in frame 0
                        #for R, attach .GlobalTemp and store it there
storeTemp(x,'X')        #same as previous but under the name X
## End(Not run)
```

`string.bounding.box`
Determine Diamentions of Strings

Description

This determins the number of rows and maximum number of columns of each string in a vector.

Usage

```
string.bounding.box(string, type = c("chars", "width"))
```

Arguments

`string`	character vector, or a vector to be coerced to a character vector
`type`	character string: How the length of the string should be measured. See Details.

Details

The length of a string can be measured in one of two ways. See `nchar` Details section for more info.

'chars' The number of human readable characters.

'width' The number of columns used to print the string.

Value

`rows`	vector containing the number of character rows in each string.
`columns`	vector containing the maximum number of characters or character columns in each string.

Note

compatable with Splus `string.bounding.box`

Author(s)

Charles Dupont

See Also

`nchar`, `stringDims`

Examples

```
a <- c("this is a single line string", "This is a\nmulty line string")
stringDims(a)
```

`string.break.line` *Break a string into many lines at carage retruns.*

Description

Takes a string and breaks it into seperate substrings where there are carrage returns.

Usage

```
string.break.line(string)
```

Arguments

`string`	character vector to be separated into many lines.

Value

Returns a list that is the same length of as the `string` argument.

Each list element is a character vector.

Each character vectors elements are the split lines of the corasponding element in the `string` argument vector.

Author(s)

Charles Dupont

See Also

`strsplit`

Examples

```
a <- c('', 'this is a single line string',
       'This is a\nmulti-line string.')

b <- string.break.line(a)
```

`stringDims` *String Dimentions*

Description

Finds the height and width of all the string in a character vector.

Usage

```
stringDims(string)
```

Arguments

`string`	vector of strings

Details

`stringDims` finds the number of characters in width and number of lines in height for each string in the `string` argument.

Value

`height`	a vector of the number of lines in each string.
`width`	a vector with the number of character columns in the longest line.

Author(s)

Charles Dupont

See Also

`string.bounding.box`, `nchar`

Examples

```
a <- c("this is a single line string", "This is a\nmulty line string")
stringDims(a)
```

summarize — *Summarize Scalars or Matrices by Cross-Classification*

Description

summarize is a fast version of summary(formula, method="cross",overall=FALSE) for producing stratified summary statistics and storing them in a data frame for plotting (especially with trellis xyplot and dotplot and Hmisc xYplot). Unlike aggregate, summarize accepts a matrix as its first argument and a multi-valued FUN argument and summarize also labels the variables in the new data frame using their original names. Unlike methods based on tapply, summarize stores the values of the stratification variables using their original types, e.g., a numeric by variable will remain a numeric variable in the collapsed data frame. summarize also retains "label" attributes for variables. summarize works especially well with the Hmisc xYplot function for displaying multiple summaries of a single variable on each panel, such as means and upper and lower confidence limits.

asNumericMatrix converts a data frame into a numeric matrix, saving attributes to reverse the process by matrix2dataframe. It saves attributes that are commonly preserved across row subsetting (i.e., it does not save dim, dimnames, or names attributes).

matrix2dataFrame converts a numeric matrix back into a data frame if it was created by asNumericMatrix.

Usage

```
summarize(X, by, FUN, ...,
          stat.name=deparse(substitute(X)),
          type=c('variables','matrix'), subset=TRUE)

asNumericMatrix(x)

matrix2dataFrame(x, at=origAttributes, restoreAll=TRUE)
```

Arguments

X	a vector or matrix capable of being operated on by the function specified as the FUN argument
by	one or more stratification variables. If a single variable, by may be a vector, otherwise it should be a list. Using the Hmisc llist function instead of list will result in individual variable names being accessible to summarize. For example, you can specify llist(age.group,sex) or llist(Age=age.group,sex). The latter gives age.group a new temporary name, Age.
FUN	a function of a single vector argument, used to create the statistical summaries for summarize. FUN may compute any number of statistics.
...	extra arguments are passed to FUN
stat.name	the name to use when creating the main summary variable. By default, the name of the X argument is used. Set stat.name to NULL to suppress this name replacement.

`type`	Specify `type="matrix"` to store the summary variables (if there are more than one) in a matrix.
`subset`	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
`x`	a data frame (for `asNumericMatrix`) or a numeric matrix (for `matrix2dataFrame`).
`at`	List containing attributes of original data frame that survive subsetting. Defaults to object `"origAttributes"` in the global environment, created by the last call to `asNumericMatrix`
`restoreAll`	set to `FALSE` to only restore attributes `label`, `units`, and `levels` instead of all attributes

Value

For `summarize`, a data frame containing the `by` variables and the statistical summaries (the first of which is named the same as the `X` variable unless `stat.name` is given). If `type="matrix"`, the summaries are stored in a single variable in the data frame, and this variable is a matrix.

`asNumericMatrix` returns a numeric matrix and stores an object `origAttributes` in the global environment, with original attributes of component variables, plus an indicator for whether a variable was converted from character to factor to allow it to be made numeric.

`matrix2dataFrame` returns a data frame.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

`label`, `cut2`, `llist`, `by`

Examples

```
## Not run:
s <- summarize(ap>1, llist(size=cut2(sz, g=4), bone), mean,
               stat.name='Proportion')
dotplot(Proportion ~ size | bone, data=s7)
## End(Not run)

set.seed(1)
temperature <- rnorm(300, 70, 10)
month <- sample(1:12, 300, TRUE)
year  <- sample(2000:2001, 300, TRUE)
g <- function(x)c(Mean=mean(x,na.rm=TRUE),Median=median(x,na.rm=TRUE))
summarize(temperature, month, g)
mApply(temperature, month, g)

mApply(temperature, month, mean, na.rm=TRUE)
```

```
w <- summarize(temperature, month, mean, na.rm=TRUE)
if(.R.) library(lattice)
xyplot(temperature ~ month, data=w) # plot mean temperature by month

w <- summarize(temperature, llist(year,month),
               quantile, probs=c(.5,.25,.75), na.rm=TRUE, type='matrix')
xYplot(Cbind(temperature[,1],temperature[,-1]) ~ month | year, data=w)
mApply(temperature, llist(year,month),
       quantile, probs=c(.5,.25,.75), na.rm=TRUE)

# Compute the median and outer quartiles.  The outer quartiles are
# displayed using "error bars"
set.seed(111)
dfr <- expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
attach(dfr)
y <- abs(month-6.5) + 2*runif(length(month)) + year-1997
s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)
s
mApply(y, llist(month,year), smedian.hilow, conf.int=.5)

xYplot(Cbind(y,Lower,Upper) ~ month, groups=year, data=s,
       keys='lines', method='alt')
# Can also do:
s <- summarize(y, llist(month,year), quantile, probs=c(.5,.25,.75),
               stat.name=c('y','Q1','Q3'))
xYplot(Cbind(y, Q1, Q3) ~ month, groups=year, data=s, keys='lines')
# To display means and bootstrapped nonparametric confidence intervals
# use for example:
s <- summarize(y, llist(month,year), smean.cl.boot)
xYplot(Cbind(y, Lower, Upper) ~ month | year, data=s)

# For each subject use the trapezoidal rule to compute the area under
# the (time,response) curve using the Hmisc trap.rule function
x <- cbind(time=c(1,2,4,7, 1,3,5,10),response=c(1,3,2,4, 1,3,2,4))
subject <- c(rep(1,4),rep(2,4))
trap.rule(x[1:4,1],x[1:4,2])
summarize(x, subject, function(y) trap.rule(y[,1],y[,2]))

## Not run:
# Another approach would be to properly re-shape the mm array below
# This assumes no missing cells.  There are many other approaches.
# mApply will do this well while allowing for missing cells.
m <- tapply(y, list(year,month), quantile, probs=c(.25,.5,.75))
mm <- array(unlist(m), dim=c(3,2,12),
            dimnames=list(c('lower','median','upper'),c('1997','1998'),
                          as.character(1:12)))
# aggregate will help but it only allows you to compute one quantile
# at a time; see also the Hmisc mApply function
dframe <- aggregate(y, list(Year=year,Month=month), quantile, probs=.5)

# Compute expected life length by race assuming an exponential
# distribution - can also use summarize
g <- function(y) { # computations for one race group
```

```
  futime <- y[,1]; event <- y[,2]
  sum(futime)/sum(event)  # assume event=1 for death, 0=alive
}
mApply(cbind(followup.time, death), race, g)

# To run mApply on a data frame:
xn <- asNumericMatrix(x)
m <- mApply(xn, race, h)
# Here assume h is a function that returns a matrix similar to x
matrix2dataFrame(m)

# Get stratified weighted means
g <- function(y) wtd.mean(y[,1],y[,2])
summarize(cbind(y, wts), llist(sex,race), g, stat.name='y')
mApply(cbind(y,wts), llist(sex,race), g)

# Compare speed of mApply vs. by for computing
d <- data.frame(sex=sample(c('female','male'),100000,TRUE),
                country=sample(letters,100000,TRUE),
                y1=runif(100000), y2=runif(100000))
g <- function(x) {
  y <- c(median(x[,'y1']-x[,'y2']),
         med.sum =median(x[,'y1']+x[,'y2']))
  names(y) <- c('med.diff','med.sum')
  y
}

system.time(by(d, llist(sex=d$sex,country=d$country), g))
system.time({
             x <- asNumericMatrix(d)
             a <- subsAttr(d)
             m <- mApply(x, llist(sex=d$sex,country=d$country), g)
            })
system.time({
             x <- asNumericMatrix(d)
             summarize(x, llist(sex=d$sex, country=d$country), g)
            })

# An example where each subject has one record per diagnosis but sex of
# subject is duplicated for all the rows a subject has.  Get the cross-
# classified frequencies of diagnosis (dx) by sex and plot the results
# with a dot plot

count <- rep(1,length(dx))
d <- summarize(count, llist(dx,sex), sum)
Dotplot(dx ~ count | sex, data=d)
## End(Not run)
d <- list(x=1:10, a=factor(rep(c('a','b'),5)),
          b=structure(letters[1:10], label='label for a'))
x <- asNumericMatrix(d)
origAttributes
matrix2dataFrame(x)
```

```
detach('dfr')

# Run summarize on a matrix to get column means
x <- c(1:19,NA)
y <- 101:120
z <- cbind(x, y)
g <- c(rep(1, 10), rep(2, 10))
summarize(z, g, colMeans, na.rm=TRUE, stat.name='x')
# Also works on an all numeric data frame
summarize(as.data.frame(z), g, colMeans, na.rm=TRUE, stat.name='x')
```

`summary.formula` *Summarize Data for Making Tables and Plots*

Description

`summary.formula` summarizes the variables listed in an S-Plus formula, computing descriptive statistics (including ones in a user-specified function). The summary statistics may be passed to `print` methods, `plot` methods for making annotated dot charts, and `latex` methods for typesetting tables using LaTeX. `summary.formula` has three methods for computing descriptive statistics on univariate or multivariate responses, subsetted by categories of other variables. The method of summarization is specified in the parameter `method` (see details below). For the `response` and `cross` methods, the statistics used to summarize the data may be specified in a very flexible way (e.g., the geometric mean, 33rd percentile, Kaplan-Meier 2-year survival estimate, mixtures of several statistics). The default summary statistic for these methods is the mean (the proportion of positive responses for a binary response variable). The `cross` method is useful for creating data frames which contain summary statistics that are passed to `trellis` as raw data (to make multi-panel dot charts, for example). The `print` methods use the `print.char.matrix` function to print boxed tables, if it is available (it is included in S-Plus versions 3.2 and later).

The right hand side of `formula` may contain `mChoice` ("multiple choice") variables. When `test=TRUE` each choice is tested separately as a binary categorical response.

The `plot` method for `method="reverse"` creates a temporary function `Key` in frame 0 as is done by the `xYplot` and `Ecdf.formula` functions. After `plot` runs, you can type `Key()` to put a legend in a default location, or e.g. `Key(locator(1))` to draw a legend where you click the left mouse button. This key is for categorical variables, so to have the opportunity to put the key on the graph you will probably want to use the command `plot(object, which="categorical")` [Note however that in Windows S-Plus you can switch back and forth between multiple pages on a graph sheet, and issue a `Key()` or `Key2()` command according to which graph sheet page is active.]. A second function `Key2` is created if continuous variables are being plotted. It is used the same as `Key`. If the `which` argument is not specified to `plot`, two pages of plots will be produced. If you don't define `par(mfrow=)` yourself, `plot.summary.formula.reverse` will try to lay out a multi-panel graph to best fit all the individual dot charts for continuous variables.

There is a subscripting method for objects created with `method="response"`. This can be used to print or plot selected variables or summary statistics where there would otherwise be too many on one page.

`cumcategory` is a utility function useful when summarizing an ordinal response variable. It converts such a variable having `k` levels to a matrix with `k-1` columns, where column `i` is a vector of zeros and ones indicating that the categorical response is in level `i+1` or greater. When the left hand side of `formula` is `cumcategory(y)`, the default `fun` will summarize it by computing all of the relevant cumulative proportions.

Usage

```
## S3 method for class 'formula':
summary(formula, data, subset, na.action, fun = NULL,
        method = c("response", "reverse", "cross"),
        overall = method == "response" | method == "cross",
        continuous = 10, na.rm = TRUE, na.include = method != "reverse",
        g = 4, quant  = c(0.025, 0.05, 0.125, 0.25, 0.375, 0.5, 0.625,
                          0.75, 0.875, 0.95, 0.975),
        nmin = if (method == "reverse") 100
               else 0,
         test = FALSE,
        conTest = function(group, x) {
          st <- spearman2(group, x)
          list(P = st["P"], stat = st["F"], df = st[c("df1", "df2")],
               testname = if (st["df1"] == 1) "Wilcoxon"
                          else "Kruskal-Wallis",
               statname = "F", latexstat = "F_{df}",
               plotmathstat = "F[df]")
        },
        catTest = function(tab) {
          st <- if (!is.matrix(tab) || nrow(tab) < 2 | ncol(tab) < 2)
                  list(p.value = NA, statistic = NA, parameter = NA)
                else chisq.test(tab, correct = FALSE)
          list(P = st$p.value, stat = st$statistic, df = st$parameter,
               testname = "Pearson", statname = "Chi-square",
               latexstat = "\chi^{2}_{df}", plotmathstat = "chi[df]^2")
        },
        ordTest = function(group, x) {
          f <- lrm(x ~ group)$stats
          list(P = stats["P"], stat = stats["Model L.R."],
               df = stats["d.f."],
               testname = "Proportional odds likelihood ratio",
               statname = "Chi-square", latexstat = "\chi^{2}_{df}",
               plotmathstat = "chi[df]^2")
        }, ...)

## S3 method for class 'summary.formula.response':
print(x, vnames=c('labels','names'), prUnits=TRUE,
      abbreviate.dimnames=FALSE,
      prefix.width, min.colwidth, formatArgs, ...)

## S3 method for class 'summary.formula.response':
plot(x, which = 1, vnames = c('labels','names'), xlim, xlab,
     pch = c(16, 1, 2, 17, 15, 3, 4, 5, 0), superposeStrata = TRUE,
     dotfont = 1, add = FALSE, reset.par = TRUE, main, subtitles = TRUE,
     ...)

## S3 method for class 'summary.formula.response':
latex(object, title = first.word(deparse(substitute(object))), caption,
      trios, vnames = c('labels', 'names'), prn = TRUE, prUnits = TRUE,
      rowlabel = '', cdec = 2, ncaption = TRUE, ...)

## S3 method for class 'summary.formula.response':
x[i, j, drop=FALSE, ...]


## S3 method for class 'summary.formula.reverse':
print(x, digits, prn = any(n != N), pctdig = 0,
      npct = c('numerator', 'both', 'denominator', 'none'),
      exclude1 = TRUE, vnames = c('labels', 'names'), prUnits = TRUE,
      sep = '/', abbreviate.dimnames = FALSE,
      prefix.width = max(nchar(lab)), min.colwidth, formatArgs,
      prtest = c('P','stat','df','name'), prmsd = FALSE, long = FALSE,
      pdig = 3, eps = 0.001, ...)

## S3 method for class 'summary.formula.reverse':
plot(x, vnames = c('labels', 'names'), what = c('proportion', '%'),
     which = c('both', 'categorical', 'continuous'),
     xlim = if(what == 'proportion') c(0,1)
            else c(0,100),
     xlab = if(what=='proportion') 'Proportion'
            else 'Percentage',
     pch = c(16, 1, 2, 17, 15, 3, 4, 5, 0), exclude1 = TRUE,
     dotfont = 1, main, subtitles = TRUE,
     prtest = c('P', 'stat', 'df', 'name'), pdig = 3, eps = 0.001,
     conType = c('dot', 'bp', 'raw'), cex.means = 0.5, ...)

## S3 method for class 'summary.formula.reverse':
latex(object, title = first.word(deparse(substitute(object))), digits,
      prn = any(n != N), pctdig = 0,
      npct = c('numerator', 'both', 'denominator', 'none'),
      npct.size = 'scriptsize', Nsize = 'scriptsize', exclude1 = TRUE,
      vnames=c("labels", "names"), prUnits = TRUE, middle.bold = FALSE,
      outer.size = "scriptsize", caption, rowlabel = "",
      insert.bottom = TRUE, dcolumn = FALSE,
      prtest = c('P', 'stat', 'df', 'name'), prmsd = FALSE,
      msdsize = NULL, long = dotchart, pdig = 3, eps = 0.001,
      auxCol = NULL, dotchart=FALSE, ...)

## S3 method for class 'summary.formula.cross':
print(x, twoway = nvar == 2, prnmiss = any(stats$Missing > 0), prn = TRUE,
      abbreviate.dimnames = FALSE, prefix.width = max(nchar(v)),
      min.colwidth, formatArgs = NULL, ...)

## S3 method for class 'summary.formula.cross':
latex(object, title = first.word(deparse(substitute(object))),
      twoway = nvar == 2, prnmiss = TRUE, prn = TRUE,
      caption=attr(object,"heading"), vnames=c('labels','names'),
      rowlabel="", ...)

stratify(..., na.group=FALSE, shortlabel=TRUE)

## S3 method for class 'summary.formula.cross':
formula(x, ...)

cumcategory(y)
```

Arguments

`formula`	An S formula with additive effects. For `method="response"` or `"cross"`, the dependent variable has the usual connotation. For `method="reverse"`, the dependent variable is what is usually thought of as an independent variable, and it is one that is used to stratify all of the right hand side variables. For `method="response"` (only), the `formula` may contain one or more invocations of the `stratify` function whose arguments are defined below. This causes the entire analysis to be stratified by cross-classifications of the combined list of stratification factors. This stratification will be reflected as major column groupings in the resulting table, or as more response columns for plotting. If `formula` has no dependent variable `method="reverse"` is the only legal value and so `method` defaults to `"reverse"` in this case.
`x`	an object created by `summary.formula`
`y`	a numeric, character, category, or factor vector for `cumcategory`. Is converted to a categorical variable is needed.
`data`	name or number of a data frame. Default is the current frame.
`subset`	a logical vector or integer vector of subscripts used to specify the subset of data to use in the analysis. The default is to use all observations in the data frame.
`na.action`	function for handling missing data in the input data. The default is a function defined here called `na.retain`, which keeps all observations for processing, with missing variables or not.
`fun`	function for summarizing data in each cell. Default is to take the mean of each column of the possibly multivariate response variable. You can specify `fun="%"` to compute percentages (100 times the mean of a series of logical or binary variables). User–specified functions can also return a matrix. For example, you might compute quartiles on a bivariate response.
`method`	The default is `"response"`, in which case the response variable may be multivariate and any number of statistics may be used to summarize them. Here the responses are summarized separately for each of any number of independent variables. Continuous independent variables (see the `continuous` parameter below) are automatically stratified into `g` (see below) quantile groups (if you

want to control the discretization for selected variables, use the `cut2` function on them). Otherwise, the data are subsetted by all levels of discrete right hand side variables. For multivariate responses, subjects are considered to be missing if any of the columns is missing.

The `method="reverse"` option is typically used to make baseline characteristic tables, for example. The single left hand side variable must be categorical (e.g., treatment), and the right hand side variables are broken down one at a time by the "dependent" variable. Continuous variables are described by three quantiles (quartiles by default) along with outer quantiles (used only for scaling x-axes when plotting quartiles; all are used when plotting box-percentile plots), and categorical ones are described by counts and percentages. If there is no left hand side variable, `summary` assumes that there is only one group in the data, so that only one column of summaries will appear. If there is no dependent variable in `formula`, `method` defaults to `"reverse"` automatically.

The `method="cross"` option allows for a multivariate dependent variable and for up to three independents. Continuous independent variables (those with at least `continuous` unique values) are automatically divided into `g` quantile groups. The independents are cross-classified, and marginal statistics may optionally be computed. The output of `summary.formula` in this case is a data frame containing the independent variable combinations (with levels of `"All"` corresponding to marginals) and the corresponding summary statistics in the matrix `S`. The output data frame is suitable for direct use in `trellis`. The `print` and `latex` typesetting methods for this method allows for a special two-way format if there are two right hand variables.

`overall` For `method="reverse"`, setting `overall=TRUE` makes a new column with overall statistics for the whole sample. For `method="cross"`, `overall=TRUE` (the default) results in all marginal statistics being computed. For `trellis` displays (usually multi-panel dot plots), these marginals just form other categories. For `"response"`, the default is `overall=TRUE`, causing a final row of global summary statistics to appear in tables and dot charts. If `test=TRUE` these marginal statistics are ignored in doing statistical tests.

`continuous` specifies the threshold for when a variable is considered to be continuous (when there are at least `continuous` unique values). `factor` variables are always considered to be categorical no matter how many levels they have.

`na.rm` `TRUE` (the default) to exclude `NA`s before passing data to `fun` to compute statistics, `FALSE` otherwise. `na.rm=FALSE` is useful if the response variable is a matrix and you do not wish to exclude a row of the matrix if any of the columns in that row are `NA`. `na.rm` also applies to summary statistic functions such as `smean.cl.normal`. For these `na.rm` defaults to `TRUE` unlike built-in functions.

`na.include` for `method="response"`, set `na.include=FALSE` to exclude missing values from being counted as their own category when subsetting the response(s) by levels of a categorical variable. For `method="reverse"` set `na.include=TRUE` to keep missing values of categorical variables from being excluded from the table.

`g` number of quantile groups to use when variables are automatically categorized with `method="response"` or `"cross"` using `cut2`

`nmin`	if fewer than `nmin` observations exist in a category for `"response"` (over all strata combined), that category will be ignored. For `"reverse"`, for categories of the response variable in which there are less than or equal to `nmin` non-missing observations, the raw data are retained for later plotting in place of box plots.
`test`	applies if `method="reverse"`. Set to `TRUE` to compute test statistics using tests specified in `conTest` and `catTest`.
`conTest`	a function of two arguments (grouping variable and a continuous variable) that returns a list with components `P` (the computed P-value), `stat` (the test statistic, either chi-square or F), `df` (degrees of freedom), `testname` (test name), `statname` (statistic name), an optional component `latexstat` (LaTeX representation of `statname`), an optional component `plotmathstat` (for R - the `plotmath` representation of `statname`, as a character string), and an optional component `note` that contains a character string note about the test (e.g., `"test not done because n < 5"`). `conTest` is applied to continuous variables on the right-hand-side of the formula when `method="reverse"`. The default uses the `spearman2` function to run the Wilcoxon or Kruskal-Wallis test using the F distribution.
`catTest`	a function of a frequency table (an integer matrix) that returns a list with the same components as created by `conTest`. By default, the Pearson chi-square test is done, without continuity correction (the continuity correction would make the test conservative like the Fisher exact test).
`ordTest`	a function of a frequency table (an integer matrix) that returns a list with the same components as created by `conTest`. By default, the Proportional odds likelihood ratio test is done.
`...`	for `summary.formula` these are optional arguments for `cut2` when variables are automatically categorized. For `plot` methods these arguments are passed to `dotchart2`. For `Key` and `Key2` these arguments are passed to `key`, `text`, or `mtitle`. For `print` methods these are optional arguments to `print.char.matrix`. For `latex` methods these are passed to `latex.default`. One of the most important of these is `file`. Specifying `file=""` will cause LaTeX code to just be printed to standard output rather than be stored in a permanent file.
`drop`	logical. If 'TRUE' the result is coerced to the lowest possible dimension. The default is to drop if only one column is left, but *not* to drop if only one row is left.
`object`	an object created by `summary.formula`
`quant`	vector of quantiles to use for summarizing data with `method="reverse"`. This must be numbers between 0 and 1 inclusive and must include the numbers 0.5, 0.25, and 0.75 which are used for printing and for plotting quantile intervals. The outer quantiles are used for scaling the x-axes for such plots. Specify outer quantiles as `0` and `1` to scale the x-axes using the whole observed data ranges instead of the default (a 0.95 quantile interval). Box-percentile plots are drawn using all but the outer quantiles.
`vnames`	By default, tables and plots are usually labeled with variable labels (see the `label` and `sas.get` functions). To use the shorter variable names, specify `vnames="name"`.

`pch`	vector of plotting characters to represent different groups, in order of group levels. For `method="response"` the characters correspond to levels of the `stratify` variable if `superposeStrata=TRUE`, and if no `strata` are used or if `superposeStrata=FALSE`, the `pch` vector corresponds to the `which` argument for `method="response"`.
`superposeStrata`	If `stratify` was used, set `superposeStrata=FALSE` to make separate dot charts for each level of the `stratification` variable, for `method='response'`. The default is to superposition all strata on one dot chart.
`dotfont`	font for plotting points
`reset.par`	set to `FALSE` to suppress the restoring of the old par values in `plot.summary.formula.response`
`abbreviate.dimnames`	see `print.char.matrix`
`prefix.width`	see `print.char.matrix`
`min.colwidth`	minimum column width to use for boxes printed with `print.char.matrix`. The default is the maximum of the minimum column label length and the minimum length of entries in the data cells.
`formatArgs`	a list containing other arguments to pass to `format.default` such as `scientific`, e.g., `formatArgs=list(scientific=c(-5,5))`. For `print.summary.formula.revers` `formatArgs` applies only to statistics computed on continuous variables, not to percents, numerators, and denominators.
`digits`	number of significant digits to print. Default is to use the current value of the `digits` system option.
`prn`	set to `TRUE` to print the number of non-missing observations on the current (row) variable. The default is to print these only if any of the counts of non-missing values differs from the total number of non-missing values of the left-hand-side variable. For `method="cross"` the default is to always print `N`.
`prnmiss`	set to `FALSE` to suppress printing counts of missing values for `"cross"`
`pctdig`	number of digits to the right of the decimal place for printing percentages. The default is zero, so percents will be rounded to the nearest percent.
`npct`	specifies which counts are to be printed to the right of percentages. The default is to print the frequency (numerator of the percent) in parentheses. You can specify `"both"` to print both numerator and denominator, `"denominator"`, or `"none"`.
`npct.size`	the size for typesetting `npct` information which appears after percents. The default is `"scriptsize"`.
`Nsize`	When a second row of column headings is added showing sample sizes, `Nsize` specifies the LaTeX size for these subheadings. Default is `"scriptsize"`.
`exclude1`	by default, `method="reverse"` objects will be printed, plotted, or typeset by removing redundant entries from percentage tables for categorical variables. For example, if you print the percent of females, you don't need to print the percent of males. To override this, set `exclude1=FALSE`.
`prUnits`	set to `FALSE` to suppress printing or latexing `units` attributes of variables, when `method='reverse'` or `'response'`

`sep`	character to use to separate quantiles when printing `method="reverse"` tables
`prtest`	a vector of test statistic components to print if `test=TRUE` was in effect when `summary.formula` was called. Defaults to printing all components. Specify `prtest=FALSE` or `prtest="none"` to not print any tests. This applies to `print`, `latex`, and `plot` methods for `method='reverse'`.
`prmsd`	set to `TRUE` to print mean and SD after the three quantiles, for continuous variables with `method="reverse"`
`msdsize`	defaults to `NULL` to use the current font size for the mean and standard deviation if `prmsd` is `TRUE`. Set to a character string to specify an alternate LaTeX font size.
`long`	set to `TRUE` to print the results for the first category on its own line, not on the same line with the variable label (for `method="reverse"` with `print` and `latex` methods)
`pdig`	number of digits to the right of the decimal place for printing P-values. Default is `3`. This is passed to `format.pval`.
`eps`	P-values less than `eps` will be printed as `< eps`. See `format.pval`.
`auxCol`	an optional auxiliary column of information, right justified, to add in front of statistics typeset by `latex.summary.formula.reverse`. This argument is a list with a single element that has a name specifying the column heading. If this name includes a newline character, the portions of the string before and after the newline form respectively the main heading and the subheading (typically set in smaller font), respectively. See the `extracolheads` argument to `latex.default`. `auxCol` is filled with blanks when a variable being summarized takes up more than one row in the output. This happens with categorical variables.
`what`	for `method="reverse"` specifies whether proportions or percentages are to be plotted
`twoway`	for `method="cross"` with two right hand side variables, `twoway` controls whether the resulting table will be printed in enumeration format or as a two-way table (the default)
`which`	For `method="response"` specifies the sequential number or a vector of subscripts of response variables to plot. If you had any `stratify` variables, these are counted as if multiple response variables were analyzed. For `method="reverse"` specifies whether to plot results for categorical variables, continuous variables, or both (the default).
`conType`	For plotting `method="reverse"` plots for continuous variables, dot plots showing quartiles are drawn by default. Specify `conType='bp'` to draw box-percentile plots using all the quantiles in `quant` except the outermost ones. Means are drawn with a solid dot and vertical reference lines are placed at the three quartiles. Specify `conType='raw'` to make a strip chart showing the raw data. This can only be used if the sample size for each left-hand-side group is less than or equal to `nmin`.
`cex.means`	character size for means in box-percentile plots; default is .5

`xlim`	vector of length two specifying x-axis limits. For `method="reverse"`, this is only used for plotting categorical variables. Limits for continuous variables are determined by the outer quantiles specified in `quant`.
`xlab`	x-axis label
`add`	set to `TRUE` to add to an existing plot
`main`	a main title. For `method="reverse"` this applies only to the plot for categorical variables.
`subtitles`	set to `FALSE` to suppress automatic subtitles
`caption`	character string containing LaTeX table captions.
`title`	name of resulting LaTeX file omitting the `.tex` suffix. Default is the name of the `summary` object. If `caption` is specied, `title` is also used for the table's symbolic reference label.
`trios`	If for `method="response"` you summarized the response(s) by using three quantiles, specify `trios=TRUE` or `trios=v` to group each set of three statistics into one column for `latex` output, using the format a B c, where the outer quantiles are in smaller font (`scriptsize`). For `trios=TRUE`, the overall column names are taken from the column names of the original data matrix. To give new column names, specify `trios=v`, where `v` is a vector of column names, of length `m/3`, where `m` is the original number of columns of summary statistics.
`rowlabel`	see `latex.default` (under the help file `latex`)
`cdec`	number of decimal places to the right of the decimal point for `latex`. This value should be a scalar (which will be properly replicated), or a vector with length equal to the number of columns in the table. For `"response"` tables, this length does not count the column for `N`.
`ncaption`	set to `FALSE` to not have `latex.summary.formula.response` put sample sizes in captions
`i`	a vector of integers, or character strings containing variable names to subset on. Note that each row subsetted on in an `summary.formula.reverse` object subsets on all the levels that make up the corresponding variable (automatically).
`j`	a vector of integers representing column numbers
`middle.bold`	set to `TRUE` to have LaTeX use bold face for the middle quantile for `method="reverse"`
`outer.size`	the font size for outer quantiles for `"reverse"` tables
`insert.bottom`	set to `FALSE` to suppress inclusion of definitions placed at the bottom of LaTeX tables for `method="reverse"`
`dcolumn`	see `latex`
`na.group`	set to `TRUE` to have missing stratification variables given their own category (`NA`)
`shortlabel`	set to `FALSE` to include stratification variable names and equal signs in labels for strata levels
`dotchart`	set to `TRUE` to output a dotchart in the latex table being generated.

Value

summary.formula returns a data frame or list depending on method. plot.summary.formula.reverse returns the number of pages of plots that were made.

Side Effects

plot.summary.formula.reverse creates a function Key and Key2 in frame 0 that will draw legends.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Harrell FE (2004): Statistical tables and plots using S and LaTeX. Document available from http://biostat.mc.vanderbilt.edu/twiki/pub/Main/StatReport/summary.pdf.

See Also

mChoice, smean.sd, summarize, label, strata, dotchart2, print.char.matrix, update, formula, cut2, llist, format.default, latex, latexTranslate bpplt

Examples

```
options(digits=3)
set.seed(173)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
treatment <- factor(sample(c("Drug","Placebo"), 500, rep=TRUE))

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
          'Muscle Ache','Depressed')
symptom1 <- sample(symp, 500,TRUE)
symptom2 <- sample(symp, 500,TRUE)
symptom3 <- sample(symp, 500,TRUE)
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
table(Symptoms)

# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
# mChoice will ignore these redundant selections

#Frequency table sex*treatment, sex*Symptoms
summary(sex ~ treatment + Symptoms, fun=table)
# could also do summary(sex ~ treatment +
#   mChoice(symptom1,symptom2,symptom3), fun=table)
```

```
#Compute mean age, separately by 3 variables
summary(age ~ sex + treatment + Symptoms)

f <- summary(treatment ~ age + sex + Symptoms, method="reverse", test=TRUE)
f
# trio of numbers represent 25th, 50th, 75th percentile
print(f, long=TRUE)
plot(f)
plot(f, conType='bp', prtest='P')
bpplt()    # annotated example showing layout of bp plot

#Compute predicted probability from a logistic regression model
#For different stratifications compute receiver operating
#characteristic curve areas (C-indexes)
predicted <- plogis(.4*(sex=="m")+.15*(age-50))
positive.diagnosis <- ifelse(runif(500)<=predicted, 1, 0)
roc <- function(z) {
   x <- z[,1];
   y <- z[,2];
   n <- length(x);
   if(n<2)return(c(ROC=NA));
   n1 <- sum(y==1);
   c(ROC= (mean(rank(x)[y==1])-(n1+1)/2)/(n-n1) );
 }
y <- cbind(predicted, positive.diagnosis)
options(digits=2)
summary(y ~ age + sex, fun=roc)

options(digits=3)
summary(y ~ age + sex, fun=roc, method="cross")

#Use stratify() to produce a table in which time intervals go down the
#page and going across 3 continuous variables are summarized using
#quartiles, and are stratified by two treatments

set.seed(1)
d <- expand.grid(visit=1:5, treat=c('A','B'), reps=1:100)
d$sysbp <- rnorm(100*5*2, 120, 10)
label(d$sysbp) <- 'Systolic BP'
d$diasbp <- rnorm(100*5*2, 80,  7)
d$diasbp[1] <- NA
d$age    <- rnorm(100*5*2, 50, 12)
g <- function(y) {
  N <- apply(y, 2, function(w) sum(!is.na(w)))
  h <- function(x) {
    qu <- quantile(x, c(.25,.5,.75), na.rm=TRUE)
    names(qu) <- c('Q1','Q2','Q3')
    c(N=sum(!is.na(x)), qu)
}
  w <- as.vector(apply(y, 2, h))
  names(w) <- as.vector( outer(c('N','Q1','Q2','Q3'), dimnames(y)[[2]],
                                function(x,y) paste(y,x)))
  w
}
#Use na.rm=FALSE to count NAs separately by column
s <- summary(cbind(age,sysbp,diasbp) ~ visit + stratify(treat),
             na.rm=FALSE, fun=g, data=d)
#The result is very wide.  Re-do, putting treatment vertically
x <- with(d, factor(paste('Visit', visit, treat)))
summary(cbind(age,sysbp,diasbp) ~ x, na.rm=FALSE, fun=g, data=d)

#Compose LaTeX code directly
g <- function(y) {
  h <- function(x) {
    qu <- format(round(quantile(x, c(.25,.5,.75), na.rm=TRUE),1),nsmall=1)
    paste('{\scriptsize(',sum(!is.na(x)),
          ')} \hfill{\scriptsize ', qu[1], '} \textbf{', qu[2],
          '} {\scriptsize ', qu[3],'}', sep='')
  }
  apply(y, 2, h)
}
s <- summary(cbind(age,sysbp,diasbp) ~ visit + stratify(treat),
             na.rm=FALSE, fun=g, data=d)
# latex(s, prn=FALSE)
## need option in latex to not print n
#Put treatment vertically
s <- summary(cbind(age,sysbp,diasbp) ~ x, fun=g, data=d, na.rm=FALSE)
# latex(s, prn=FALSE)

#Plot estimated mean life length (assuming an exponential distribution)
#separately by levels of 4 other variables.  Repeat the analysis
#by levels of a stratification variable, drug.  Automatically break
#continuous variables into tertiles.
#We are using the default, method='response'
## Not run:
life.expect <- function(y) c(Years=sum(y[,1])/sum(y[,2]))
attach(pbc)
S <- Surv(follow.up.time, death)
s2 <- summary(S ~ age + albumin + ascites + edema + stratify(drug),
                         fun=life.expect, g=3)

#Note: You can summarize other response variables using the same
#independent variables using e.g. update(s2, response~.), or you
#can change the list of independent variables using e.g.
#update(s2, response ~.- ascites) or update(s2, .~.-ascites)
#You can also print, typeset, or plot subsets of s2, e.g.
#plot(s2[c('age','albumin'),]) or plot(s2[1:2,])

s2    # invokes print.summary.formula.response

#Plot results as a separate dot chart for each of the 3 strata levels
par(mfrow=c(2,2))
plot(s2, cex.labels=.6, xlim=c(0,40), superposeStrata=FALSE)

#Typeset table, creating s2.tex
```

```
w <- latex(s2, cdec=1)
#Typeset table but just print LaTeX code
latex(s2, file="")    # useful for Sweave

#Take control of groups used for age.  Compute 3 quartiles for
#both cholesterol and bilirubin (excluding observations that are missing
#on EITHER ONE)

age.groups <- cut2(age, c(45,60))
g <- function(y) apply(y, 2, quantile, c(.25,.5,.75))
y <- cbind(Chol=chol,Bili=bili)
label(y) <- 'Cholesterol and Bilirubin'
#You can give new column names that are not legal S-Plus names
#by enclosing them in quotes, e.g. 'Chol (mg/dl)'=chol

s <- summary(y ~ age.groups + ascites, fun=g)

par(mfrow=c(1,2), oma=c(3,0,3,0))   # allow outer margins for overall
for(ivar in 1:2) {                  # title
  isub <- (1:3)+(ivar-1)*3          # *3=number of quantiles/var.
  plot(s3, which=isub, main='',
       xlab=c('Cholesterol','Bilirubin')[ivar],
       pch=c(91,16,93))            # [, closed circle, ]
  }
mtext(paste('Quartiles of', label(y)), adj=.5, outer=TRUE, cex=1.75)
#Overall (outer) title

prlatex(latex(s3, trios=TRUE))
# trios -> collapse 3 quartiles

#Summarize only bilirubin, but do it with two statistics:
#the mean and the median.  Make separate tables for the two randomized
#groups and make plots for the active arm.

g <- function(y) c(Mean=mean(y), Median=median(y))

for(sub in c("D-penicillamine", "placebo")) {
  ss <- summary(bili ~ age.groups + ascites + chol, fun=g,
                subset=drug==sub)
  cat('\n',sub,'\n\n')
  print(ss)

  if(sub=='D-penicillamine') {
    par(mfrow=c(1,1))
    plot(s4, which=1:2, dotfont=c(1,-1), subtitles=FALSE, main='')
    #1=mean, 2=median     -1 font = open circle
    title(sub='Closed circle: mean;  Open circle: median', adj=0)
    title(sub=sub, adj=1)
  }

  w <- latex(ss, append=TRUE, fi='my.tex',
             label=if(sub=='placebo') 's4b' else 's4a',
             caption=paste(label(bili),' {\\em (',sub,')}', sep=''))
  #Note symbolic labels for tables for two subsets: s4a, s4b
  prlatex(w)
}

#Now consider examples in 'reverse' format, where the lone dependent
#variable tells the summary function how to stratify all the
#'independent' variables.  This is typically used to make tables
#comparing baseline variables by treatment group, for example.

s5 <- summary(drug ~ bili + albumin + stage + protime + sex +
                     age + spiders,
              method='reverse')
#To summarize all variables, use summary(drug ~., data=pbc)
#To summarize all variables with no stratification, use
#summary(~a+b+c) or summary(~.,data=...)

options(digits=1)
print(s5, npct='both')
#npct='both' : print both numerators and denominators
plot(s5, which='categorical')
Key(locator(1))  # draw legend at mouse click
par(oma=c(3,0,0,0))  # leave outer margin at bottom
plot(s5, which='continuous')
Key2()           # draw legend at lower left corner of plot
                 # oma= above makes this default key fit the page better

options(digits=3)
w <- latex(s5, npct='both', here=TRUE)
# creates s5.tex

#Turn to a different dataset and do cross-classifications on possibly
#more than one independent variable.  The summary function with
#method='cross' produces a data frame containing the cross-
#classifications.  This data frame is suitable for multi-panel
#trellis displays, although `summarize' works better for that.

attach(prostate)
size.quartile <- cut2(sz, g=4)
bone <- factor(bm,labels=c("no mets","bone mets"))

s7 <- summary(ap>1 ~ size.quartile + bone, method='cross')
#In this case, quartiles are the default so could have said sz + bone

options(digits=3)
print(s7, twoway=FALSE)
s7   # same as print(s7)
w <- latex(s7, here=TRUE)   # Make s7.tex

library(trellis,TRUE)
invisible(ps.options(reset=TRUE))
trellis.device(postscript, file='demo2.ps')

dotplot(S ~ size.quartile|bone, data=s7, #s7 is name of summary stats
                 xlab="Fraction ap>1", ylab="Quartile of Tumor Size")
#Can do this more quickly with summarize:
# s7 <- summarize(ap>1, llist(size=cut2(sz, g=4), bone), mean,
#                 stat.name='Proportion')
# dotplot(Proportion ~ size | bone, data=s7)

summary(age ~ stage, method='cross')
summary(age ~ stage, fun=quantile, method='cross')
summary(age ~ stage, fun=smean.sd, method='cross')
summary(age ~ stage, fun=smedian.hilow, method='cross')
summary(age ~ stage, fun=function(x) c(Mean=mean(x), Median=median(x)),
        method='cross')
#The next statements print real two-way tables
summary(cbind(age,ap) ~ stage + bone,
        fun=function(y) apply(y, 2, quantile, c(.25,.75)),
        method='cross')
options(digits=2)
summary(log(ap) ~ sz + bone,
        fun=function(y) c(Mean=mean(y), quantile(y)),
        method='cross')

#Summarize an ordered categorical response by all of the needed
#cumulative proportions
summary(cumcategory(disease.severity) ~ age + sex)

## End(Not run)
```

`symbol.freq` *Graphic Representation of a Frequency Table*

Description

This function can be used to represent contingency tables graphically. Frequency counts are represented as the heights of "thermometers" by default; you can also specify `symbol='circle'` to the function. There is an option to include marginal frequencies, which are plotted on a halved scale so as to not overwhelm the plot. If you do not ask for marginal frequencies to be plotted using `marginals=T`, `symbol.freq` will ask you to click the mouse where a reference symbol is to be drawn to assist in reading the scale of the frequencies.

`label` attributes, if present, are used for x- and y-axis labels. Otherwise, names of calling arguments are used.

Usage

```
symbol.freq(x, y, symbol = c("thermometer", "circle"),
             marginals = FALSE, orig.scale = FALSE,
             inches = 0.25, width = 0.15, subset, srtx = 0, ...)
```

Arguments

`x`	first variable to cross-classify
`y`	second variable
`symbol`	specify `"thermometer"` (the default) or `"circle"`
`marginals`	set to `TRUE` to add marginal frequencies (scaled by half) to the plot
`orig.scale`	set to `TRUE` when the first two arguments are numeric variables; this uses their original values for x and y coordinates)
`inches`	see `symbols`
`width`	see `thermometers` option in `symbols`
`subset`	the usual subsetting vector
`srtx`	rotation angle for x-axis labels
`...`	other arguments to pass to `symbols`

Author(s)

Frank Harrell

See Also

`symbols`

Examples

```
## Not run:
getHdata(titanic)
attach(titanic)
age.tertile <- cut2(titanic$age, g=3)
symbol.freq(age.tertile, pclass, marginals=T, srtx=45)
detach(2)
## End(Not run)
```

`sys` *Run Unix or Dos Depending on System*

Description

Runs `unix` or `dos` depending on the current operating system. For R, just runs `system` with optional concatenation of first two arguments which are assumed named `command` and `text`.

Usage

```
sys(command, text=NULL, output=TRUE)
# S-Plus: sys(..., minimized=FALSE)
```

Arguments

`command`	system command to execute
`text`	text to concatenate to system command, if any (typically options or file names or both)
`output`	set to `FALSE` to not return output of command as a character vector

Value

see `unix` or `dos`

Side Effects

executes system commands

See Also

`unix`, `system`

`t.test.cluster` *t-test for Clustered Data*

Description

Does a 2-sample t-test for clustered data.

Usage

```
t.test.cluster(y, cluster, group, conf.int = 0.95, ...)
## S3 method for class 't.test.cluster':
print(x, digits, ...)
```

Arguments

`y`	normally distributed response variable to test
`cluster`	cluster identifiers, e.g. subject ID
`group`	grouping variable with two values
`conf.int`	confidence coefficient to use for confidence limits
`x`	an object created by `t.test.cluster`
`digits`	number of significant digits to print
`...`	unused

Value

a matrix of statistics of class `t.test.cluster`

Author(s)

Frank Harrell

References

Donner A, Birkett N, Buck C, Am J Epi 114:906-914, 1981.

Donner A, Klar N, J Clin Epi 49:435-439, 1996.

Hsieh FY, Stat in Med 8:1195-1201, 1988.

See Also

`t.test`

Examples

```
set.seed(1)
y <- rnorm(800)
group <- sample(1:2, 800, TRUE)
cluster <- sample(1:40, 800, TRUE)
table(cluster,group)
t.test(y ~ group)   # R only
t.test.cluster(y, cluster, group)
# Note: negate estimates of differences from t.test to
# compare with t.test.cluster
```

`transace` *Additive Regression and Transformations using ace or avas*

Description

`transace` is `ace` packaged for easily automatically transforming all variables in a matrix. `transace` is a fast one-iteration version of `transcan` without imputation of NAs.

`areg.boot` uses `areg` or `avas` to fit additive regression models allowing all variables in the model (including the left-hand-side) to be transformed, with transformations chosen so as to optimize certain criteria. The default method uses `areg` whose goal it is to maximize R^2. `method='avas'` explicity tries to transform the response variable so as to stabilize the variance of the residuals. All-variables-transformed models tend to inflate `R^2` and it can be difficult to get confidence limits for each transformation. `areg.boot` solves both of these problems using the bootstrap. As with the `validate` function in the Design library, the Efron bootstrap is used to estimate the optimism in the apparent R^2, and this optimism is subtracted from the apparent R^2 to optain a bias-corrected R^2. This is done however on the transformed response variable scale.

Tests with 3 predictors show that the `avas` and `ace` estimates are unstable unless the sample size exceeds 350. Apparent R^2 with low sample sizes can be very inflated, and bootstrap estimates of R^2 can be even more unstable in such cases, resulting in optimism-corrected R^2 that are much lower even than the actual R^2. The situation can be improved a little by restricting predictor transformations to be monotonic. On the other hand, the `areg` approach allows one to control overfitting by specifying the number of knots to use for each continuous variable in a restricted cubic spline function.

For `method="avas"` the response transformation is restricted to be monotonic. You can specify restrictions for transformations of predictors (and linearity for the response). When the first argument is a formula, the function automatically determines which variables are categorical (i.e., `factor`, `category`, or character vectors). Specify linear transformations by enclosing variables by the identify function (`I()`), and specify monotonicity by using `monotone(variable)`. Monotonicity restrictions are not allowed with `method='areg'`.

The `summary` method for `areg.boot` computes bootstrap estimates of standard errors of differences in predicted responses (usually on the original scale) for selected levels of each predictor against the lowest level of the predictor. The smearing estimator (see below) can be used here to estimate differences in predicted means, medians, or many other statistics. By default, quartiles are used for continuous predictors and all levels are used for categorical ones. See DETAILS below. There is also a `plot` method for plotting transformation estimates, transformations for individual bootstrap re–samples, and pointwise confidence limits for transformations. Unless you already have a `par(mfrow=)` in effect with more than one row or column, `plot` will try to fit the plots on one page. A `predict` method computes predicted values on the original or transformed response scale, or a matrix of transformed predictors. There is a `Function` method for producing a list of S functions that perform the final fitted transformations. There is also a `print` method for `areg.boot` objects.

When estimated means (or medians or other statistical parameters) are requested for models fitted with `areg.boot` (by `summary.areg.boot` or `predict.areg.boot`), the "smearing" estimator of Duan (1983) is used. Here we estimate the mean of the untransformed response by computing the arithmetic mean of ginverse(lp + residuals), where ginverse is the inverse of the nonparametric transformation of the response (obtained by reverse linear interpolation), `lp` is the linear predictor for an individual observation on the transformed scale, and `residuals` is the entire vector of residuals estimated from the fitted model, on the transformed scales (n residuals for n original observations). The `smearingEst` function computes the general smearing estimate. For efficiency `smearingEst` recognizes that quantiles are transformation-preserving, i.e., when one wishes to estimate a quantile of the untransformed distribution one just needs to compute the inverse transformation of the transformed estimate after the chosen quantile of the vector of residuals is added to it. When the median is desired, the estimate is ginverse(lp + median(residuals)). See the last example for how `smearingEst` can be used outside of `areg.boot`.

`Mean` is a generic function that returns an S function to compute the estimate of the mean of a variable. Its input is typically some kind of model fit object. Likewise, `Quantile` is a generic quantile function-producing function. `Mean.areg.boot` and `Quantile.areg.boot` create functions of a vector of linear predictors that transform them into the smearing estimates of the mean or quantile of the response variable, respectively. `Quantile.areg.boot` produces exactly the same value as `predict.areg.boot` or `smearingEst`. `Mean` approximates the mapping of linear predictors to means over an evenly spaced grid of by default 200 points. Linear interpolation is used between these points. This approximate method is much faster than the full smearing estimator once `Mean` creates the function. These functions are especially useful in `nomogram.Design` (see the example on hypothetical data).

Usage

```
transace(x, monotonic=NULL, categorical=NULL, binary=NULL, pl=TRUE)
```

```
areg.boot(x, data, weights, subset, na.action=na.delete,
          B=100, method=c("areg","avas"), nk=4, evaluation=100, valrsq=TRUE,
          probs=c(.25,.5,.75), tolerance=NULL)

## S3 method for class 'areg.boot':
print(x, ...)

## S3 method for class 'areg.boot':
plot(x, ylim, boot=TRUE, col.boot=2, lwd.boot=.15,
conf.int=.95, ...)

smearingEst(transEst, inverseTrans, res,
            statistic=c('median','quantile','mean','fitted','lp'),
            q)

## S3 method for class 'areg.boot':
summary(object, conf.int=.95, values, adj.to,
        statistic='median', q, ...)

## S3 method for class 'summary.areg.boot':
print(x, ...)

## S3 method for class 'areg.boot':
predict(object, newdata,
         statistic=c("lp", "median",
                     "quantile", "mean", "fitted", "terms"),
         q=NULL, ...)

## S3 method for class 'areg.boot':
Function(object, type=c('list','individual'),
         ytype=c('transformed','inverse'),
         prefix='.', suffix='', frame=0, where=1, ...)

Mean(object, ...)

Quantile(object, ...)

## S3 method for class 'areg.boot':
Mean(object, evaluation=200, ...)

## S3 method for class 'areg.boot':
Quantile(object, q=.5, ...)
```

Arguments

`x`	for `transace` a numeric matrix. For `areg.boot` `x` is a formula. For `print` or `plot`, an object created by `areg.boot`. For `print.summary.areg.boot`, and object created by `summary.areg.boot`.

`object`	an object created by `areg.boot`, or a model fit object suitable for `Mean` or `Quantile`.
`transEst`	a vector of transformed values. In log-normal regression these could be predicted log(Y) for example.
`inverseTrans`	a function specifying the inverse transformation needed to change `transEst` to the original untransformed scale. `inverseTrans` may also be a 2-element list defining a mapping from the transformed values to untransformed values. Linear interpolation is used in this case to obtain untransform values.
`monotonic`	
`categorical`	
`binary`	These are vectors of variable names specifying what to assume about each column of `x` for `transace`. Binary variables are not transformed, of course.
`pl`	set `pl=FALSE` to prevent `transace` from plotting each fitted transformation
`data`	data frame to use if `x` is a formula and variables are not already in the search list
`weights`	a numeric vector of observation weights. By default, all observations are weighted equally.
`subset`	an expression to subset data if `x` is a formula
`na.action`	a function specifying how to handle NAs. Default is `na.delete` (in Hmisc).
`B`	number of bootstrap samples (default=100)
`method`	`"areg"` (the default) or `avas`
`nk`	number of knots for continuous variables not restricted to be linear. Default is 4. One or two is not allowed. `nk=0` forces linearity for all continuous variables.
`evaluation`	number of equally-spaced points at which to evaluate (and save) the nonparametric transformations derived by `avas` or `ace`. Default is 100. For `Mean.areg.boot`, `evaluation` is the number of points at which to evaluate exact smearing estimates, to approximate them using linear interpolation (default is 200).
`valrsq`	set to `TRUE` to more quickly do bootstrapping without validating R^2
`probs`	vector probabilities denoting the quantiles of continuous predictors to use in estimating effects of those predictors
`tolerance`	singularity criterion; list source code for the `lm.fit.qr.bare` function.
`res`	a vector of residuals from the transformed model. Not required when `statistic="lp"` or `statistic="fitted"`.
`statistic`	statistic to estimate with the smearing estimator. For `smearingEst`, the default results in computation of the sample median of the model residuals, then `smearingEst` adds the median residual and back-transforms to get estimated median responses on the original scale. `statistic="lp"` causes predicted transformed responses to be computed. For `smearingEst`, the result (for `statistic="lp"`) is the input argument `transEst`. `statistic="fitted"` gives predicted untransformed responses, i.e., ginverse(lp), where ginverse is the inverse of the estimated response transformation, estimated by reverse linear interpolation on the tabulated nonparametric response transformation or by using an explicit analytic function. `statistic="quantile"` generalizes `"median"` to any single quantile `q` which must be specified. "mean"' causes

	the population mean response to be estimated. For `predict.areg.boot`, `statistic="terms"` returns a matrix of transformed predictors. `statistic` can also be any S function that computes a single value on a vector of values, such as `statistic=var`. Note that in this case the function name is not quoted.
`q`	a single quantile of the original response scale to estimate, when `statistic="quantile"`, or for `Quantile.areg.boot`.
`ylim`	2-vector of y-axis limits
`boot`	set to `FALSE` to not plot any bootstrapped transformations. Set it to an integer `k` to plot the first `k` bootstrap estimates.
`col.boot`	color for bootstrapped transformations
`lwd.boot`	line width for bootstrapped transformations
`conf.int`	confidence level (0-1) for pointwise bootstrap confidence limits and for estimated effects of predictors in `summary.areg.boot`. The latter assumes normality of the estimated effects.
`values`	a list of vectors of settings of the predictors, for predictors for which you want to overide settings determined from `probs`. The list must have named components, with names corresponding to the predictors. Example: `values=list(x1=c(2,4,6,8), x2=c(-1,0,1))` specifies that `summary` is to estimate the effect on `y` of changing `x1` from 2 to 4, 2 to 6, 2 to 8, and separately, of changing `x2` from -1 to 0 and -1 to 1.
`adj.to`	a named vector of adjustment constants, for setting all other predictors when examining the effect of a single predictor in `summary`. The more nonlinear is the transformation of `y` the more the adjustment settings will matter. Default values are the medians of the values defined by `values` or `probs`. You only need to name the predictors for which you are overriding the default settings. Example: `adj.to=c(x2=0,x5=10)` will set `x2` to 0 and `x5` to 10 when assessing the impact of variation in the other predictors.
`newdata`	a data frame or list containing the same number of values of all of the predictors used in the fit. For `factor` predictors the `levels` attribute do not need to be in the same order as those used in the original fit, and not all levels need to be represented. If `newdata` is omitted, you can still obtain linear predictors (on the transformed response scale) and fitted values (on the original response scale), but not `"terms"`.
`type`	specifies how `Function` is to return the series of functions that define the transformations of all variables. By default a list is created, with the names of the list elements being the names of the variables. Specify `type="individual"` to have separate functions created in the session frame (`frame=0`, the default) or in location defined by `where` if `where` is specified. For the latter method, the names of the objects created are the names of the corresponding variables, prefixed by `prefix` and with `suffix` appended to the end. If any of `frame`, `where`, `prefix`, or `suffix` is specified, `type` is automatically set to `"individual"`.
`ytype`	By default the first function created by `Function` is the y-transformation. Specify `ytype="inverse"` to instead create the inverse of the transformation, to be able to obtain originally scaled y-values.

`prefix`	character string defining the prefix for function names created when `type="individual"`. By default, the function specifying the transformation for variable `x` will be named `.x`.
`suffix`	character string defining the suffix for the function names
`frame`	frame number in which to store functions (see `assign`). The default is frame 0, the session database, which disappears at the end of the S-Plus session.
`where`	location in which to store functions (see `assign`). If `where` is specified (e.g., `where=1` to store functions in search position one), `frame` is ignored. For R, the value of `where` is passed to `assign` as the `pos` argument.
`...`	arguments passed to other functions

Details

As `transace` only does one iteration over the predictors, it may not find optimal transformations and it will be dependent on the order of the predictors in `x`.

`ace` and `avas` standardize transformed variables to have mean zero and variance one for each bootstrap sample, so if a predictor is not important it will still consistently have a positive regression coefficient. Therefore using the bootstrap to estimate standard errors of the additive least squares regression coefficients would not help in drawing inferences about the importance of the predictors. To do this, `summary.areg.boot` computes estimates of, e.g., the inter-quartile range effects of predictors in predicting the response variable (after untransforming it). As an example, at each bootstrap repetition the estimated transformed value of one of the predictors is computed at the lower quartile, median, and upper quartile of the raw value of the predictor. These transformed x values are then multipled by the least squares estimate of the partial regression coefficient for that transformed predictor in predicting transformed y. Then these weighted transformed x values have the weighted transformed x value corresponding to the lower quartile subtracted from them, to estimate an x effect accounting for nonlinearity. The last difference computed is then the standardized effect of raising x from its lowest to its highest quartile. Before computing differences, predicted values are back-transformed to be on the original y scale in a way depending on `statistic` and `q`. The sample standard deviation of these effects (differences) is taken over the bootstrap samples, and this is used to compute approximate confidence intervals for effects and approximate P-values, both assuming normality.

`predict` does not re-insert NAs corresponding to observations that were dropped before the fit, when `newdata` is omitted.

`statistic="fitted"` estimates the same quantity as `statistic="median"` if the residuals on the transformed response have a symmetric distribution. The two provide identical estimates when the sample median of the residuals is exactly zero. The sample mean of the residuals is constrained to be exactly zero although this does not simplify anything.

Value

`transace` returns a matrix like `x` but containing transformed values. This matrix has attributes `rsq` (vector of R^2 with which each variable can be predicted from the others) and `omitted` (row numbers of `x` that were deleted due to NAs).

`areg.boot` returns a list of class `"areg.boot"` containing many elements, including (if `valrsq` is `TRUE`) `rsquare.app` and `rsquare.val`. `summary.areg.boot` returns a list of class "summary.areg.boot" containing a matrix of results for each predictor and a vector of adjust-to settings. It also contains the call and a label for the statistic that was computed. A print method for these objects handles the printing. predict.areg.boot returns a vector unless statistic="terms", in which case it returns a matrix. Function.areg.boot returns by default a list of functions whose argument is one of the variables (on the original scale) and whose returned values are the corresponding transformed values. The names of the list of functions correspond to the names of the original variables. When type="individual", Function.areg.boot invisibly returns the vector of names of the created function objects. Mean.areg.boot and Quantile.areg.boot also return functions.

smearingEst returns a vector of estimates of distribution parameters of class "labelled" so that print.labelled wil print a label documenting the estimate that was used (see label). This label can be retrieved for other purposes by using e.g. label(obj), where obj was the vector returned by smearingEst.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Harrell FE, Lee KL, Mark DB (1996): Stat in Med 15:361–387.

Duan N (1983): Smearing estimate: A nonparametric retransformation method. JASA 78:605–610.

Wang N, Ruppert D (1995): Nonparametric estimation of the transformation in the transform-both-sides regression model. JASA 90:522–534.

See avas, ace for primary references.

See Also

avas, ace, ols, validate, predab.resample, label, nomogram

Examples

```
# xtrans <- transace(cbind(age,sex,blood.pressure,race.code),
#                    binary='sex', monotonic='age',
#                    categorical='race.code')

# Generate random data from the model y = exp(x1 + epsilon/3) where
# x1 and epsilon are Gaussian(0,1)
set.seed(171)  # to be able to reproduce example
x1 <- rnorm(200)
x2 <- runif(200)  # a variable that is really unrelated to y]
x3 <- factor(sample(c('cat','dog','cow'), 200,TRUE))  # also unrelated to y
y  <- exp(x1 + rnorm(200)/3)
f  <- areg.boot(y ~ x1 + x2 + x3, B=40)
f
plot(f)
```

```
# Note that the fitted transformation of y is very nearly log(y)
# (the appropriate one), the transformation of x1 is nearly linear,
# and the transformations of x2 and x3 are essentially flat
# (specifying monotone(x2) if method='avas' would have resulted
# in a smaller confidence band for x2)

summary(f)

# use summary(f, values=list(x2=c(.2,.5,.8))) for example if you
# want to use nice round values for judging effects

# Plot Y hat vs. Y (this doesn't work if there were NAs)
plot(fitted(f), y)  # or: plot(predict(f,statistic='fitted'), y)

# Show fit of model by varying x1 on the x-axis and creating separate
# panels for x2 and x3.  For x2 using only a few discrete values
newdat <- expand.grid(x1=seq(-2,2,length=100),x2=c(.25,.75),
                      x3=c('cat','dog','cow'))
yhat <- predict(f, newdat, statistic='fitted')
# statistic='mean' to get estimated mean rather than simple inverse trans.
xYplot(yhat ~ x1 | x2, groups=x3, type='l', data=newdat)

## Not run:
# Another example, on hypothetical data
f <- areg.boot(response ~ I(age) + monotone(blood.pressure) + race)
# use I(response) to not transform the response variable
plot(f, conf.int=.9)
# Check distribution of residuals
plot(fitted(f), resid(f))
qqnorm(resid(f))
# Refit this model using ols so that we can draw a nomogram of it.
# The nomogram will show the linear predictor, median, mean.
# The last two are smearing estimators.
Function(f, type='individual')  # create transformation functions
f.ols <- ols(.response(response) ~ age +
             .blood.pressure(blood.pressure) + .race(race))
# Note: This model is almost exactly the same as f but there
# will be very small differences due to interpolation of
# transformations
meanr <- Mean(f)      # create function of lp computing mean response
medr  <- Quantile(f)  # default quantile is .5
nomogram(f.ols, fun=list(Mean=meanr,Median=medr))

# Create S functions that will do the transformations
# This is a table look-up with linear interpolation
g <- Function(f)
plot(blood.pressure, g$blood.pressure(blood.pressure))
# produces the central curve in the last plot done by plot(f)
## End(Not run)

# Another simulated example, where y has a log-normal distribution
# with mean x and variance 1.  Untransformed y thus has median
# exp(x) and mean exp(x + .5sigma^2) = exp(x + .5)
# First generate data from the model y = exp(x + epsilon),
# epsilon ~ Gaussian(0, 1)

set.seed(139)
n <- 1000
x <- rnorm(n)
y <- exp(x + rnorm(n))
f <- areg.boot(y ~ x, B=20)
plot(f)       # note log shape for y, linear for x.  Good!
xs <- c(-2, 0, 2)
d <- data.frame(x=xs)
predict(f, d, 'fitted')
predict(f, d, 'median')   # almost same; median residual=-.001
exp(xs)                   # population medians
predict(f, d, 'mean')
exp(xs + .5)              # population means

# Show how smearingEst works
res <- c(-1,0,1)          # define residuals
y <- 1:5
ytrans <- log(y)
ys <- seq(.1,15,length=50)
trans.approx <- list(x=log(ys), y=ys)
options(digits=4)
smearingEst(ytrans, exp, res, 'fitted')          # ignores res
smearingEst(ytrans, trans.approx, res, 'fitted') # ignores res
smearingEst(ytrans, exp, res, 'median')          # median res=0
smearingEst(ytrans, exp, res+.1, 'median')       # median res=.1
smearingEst(ytrans, trans.approx, res, 'median')
smearingEst(ytrans, exp, res, 'mean')
mean(exp(ytrans[2] + res))                       # should equal 2nd # above
smearingEst(ytrans, trans.approx, res, 'mean')
smearingEst(ytrans, trans.approx, res, mean)
# Last argument can be any statistical function operating
# on a vector that returns a single value
```

`transcan` *Transformations/Imputations using Canonical Variates*

Description

`transcan` is a nonlinear additive transformation and imputation function, and there are several functions for using and operating on its results. `transcan` automatically transforms continuous and categorical variables to have maximum correlation with the best linear combination of the other variables. There is also an option to use a substitute criterion - maximum correlation with the first principal component of the other variables. Continuous variables are expanded as restricted cubic splines and categorical variables are expanded as contrasts (e.g., dummy variables). By default, the first canonical variate is used to find optimum linear combinations of component columns. This function is similar to `ace` except that transformations for continuous variables are fitted using restricted cubic splines, monotonicity restrictions are not allowed, and NAs are allowed. When a variable has any NAs, transformed scores for that variable are imputed using least squares multiple regression incorporating optimum transformations, or NAs are optionally set to constants. Shrinkage can be used to safeguard against overfitting when imputing. Optionally, imputed values on the original scale are also computed and returned. For this purpose, recursive partitioning or multinomial logistic models can optionally be used to impute categorical variables, using what is predicted to be the most probable category.

By default, `transcan` imputes NAs with "best guess" expected values of transformed variables, back transformed to the original scale. Values thus imputed are most like conditional medians assuming the transformations make variables' distributions symmetric (imputed values are similar to conditionl modes for categorical variables). By instead specifying `n.impute`, `transcan` does approximate multiple imputation from the distribution of each variable conditional on all other variables. This is done by sampling `n.impute` residuals from the transformed variable, with replacement (a la bootstrapping), or by default, using Rubin's approximate Bayesian bootstrap, where a sample of size n with replacement is selected from the residuals on n non-missing values of the target variable, and then a sample of size m with replacement is chosen from this sample, where m is the number of missing values needing imputation for the current multiple imputation repetition. Neither of these bootstrap procedures assume normality or even symmetry of residuals. For sometimes-missing categorical variables, optimal scores are computed by adding the "best guess" predicted mean score to random residuals off this score. Then categories having scores closest to these predicted scores are taken as the random multiple imputations (`impcat="tree"` or `"rpart"` are not currently allowed with `n.impute`). The literature recommends using `n.impute=5` or greater. `transcan` provides only an approximation to multiple imputation, especially since it "freezes" the imputation model before drawing the multiple imputations rather than using different estimates of regression coefficients for each imputation. For multiple imputation, the `aregImpute` function provides a much better approximation to the full Bayesian approach while still not requiring linearity assumptions.

When you specify `n.impute` to `transcan` you can use `fit.mult.impute` to re-fit any model `n.impute` times based on `n.impute` completed datasets (if there are any sometimes missing variables not specified to `transcan`, some observations will still be dropped from these fits). After fitting `n.impute` models, `fit.mult.impute` will return the fit object from the last imputation, with `coefficients` replaced by the average of the `n.impute` coefficient vectors and with a component `var` equal to the imputation-corrected variance-covariance matrix. `fit.mult.impute` can also use the object created by the `mice` function in the MICE library to draw the multiple imputations, as well as objects created by `aregImpute`.

The `summary` method for `transcan` prints the function call, R-squares achieved in transforming each variable, and for each variable the coefficients of all other transformed variables that are used to estimate the transformation of the initial variable. If `imputed=TRUE` was used in the call to transcan, also uses the `describe` function to print a summary of imputed values. If `long=TRUE`, also prints all imputed values with observation identifiers. There is also a simple function `print.transcan` which merely prints the transformation matrix and the function call. It has an optional argument `long`, which if set to `TRUE` causes detailed parameters to be printed. Instead of plotting while `transcan()` is running, you can plot the final transformations after the fact using `plot.transcan`, if the option `trantab=TRUE` was specified to `transcan`. If in addition the option `imputed=TRUE` was specified to `transcan`, `plot.transcan` will show the location of imputed values (including multiples) along the axes.

`impute` does imputations for a selected original data variable, on the original scale (if `imputed=TRUE` was given to `transcan`). If you do not specify a variable to `impute`, it will do imputations for all variables given to `transcan` which had at least one missing value. This assumes that the original variables are accessible (i.e., they have been `attach`ed) and that you want the imputed variables to have the same names are the original variables. If `n.impute` was specified to `transcan` you must tell `impute` which `imputation` to use.

`predict` computes predicted variables and imputed values from a matrix of new data. This matrix should have the same column variables as the original matrix used with `transcan`, and in the same order (unless a formula was used with `transcan`).

`Function` is a generic function generator. `Function.transcan` creates S functions to transform variables using transformations created by `transcan`. These functions are useful for getting predicted values with predictors set to values on the original scale.

`Varcov` methods are defined here so that imputation-corrected variance-covariance matrices are readily extracted from `fit.mult.impute` objects, and so that `fit.mult.impute` can easily compute traditional covariance matrices for individual completed datasets. Specific `Varcov` methods are defined for `lm`, `glm`, and `multinom` fits.

The subscript function preserves attributes.

The `invertTabulated` function does either inverse linear interpolation or uses sampling to sample qualifying x-values having y-values near the desired values. The latter is used to get inverse values having a reasonable distribution (e.g., no floor or ceiling effects) when the transformation has a flat or nearly flat segment, resulting in a many-to-one transformation in that region. Sampling weights are a combination of the frequency of occurrence of x-values that are within `tolInverse` times the range of `y` and the squared distance between the associated y-values and the target y-value (`aty`).

Usage

```
transcan(x, method=c("canonical","pc"),
         categorical=NULL, asis=NULL, nk, imputed=FALSE, n.impute,
         boot.method=c('approximate bayesian', 'simple'),
         trantab=FALSE, transformed=FALSE,
         impcat=c("score", "multinom", "rpart", "tree"),
         mincut=40,
         inverse=c('linearInterp','sample'), tolInverse=.05,
         pr=TRUE, pl=TRUE, allpl=FALSE, show.na=TRUE,
         imputed.actual=c('none','datadensity','hist','qq','ecdf'),
         iter.max=50, eps=.1, curtail=TRUE,
         imp.con=FALSE, shrink=FALSE, init.cat="mode",
         nres=if(boot.method=='simple')200 else 400,
         data, subset, na.action, treeinfo=FALSE,
         rhsImp=c('mean','random'), details.impcat='', ...)

## S3 method for class 'transcan':
summary(object, long=FALSE, ...)

## S3 method for class 'transcan':
print(x, long=FALSE, ...)

## S3 method for class 'transcan':
```

```
plot(x, ...)

## S3 method for class 'transcan':
impute(x, var, imputation, name, where.in, data,
       where.out=1, frame.out, list.out=FALSE, pr=TRUE, check=TRUE, ...)

fit.mult.impute(formula, fitter, xtrans, data, n.impute, fit.reps=FALSE,
                derived, pr=TRUE, subset, ...)

## S3 method for class 'transcan':
predict(object, newdata, iter.max=50, eps=0.01, curtail=TRUE,
        type=c("transformed","original"),
        inverse, tolInverse, check=FALSE, ...)

Function(object, ...)

## S3 method for class 'transcan':
Function(object, prefix=".", suffix="", where=1, ...)

invertTabulated(x, y, freq=rep(1,length(x)),
                aty, name='value',
                inverse=c('linearInterp','sample'),
                tolInverse=0.05, rule=2)

Varcov(object, ...)

## Default S3 method:
Varcov(object, regcoef.only=FALSE, ...)

## S3 method for class 'lm':
Varcov(object, ...)

## S3 method for class 'glm':
Varcov(object, ...)

## S3 method for class 'multinom':
Varcov(object, ...)

## S3 method for class 'fit.mult.impute':
Varcov(object, ...)
```

Arguments

`x`	a matrix containing continuous variable values and codes for categorical variables. The matrix must have column names (`dimnames`). If row names are present, they are used in forming the `names` attribute of imputed values if `imputed=TRUE`. `x` may also be a formula, in which case the model matrix is created automatically, using data in the calling frame. Advantages of using

a formula are that `categorical` variables can be determined automatically by a variable being a `factor` variable, and variables with two unique levels are modeled `asis`. Variables with 3 unique values are considered to be `categorical` if a formula is specified. For a formula you may also specify that a variable is to remain untransformed by enclosing its name with the identify function, e.g. `I(x3)`. The user may add other variable names to the `asis` and `categorical` vectors. For `invertTabulated`, `x` is a vector or a list with three components: the x vector, the corresponding vector of transformed values, and the corresponding vector of frequencies of the pair of original and transformed variables. For `print`, `plot`, `impute`, and `predict`, `x` is an object created by `transcan`.

`formula`	any S model formula
`fitter`	any S or Design modeling function (not in quotes) that computes a vector of `coefficients` and for which `Varcov` will return a variance-covariance matrix. E.g., `fitter=lm, glm, ols`. At present models involving non-regression parameters (e.g., scale parameters in parametric survival models) are not handled fully.
`xtrans`	an object created by `transcan`, `aregImpute`, or `Mice`
`method`	use `method="canonical"` or any abbreviation thereof, to use canonical variates (the default). `method="pc"` transforms a variable instead so as to maximize the correlation with the first principal component of the other variables.
`categorical`	a character vector of names of variables in `x` which are categorical, for which the ordering of re-scored values is not necessarily preserved. If `categorical` is omitted, it is assumed that all variables are continuous (or binary). Set `categorical="*"` to treat all variables as categorical.
`asis`	a character vector of names of variables that are not to be transformed. For these variables, the guts of `lm.fit.qr` is used to impute missing values. You may want to treat binary variables `asis` (this is automatic if using a formula). If imputed=TRUE, you may want to use `"categorical"` for binary variables if you want to force imputed values to be one of the original data values. Set `asis="*"` to treat all variables `asis`.
`nk`	number of knots to use in expanding each continuous variable (not listed in `asis`) in a restricted cubic spline function. Default is 3 (yielding 2 parameters for a variable) if `n < 30`, 4 if `30 <= n < 100`, and 5 if `n >= 100` (4 parameters).
`imputed`	Set to `TRUE` to return a list containing imputed values on the original scale. If the transformation for a variable is non-monotonic, imputed values are not unique. `transcan` uses the `approx` function, which returns the highest value of the variable with the transformed score equalling the imputed score. `imputed=TRUE` also causes original-scale imputed values to be shown as tick marks on the top margin of each graph when `show.na=TRUE` (for the final iteration only). For categorical predictors, these imputed values are `jitter`ed so that their frequencies can be visualized. When `n.impute` is used, each NA will have `n.impute` tick marks.
`n.impute`	number of multiple imputations. If omitted, single predicted expected value imputation is used. `n.impute=5` is frequently recommended.

`boot.method`	default is to use the approximate Bayesian bootstrap (sample with replacement from sample with replacement of the vector of residuals). You can also specify `boot.method="simple"` to use the usual bootstrap one-stage sampling with replacement.
`trantab`	Set to `TRUE` to add an attribute `trantab` to the returned matrix. This contains a vector of lists each with components `x` and `y` containing the unique values and corresponding transformed values for the columns of `x`. This is set up to be used easily with the `approx` function. You must specify `trantab=TRUE` if you want to later use the `predict.transcan` function with `type="original"`.
`transformed`	set to `TRUE` to cause `transcan` to return an object `transformed` containing the matrix of transformed variables
`impcat`	This argument tells how to impute categorical variables on the original scale. The default is `impcat="score"` to impute the category whose canonical variate score is closest to the predicted score. Use `impcat="tree"` to impute categorical variables using the `tree()` function, using the values of all other transformed predictors. `impcat="rpart"` will use `rpart`. A better but somewhat slower approach is to use `impcat="multinom"` to fit a multinomial logistic model to the categorical variable, at the last iteraction of the `transcan` algorithm. This uses the `multinom` function in the `nnet` library of the `MASS` package (which is assumed to have been installed by the user) to fit a polytomous logistic model to the current working transformations of all the other variables (using conditional mean imputation for missing predictors). Multiple imputations are made by drawing multinomial values from the vector of predicted probabilities of category membership for the missing categorical values.
`mincut`	If `imputed=TRUE`, there are categorical variables, and `impcat="tree"`, `mincut` specifies the lowest node size that will be allowed to be split by `tree`. The default is 40.
`inverse`	By default, imputed values are back-solved on the original scale using inverse linear interpolation on the fitted tabulated transformed values. This will cause distorted distributions of imputed values (e.g., floor and ceiling effects) when the estimated transformation has a flat or nearly flat section. To instead use the `invertTabulated` function (see above) with the `"sample"` option, specify `inverse="sample"`.
`tolInverse`	the multiplyer of the range of transformed values, weighted by `freq` and by the distance measure, for determining the set of x values having y values within a tolerance of the value of `aty` in `invertTabulated`. For `predict.transcan`, `inverse` and `tolInverse` are obtained from options that were specified to `transcan` by default. Otherwise, if not specified by the user, these default to the defaults used to `invertTabulated`.
`pr`	For `transcan`, set to `FALSE` to suppress printing r-squares and shrinkage factors. For `impute.transcan` set to `FALSE` to suppress messages concerning the number of NAs imputed, or for `fit.mult.impute` set to `FALSE` to suppress printing variance inflation factors accounting for imputation, rate of missing information, and degrees of freedom.
`pl`	Set to `FALSE` to suppress plotting the final transformations with distribution of scores for imputed values (if `show.na=TRUE`).

`allpl`	Set to `TRUE` to plot transformations for intermediate iterations.
`show.na`	Set to `FALSE` to suppress the distribution of scores assigned to missing values (as tick marks on the right margin of each graph). See also `imputed`.
`imputed.actual`	The default is `"none"` to suppress plotting of actual vs. imputed values for all variables having any NAs. Other choices are `"datadensity"` to use `datadensity` to make a single plot, `"hist"` to make a series of back-to-back histograms, `"qq"` to make a series of q-q plots, or `"ecdf"` to make a series of empirical cdfs. For `imputed.actual="datadensity"` for example you get a rug plot of the non-missing values for the variable with beneath it a rug plot of the imputed values. When `imputed.actual` is not `"none"`, `imputed` is automatically set to `TRUE`.
`iter.max`	maximum number of iterations to perform for `transcan` or `predict`. For `predict`, only one iteration is used if there are no NAs in the data or if `imp.con` was used.
`eps`	convergence criterion for `transcan` and `predict`. `eps` is the maximum change in transformed values from one iteration to the next. If for a given iteration all new transformations of variables differ by less than `eps` (with or without negating the transformation to allow for "flipping") from the transformations in the previous iteration, one more iteration is done for `transcan`. During this last iteration, individual transformations are not updated but coefficients of transformations are. This improves stability of coefficients of canonical variates on the right-hand-side. `eps` is ignored when `rhsImp="random"`.
`curtail`	for `transcan`, causes imputed values on the transformed scale to be truncated so that their ranges are within the ranges of non-imputed transformed values. For `predict`, `curtail` defaults to `TRUE` to truncate predicted transformed values to their ranges in the original fit (`xt`).
`imp.con`	for `transcan`, set to `TRUE` to impute NAs on the original scales with constants (medians or most frequent category codes). Set to a vector of constants to instead always use these constants for imputation. These imputed values are ignored when fitting the current working transformation for a single variable.
`shrink`	default is `FALSE` to use ordinary least squares or canonical variate estimates. For the purposes of imputing NAs, you may want to set `shrink=TRUE` to avoid overfitting when developing a prediction equation to predict each variables from all the others (see details below).
`init.cat`	method for initializing scorings of categorical variables. Default is `"mode"` to use a dummy variable set to 1 if the value is the most frequent value (this is the default). Use `"random"` to use a random 0-1 variable. Set to `"asis"` to use the original integer codes as starting scores.
`nres`	number of residuals to store if `n.impute` is specified. If the dataset has fewer than `nres` observations, all residuals are saved. Otherwise a random sample of the residuals of length `nres` without replacement is saved. The default for `nres` is higher if `boot.method="approximate bayesian"`.
`data`	
`subset`	an integer or logical vector specifying the subset of observations to fit

`na.action`	These may be used if `x` is a formula. The default `na.action` is `na.retain` (defined by `transcan`) which keeps all observations with any `NA`s. For `impute.transcan`, `data` is a data frame to use as the source of variables to be imputed, rather than using `where.in`. For `fit.mult.impute`, `data` is mandatory and is a data frame containing the data to be used in fitting the model but before imputations are applied. Variables omitted from `data` are assumed to be available from frame 1 and do not need to be imputed.
`treeinfo`	Set to `TRUE` to get additional information printed when `impcat="tree"`, such as the predicted probabilities of category membership.
`rhsImp`	Set to `"random"` to use random draw imputation when a sometimes missing variable is moved to be a predictor of other sometimes missing variables. Default is `rhsImp="mean"`, which uses conditional mean imputation on the transformed scale. Residuals used are residuals from the transformed scale. When `"random"` is used, `transcan` runs 5 iterations and ignores `eps`.
`details.impcat`	set to a character scalar that is the name of a category variable to include in the resulting `transcan` object an element `details.impcat` containing details of how the categorical variable was multiply imputed.
`...`	arguments passed to `scat1d` or to the `fitter` function (for `fit.mult.impute`)
`long`	for `summary`, set to `TRUE` to print all imputed values. For `print`, set to `TRUE` to print details of transformations/imputations.
`var`	For `impute`, is a variable that was originally a column in `x`, for which imputated values are to be filled in. `imputed=TRUE` must have been used in `transcan`. Omit `var` to impute all variables, creating new variables in `search` position `where`.
`imputation`	specifies which of the multiple imputations to use for filling in NAs
`name`	name of variable to impute, for `impute()`. Default is character string version of the second argument (`var`) in the call to `impute`. For `invertTabulated`, is the name of variable being transformed (used only for warning messages).
`where.in`	location in `search` list to find variables that need to be imputed, when all variables are to be imputed automatically by `impute.transcan` (i.e., when no input variable name is specified). Default is first `search` position that contains the first variable to be imputed.
`where.out`	location in the `search` list for storing variables with missing values set to imputed values, for `impute.transcan` when all variables with missing values are being imputed automatically.
`frame.out`	Instead of specifying `where.out` you can specify an S frame number into which individual new imputed variables will be written. For example, `frame.out=1` is useful for putting new variables into a temporary local frame when `impute` is called within another function (see `fit.mult.impute`). See `assign` for details about frames. For R, `where.out` and `frame.out` are ignored and results are stored in `.GlobalEnv` when `list.out` is not specified (it is recommended to use `list.out=TRUE`).
`list.out`	If `var` is not specified, you can set `list.out=TRUE` to have `impute.transcan` return a list containing variables with needed values imputed. This list will contain a single imputation.

`check`	set to `FALSE` to suppress certain warning messages
`newdata`	a new data matrix for which to compute transformed variables. Categorical variables must use the same integer codes as were used in the call to `transcan`. If a formula was originally specified to `transcan` (instead of a data matrix), `newdata` is optional and if given must be a data frame; a model frame is generated automatically from the previous formula. The `na.action` is handled automatically, and the levels for factor variables must be the same and in the same order as were used in the original variables specified in the formula given to `transcan`.
`fit.reps`	set to `TRUE` to save all fit objects from the fit for each imputation in `fit.mult.impute`. Then the object returned will have a component `fits` which is a list whose `i`th element is the `i`th fit object.
`derived`	an expression containing S expressions for computing derived variables that are used in the model formula. This is useful when multiple imputations are done for component variables but the actual model uses combinations of these (e.g., ratios or other derivations). For a single derived variable you can specified for example `derived=expression(ratio <- weight/height)`. For multiple derived variables use the form `derived=expression({ratio <- weight/height; product <- weight*height})` or put the expression on separate input lines. To monitor the multiply-imputed derived variables you can add to the `expression` a command such as `print(describe(ratio))`. See the example below.
`type`	By default, the matrix of transformed variables is returned, with imputed values on the transformed scale. If you had specified `trantab=TRUE` to `transcan`, specifying `type="original"` does the table look-ups with linear interpolation to return the input matrix `x` but with imputed values on the original scale inserted for NAs. For categorical variables, the method used here is to select the category code having a corresponding scaled value closest to the predicted transformed value. This corresponds to the default `impcat`; a problem in getting predicted values for `tree` objects prevented using `tree` for this. Note: imputed values thus returned when `type="original"` are single expected value imputations even in `n.impute` is given.
`object`	an object created by `transcan`, or an object to be converted to S function code, typically a model fit object of some sort
`prefix`	
`suffix`	When creating separate S functions for each variable in `x`, the name of the new function will be `prefix` placed in front of the variable name, and `suffix` placed in back of the name. The default is to use names of the form `.varname`, where `varname` is the variable name.
`where`	position in `search` list at which to store new functions (for `Function`). Default is position 1 in the search list. See the `assign` function for more documention on the `where` argument.
`y`	a vector corresponding to `x` for `invertTabulated`, if its first argument `x` is not a list
`freq`	a vector of frequencies corresponding to cross-classified `x` and `y` if `x` is not a list. Default is a vector of ones.

`aty`	vector of transformed values at which inverses are desired
`rule`	see `approx`. `transcan` assumes `rule` is always `2`
`regcoef.only`	set to `TRUE` to make `Varcov.default` delete positions in the covariance matrix for any non-regression coefficients (e.g., log scale parameter from `psm` or `survreg`)

Details

The starting approximation to the transformation for each variable is taken to be the original coding of the variable. The initial approximation for each missing value is taken to be the median of the non-missing values for the variable (for continuous ones) or the most frequent category (for categorical ones). Instead, if `imp.con` is a vector, its values are used for imputing NAs. When using each variable as a dependent variable, NAs on that variable cause all observations to be temporarily deleted. Once a new working transformation is found for the variable, along with a model to predict that transformation from all the other variables, that latter model is used to impute NAs in the selected dependent variable if `imp.con` is not specified. When that variable is used to predict a new dependent variable, the current working imputed values are inserted. Transformations are updated after each variable becomes a dependent variable, so the order of variables on `x` could conceivably make a difference in the final estimates. For obtaining out-of-sample predictions/transformations, `predict` uses the same iterative procedure as `transcan` for imputation, with the same starting values for fill-ins as were used by `transcan`. It also (by default) uses a conservative approach of curtailing transformed variables to be within the range of the original ones. Even when `method="pc"` is specified, canonical variables are used for imputing missing values.

Note that fitted transformations, when evaluated at imputed variable values (on the original scale), will not precisely match the transformed imputed values returned in `xt`. This is because `transcan` uses an approximate method based on linear interpolation to back-solve for imputed values on the original scale.

Shrinkage uses the method of Van Houwelingen and Le Cessie (1990) (similar to Copas, 1983). The shrinkage factor is `[1-(1-R2)(n-1)/(n-k-1)]/R2`, where `R2` is the apparent R-squared for predicting the variable, `n` is the number of non-missing values, and `k` is the effective number of degrees of freedom (aside from intercepts). A heuristic estimate is used for `k`: `A - 1 + sum(max(0,Bi-1))/m + m`, where `A` is the number of d.f. required to represent the variable being predicted, the `Bi` are the number of columns required to represent all the other variables, and `m` is the number of all other variables. Division by `m` is done because the transformations for the other variables are fixed at their current transformations the last time they were being predicted. The `+ m` term comes from the number of coefficients estimated on the right hand side, whether by least squares or canonical variates. If a shrinkage factor is negative, it is set to 0. The shrinkage factor is the ratio of the adjusted R-squared to the ordinary R-squared. The adjusted R-squared is `1 - (1 - R2)(n-1)/(n-k-1)`, which is also set to zero if it is negative. If `shrink=FALSE` and the adjusted R-squares are much smaller than the ordinary R-squares, you may want to run `transcan` with `shrink=TRUE`.

Canonical variates are scaled to have variance of 1.0, by multiplying canonical coefficients from `cancor` by `sqrt(n-1)`.

When specifying a non-Design library fitting function to `fit.mult.impute` (e.g., `lm`, `glm`), running the result of `fit.mult.impute` through that fit's `summary` method will not use the imputation-adjusted variances. You may obtain the new variances using `fit$var` or `Varcov(fit)`.

When you specify a Design function to `fit.mult.impute` (e.g., `lrm, ols, cph, psm, bj`), automatically computed transformation parameters (e.g., knot locations for `rcs`) that are estimated for the first imputation are used for all other imputations. This ensures that knot locations will not vary, which would change the meaning of the regression coefficients.

Warning: even though `fit.mult.impute` takes imputation into account when estimating variances of regression coefficient, it does not take into account the variation that results from estimation of the shapes and regression coefficients of the customized imputation equations. Specifying `shrink=TRUE` solves a small part of this problem. To fully account for all sources of variation you should consider putting the `transcan` invocation inside a bootstrap or loop, if execution time allows. Better still, use `aregImpute` or one of the libraries such as MICE that uses real Bayesian posterior realizations to multiply impute missing values correctly.

It is strongly recommended that you use the Hmisc `naclus` function to determine is there is a good basis for imputation. `naclus` will tell you, for example, if systolic blood pressure is missing whenever diastolic blood pressure is missing. If the only variable that is well correlated with diastolic bp is systolic bp, there is no basis for imputing diastolic bp in this case.

At present, `predict` does not work with multiple imputation.

When calling `fit.mult.impute` with `glm` as the `fitter` argument, if you need to pass a `family` argument to `glm` do it by quoting the family, e.g., `family="binomial"`.

You should be able to use a variable in the formula given to `fit.mult.impute` as a numeric variable in the regression model even though it was a factor variable in the invocation of `transcan`. Use for example `fit.mult.impute(y ~ codes(x), lrm, trans)` (thanks to Trevor Thompson ⟨trevor@hp5.eushc.org⟩).

Value

For `transcan`, a list of class `transcan` with elements `call` (with the function call), `iter` (number of iterations done) and `rsq` and `rsq.adj` containing the R-squares and adjusted R-squares achieved in predicting each variable from all the others. It also has elements `categorical`, `asis`, `coef`, `xcoef`, `parms`, `fillin`, `ranges`, `scale`, and `formula` containing respectively the values supplied for `categorical` and `asis`, the within-variable coefficients used to compute the first canonical variate, the (possibly shrunk) across-variables coefficients of the first canonical variate that predicts each variable in turn, the parameters of the transformation (knots for splines, contrast matrix for categorical variables), the initial estimates for missing values (NA if variable never missing), the matrix of ranges of the transformed variables (min and max in first and second row), a vector of scales used to determine convergence for a transformation, the formula (if `x` was a formula), and optionally a vector of shrinkage factors used for predicting each variable from the others. For `"asis"` variables, the scale is the average absolute difference about the median. For other variables it is unity, since canonical variables are standardized. For `xcoef`, row `i` has the coefficients to predict transformed variable `i`, with the column for the coefficient of variable `i` set to NA. If `imputed=TRUE` was given, an optional element `imputed` also appears. This is a list with the vector of imputed values (on the original scale) for each variable containing NAs. Matrices rather than vectors are returned if `n.impute` is given. If `trantab=TRUE, the 'trantab` element also appears, as described above. If `n.impute > 0`, `transcan` also returns a list `residuals` that can be used for future multiple imputation.

`impute` returns a vector (the same length as `var`) of class `"impute"` with NAs imputed. `predict` returns a matrix with the same number of columns or variables as were in `x`.

`fit.mult.impute` returns a fit object that is a modification of the fit object created by fitting the completed dataset for the final imputation. The `var` matrix in the fit object has the imputation-corrected variance-covariance matrix. `coefficients` is the average (over imputations) of the coefficient vectors, `variance.inflation.impute` is a vector containing the ratios of the diagonals of the between-imputation variance matrix to the diagonals of the average apparent (within-imputation) variance matrix. `missingInfo` is Rubin's "rate of missing information" and `dfmi` is Rubin's degrees of freedom for a t-statistic for testing a single parameter. The last two objects are vectors corresponding to the diagonal of the variance matrix.

Side Effects

prints, plots, and `impute.transcan` creates new variables.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

References

Kuhfeld, Warren F: The PRINQUAL Procedure. SAS/STAT User's Guide, Fourth Edition, Volume 2, pp. 1265–1323, 1990.

Van Houwelingen JC, Le Cessie S: Predictive value of statistical models. Statistics in Medicine 8:1303–1325, 1990.

Copas JB: Regression, prediction and shrinkage. JRSS B 45:311–354, 1983.

He X, Shen L: Linear regression after spline transformation. Biometrika 84:474–481, 1997.

Little RJA, Rubin DB: Statistical Analysis with Missing Data. New York: Wiley, 1987.

Rubin DJ, Schenker N: Multiple imputation in health-care databases: An overview and some applications. Stat in Med 10:585–598, 1991.

Faris PD, Ghali WA, et al:Multiple imputation versus data enhancement for dealing with missing data in observational health care outcome analyses. J Clin Epidem 55:184–191, 2002.

See Also

`aregImpute`, `impute`, `naclus`, `naplot`, `ace`, `avas`, `cancor`, `prcomp`, `rcspline.eval`, `lsfit`, `approx`, `datadensity`, `mice`

Examples

```
## Not run:
x <- cbind(age, disease, blood.pressure, pH)
#cbind will convert factor object `disease' to integer
par(mfrow=c(2,2))
x.trans <- transcan(x, categorical="disease", asis="pH",
                    transformed=TRUE, imputed=TRUE)
summary(x.trans)  #Summary distribution of imputed values, and R-squares
```

```
f <- lm(y ~ x.trans$transformed)   #use transformed values in a regression
#Now replace NAs in original variables with imputed values, if not
#using transformations
age            <- impute(x.trans, age)
disease        <- impute(x.trans, disease)
blood.pressure <- impute(x.trans, blood.pressure)
pH             <- impute(x.trans, pH)
#Do impute(x.trans) to impute all variables, storing new variables under
#the old names
summary(pH)       #uses summary.impute to tell about imputations
                  #and summary.default to tell about pH overall
# Get transformed and imputed values on some new data frame xnew
newx.trans     <- predict(x.trans, xnew)
w              <- predict(x.trans, xnew, type="original")
age            <- w[,"age"]            #inserts imputed values
blood.pressure <- w[,"blood.pressure"]
Function(x.trans)  #creates .age, .disease, .blood.pressure, .pH()
#Repeat first fit using a formula
x.trans <- transcan(~ age + disease + blood.pressure + I(pH),
                    imputed=TRUE)
age <- impute(x.trans, age)
predict(x.trans, expand.grid(age=50, disease="pneumonia",
        blood.pressure=60:260, pH=7.4))
z <- transcan(~ age + factor(disease.code),  # disease.code categorical
              transformed=TRUE, trantab=TRUE, imputed=TRUE, pl=FALSE)
plot(z$transformed)
## End(Not run)

# Multiple imputation and estimation of variances and covariances of
# regression coefficient estimates accounting for imputation
set.seed(1)
x1 <- factor(sample(c('a','b','c'),100,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(100)
y  <- x2 + 1*(x1=='c') + rnorm(100)
x1[1:20] <- NA
x2[18:23] <- NA
d <- data.frame(x1,x2,y)
n <- naclus(d)
plot(n); naplot(n)  # Show patterns of NAs
f  <- transcan(~y + x1 + x2, n.impute=10, shrink=FALSE, data=d)
options(digits=3)
summary(f)

f  <- transcan(~y + x1 + x2, n.impute=10, shrink=TRUE, data=d)
summary(f)

h <- fit.mult.impute(y ~ x1 + x2, lm, f, data=d)
# Add ,fit.reps=TRUE to save all fit objects in h, then do something like:
# for(i in 1:length(h$fits)) print(summary(h$fits[[i]]))

diag(Varcov(h))

h.complete <- lm(y ~ x1 + x2, na.action=na.omit)
```

```
h.complete
diag(Varcov(h.complete))

# Note: had Design's ols function been used in place of lm, any
# function run on h (anova, summary, etc.) would have automatically
# used imputation-corrected variances and covariances

# Example demonstrating how using the multinomial logistic model
# to impute a categorical variable results in a frequency
# distribution of imputed values that matches the distribution
# of non-missing values of the categorical variable

## Not run:
set.seed(11)
x1 <- factor(sample(letters[1:4], 1000,TRUE))
x1[1:200] <- NA
table(x1)/sum(table(x1))
x2 <- runif(1000)
z  <- transcan(~ x1 + I(x2), n.impute=20, impcat='multinom')
table(z$imputed$x1)/sum(table(z$imputed$x1))
## End(Not run)

# Example where multiple imputations are for basic variables and
# modeling is done on variables derived from these

set.seed(137)
n <- 400
x1 <- runif(n)
x2 <- runif(n)
y  <- x1*x2 + x1/(1+x2) + rnorm(n)/3
x1[1:5] <- NA
d <- data.frame(x1,x2,y)
w <- transcan(~ x1 + x2 + y, n.impute=5, data=d)
# Add ,show.imputed.actual for graphical diagnostics
## Not run:
g <- fit.mult.impute(y ~ product + ratio, ols, w,
                     data=data.frame(x1,x2,y),
                     derived=expression({
                       product <- x1*x2
                       ratio   <- x1/(1+x2)
                       print(cbind(x1,x2,x1*x2,product)[1:6,])}))
## End(Not run)

# Here's a method for creating a permanent data frame containing
# one set of imputed values for each variable specified to transcan
# that had at least one NA, and also containing all the variables
# in an original data frame.  The following is based on the fact
# that the default output location for impute.transcan is
# given by where.out=1 (search position 1)

## Not run:
xt <- transcan(~. , data=mine,
               imputed=TRUE, shrink=TRUE, n.impute=10, trantab=TRUE)
```

```
attach(mine, pos=1, use.names=FALSE)
impute(xt, imputation=1) # use first imputation
# omit imputation= if using single imputation
detach(1, 'mine2')
## End(Not run)

# Example of using invertTabulated outside transcan
x    <- c(1,2,3,4,5,6,7,8,9,10)
y    <- c(1,2,3,4,5,5,5,5,9,10)
freq <- c(1,1,1,1,1,2,3,4,1,1)
# x=5,6,7,8 with prob. .1 .2 .3 .4 when y=5
# Within a tolerance of .05*(10-1) all y's match exactly
# so the distance measure does not play a role
set.seed(1)      # so can reproduce
for(inverse in c('linearInterp','sample'))
 print(table(invertTabulated(x, y, freq, rep(5,1000), inverse=inverse)))

# Test inverse='sample' when the estimated transformation is
# flat on the right.  First show default imputations
set.seed(3)
x <- rnorm(1000)
y <- pmin(x, 0)
x[1:500] <- NA
for(inverse in c('linearInterp','sample')) {
par(mfrow=c(2,2))
  w <- transcan(~ x + y, imputed.actual='hist',
                inverse=inverse, curtail=FALSE,
                data=data.frame(x,y))
  if(inverse=='sample') next
# cat('Click mouse on graph to proceed\n')
# locator(1)
}
```

---

translate   *Translate Vector or Matrix of Text Strings*

Description

Uses the UNIX tr command to translate any character in `old` in `text` to the corresponding character in `new`. If multichar=T or `old` and `new` have more than one element, or each have one element but they have different numbers of characters, uses the UNIX `sed` command to translate the series of characters in `old` to the series in `new` when these characters occur in `text`. If `old` or `new` contain a backslash, you sometimes have to quadruple it to make the UNIX command work. If they contain a forward slash, preceed it by two backslashes. The Microsoft Windows version of `translate` invokes the `sedit()` function and does not allow `multichar=FALSE`, i.e., it does not support the UNIX `tr` function. The R version of `translate` invokes the builtin chartr function if `multichar=FALSE`.

Usage

```
translate(text, old, new, multichar=FALSE)
```

Arguments

`text`	scalar, vector, or matrix of character strings to translate.
`old`	vector old characters
`new`	corresponding vector of new characters
`multichar`	See above.

Details

At present, `multichar=FALSE`, which requires the UNIX `tr` program, is not implemented under MS Windows.

Value

an object like text but with characters translated

See Also

unix, grep

Examples

```
translate(c("ABC","DEF"),"ABCDEFG", "abcdefg")
translate("23.12","[.]","\\\cdot ") # change . to \cdot
translate(c("dog","cat","tiger"),c("dog","cat"),c("DOG","CAT"))
# S-Plus gives  [1] "DOG"   "CAT"   "tiger" - check discrepency
translate(c("dog","cat2","snake"),c("dog","cat"),"animal")
# S-Plus gives  [1] "animal"  "animal2" "snake"
```

`trunc.POSIXt` *return the floor, ceiling, or rounded value of date or time to specified unit.*

Description

`trunc.POSIXt` returns the date truncated to the specified unit. `ceiling.POSIXt` returns next ceiling of the date at the unit selected in `units`. `floor.POSIXt` `trunk.POSIXt` by another name. `round.POSIXt` returns the date or time value rounded to nearest specified unit selected in `digits`.

`trunc.POSIXt` and `round.POSIXt` have been extended from the `base` package functions.

Usage

```
ceil(x, units,...)
## Default S3 method:
ceil(x, units, ...)
## S3 method for class 'POSIXt':
trunc(x, units = c("secs", "mins", "hours", "days",
"months", "years"), ...)
## S3 method for class 'POSIXt':
ceil(x, units = c("secs", "mins", "hours", "days",
"months", "years"), ...)
## S3 method for class 'POSIXt':
round(x, digits = c("secs", "mins", "hours", "days", "months", "years"))
```

Arguments

`x`	date to be floored, ceilinged, truncated, or rounded
`units`	unit to that is is rounded up or down to.
`digits`	same as `units` but different name to be compatible with `round` generic.
`...`	further arguments to be passed to or from other methods.

Value

An object of class `POSIXlt`.

Author(s)

Charles Dupont

See Also

`Date` `POSIXt` `POSIXlt` `DateTimeClasses`

Examples

```
date <- ISOdate(1832, 7, 12)
ceil(date, units='months')  # '1832-8-1'
trunc(date, units='years')     # '1832-1-1'
round.POSIXt(date, digits='months')    # '1832-7-1'
```

`units` *Units Attribute of a Vector*

Description

Sets or retrieves the `"units"` attribute of an object. For `units.default` replaces the builtin version, which only works for time series objects. If the variable is also given a `label`, subsetting (using `[.labelled`) will retain the `"units"` attribute. For S-Plus 6 which uses version 4 of the S language, the latter does not work.

Usage

```
units(x, ...)
## Default S3 method:
units(x, none='', ...)
## S3 replacement method for class 'default':
units(x) <- value
```

Arguments

`x`	any object
`...`	ignored
`value`	the units of the object, or ""
`none`	value to which to set result if no appropriate attribute is found

Value

the units attribute of x, if any; otherwise, the `units` attribute of the `tspar` attribute of `x` if any; otherwise the value `none`

See Also

`label`

Examples

```
fail.time <- c(10,20)
units(fail.time) <- "Day"
describe(fail.time)
label(fail.time) <- 'Failure Time'
fail.time
## Not run:
f <- cph(Surv(fail.time, event) ~ xx)
plot(xx,xx2,xlab=paste(label(xx),", ",units(xx),"s",sep=""))
## End(Not run)
```

`upData` *Update a Data Frame or Cleanup a Data Frame after Importing*

Description

`cleanup.import` will correct errors and shrink the size of data frames created by the S-Plus `File ... Import` dialog or by other methods such as `scan` and `read.table`. By default, double precision numeric variables are changed to single precision (S-Plus only) or to integer when they contain no fractional components. Infinite values or values greater than 1e20 in absolute value are set to NA. This solves problems of importing Excel spreadsheets that contain occasional character values for numeric columns, as S-Plus converts these to `Inf` without warning. There is also an option to convert variable names to lower case and to add labels to variables. The latter can be made easier by importing a CNTLOUT dataset created by SAS PROC FORMAT and using the `sasdict` option as shown in the example below. `cleanup.import` can also transform character or factor variables to dates.

`upData` is a function facilitating the updating of a data frame without attaching it in search position one. New variables can be added, old variables can be modified, variables can be removed or renamed, and `"labels"` and `"units"` attributes can be provided. Various checks are made for errors and inconsistencies, with warnings issued to help the user. Levels of factor variables can be replaced, especially using the `list` notation of the standard `merge.levels` function. Unless `force.single` is set to `FALSE`, `upData` also converts double precision vectors to single precision (if not under R), or to integer if no fractional values are present in a vector.

Both `cleanup.import` and `upData` will fix a problem with data frames created under S-Plus before version 5 that are used in S-Plus 5 or later. The problem was caused by use of the `label` function to set a variable's class to `"labelled"`. These classes are removed as the S version 4 language does not support multiple inheritance. Failure to run data frames through one of the two functions when these conditions apply will result in simple numeric variables being set to `factor` in some cases. Extraneous `"AsIs"` classes are also removed.

For S-Plus, a function `exportDataStripped` is provided that allows exporting of data to other systems by removing attributes `label, imputed, format, units`, and `comment`. It calls `exportData` after stripping these attributes. Otherwise `exportData` will fail.

The `dataframeReduce` function removes variables from a data frame that are problematic for certain analyses. Variables can be removed because the fraction of missing values exceeds a threshold, because they are character or categorical variables having too many levels, or because they are binary and have too small a prevalence in one of the two values. Categorical variables can also have their levels combined when a level is of low prevalence.

Usage

```
cleanup.import(obj, labels, lowernames=FALSE,
               force.single=TRUE, force.numeric=TRUE, rmnames=TRUE,
               big=1e20, sasdict, pr, datevars=NULL, datetimevars=NULL,
               dateformat='%F',
               fixdates=c('none','year'), charfactor=FALSE)

upData(object, ...,
       rename, drop, labels, units, levels,
       force.single=TRUE, lowernames=FALSE, moveUnits=FALSE, charfactor=FALSE)

exportDataStripped(data, ...)

dataframeReduce(data, fracmiss=1, maxlevels=NULL,  minprev=0, pr=TRUE)
```

Arguments

`obj`	a data frame or list
`object`	a data frame or list
`data`	a data frame

`force.single`	By default, double precision variables are converted to single precision (in S-Plus only) unless `force.single=FALSE`. `force.single=TRUE` will also convert vectors having only integer values to have a storage mode of integer, in R or S-Plus.
`force.numeric`	Sometimes importing will cause a numeric variable to be changed to a factor vector. By default, `cleanup.import` will check each factor variable to see if the levels contain only numeric values and `""`. In that case, the variable will be converted to numeric, with `""` converted to NA. Set `force.numeric=FALSE` to prevent this behavior.
`rmnames`	set to 'F' to not have 'cleanup.import' remove 'names' or '.Names' attributes from variables
`labels`	a character vector the same length as the number of variables in `obj`. These character values are taken to be variable labels in the same order of variables in `obj`. For `upData`, `labels` is a named list or named vector with variables in no specific order.
`lowernames`	set this to `TRUE` to change variable names to lower case. `upData` does this before applying any other changes, so variable names given inside arguments to `upData` need to be lower case if `lowernames==TRUE`.
`big`	a value such that values larger than this in absolute value are set to missing by `cleanup.import`
`sasdict`	the name of a data frame containing a raw imported SAS PROC CONTENTS CNTLOUT= dataset. This is used to define variable names and to add attributes to the new data frame specifying the original SAS dataset name and label.
`pr`	set to `TRUE` or `FALSE` to force or prevent printing of the current variable number being processed. By default, such messages are printed if the product of the number of variables and number of observations in `obj` exceeds 500,000. For `dataframeReduce` set `pr` to `FALSE` to suppress printing information about dropped or modified variables.
`datevars`	character vector of names (after `lowernames` is applied) of variables to consider as a factor or character vector containing dates in a format matching `dateformat`. The default is `"%F"` which uses the yyyy-mm-dd format.
`datetimevars`	character vector of names (after `lowernames` is applied) of variables to consider to be date-time variables, with date formats as described under `datevars` followed by a space followed by time in hh:mm:ss format. `chron` is used to store date-time variables. If all times in the variable are 00:00:00 the variable will be converted to an ordinary date variable.
`dateformat`	for `cleanup.import` is the input format (see `strptime`)
`fixdates`	for any of the variables listed in `datevars` that have a `dateformat` that `cleanup.import` understands, specifying `fixdates` allows corrections of certain formatting inconsistencies before the fields are attempted to be converted to dates (the default is to assume that the `dateformat` is followed for all observation for `datevars`). Currently `fixdates='year'` is implemented, which will cause 2-digit or 4-digit years to be shifted to the alternate number of digits when `dateform` is the default `"%F"` or is `"%y-%m-%d"`,

	"%m/%d/%y", or "%m/%d/%Y". Two-digits years are padded with 20 on the left. Set dateformat to the desired format, not the exceptional format.
charfactor	set to TRUE to change character variables to factors if they have at least two characters in an observation but have fewer than n/2 unique values
...	for upData, one or more expressions of the form variable=expression, to derive new variables or change old ones. For exportDataStripped, optional arguments that are passed to exportData.
rename	list or named vector specifying old and new names for variables. Variables are renamed before any other operations are done. For example, to rename variables age and sex to respectively Age and gender, specify rename=list(age="Age", sex="gender") or rename=c(age=...).
drop	a vector of variable names to remove from the data frame
units	a named vector or list defining "units" attributes of variables, in no specific order
levels	a named list defining "levels" attributes for factor variables, in no specific order. The values in this list may be character vectors redefining levels (in order) or another list (see merge.levels if using S-Plus).
moveUnits	set to TRUE to look for units of measurements in variable labels and move them to a "units" attribute. If an expression in a label is enclosed in parentheses or brackets it is assumed to be units if moveUnits=TRUE.
fracmiss	the maximum permissable proportion of NAs for a variable to be kept. Default is to keep all variables no matter how many NAs are present.
maxlevels	the maximum number of levels of a character or categorical or factor variable before the variable is dropped
minprev	the minimum proportion of non-missing observations in a category for a binary variable to be retained, and the minimum relative frequency of a category before it will be combined with other small categories

Value

a new data frame

Author(s)

Frank Harrell, Vanderbilt University

See Also

sas.get, data.frame, describe, label, read.csv, strptime, POSIXct,Date

Examples

```
## Not run:
dat <- read.table('myfile.asc')
dat <- cleanup.import(dat)
## End(Not run)
dat <- data.frame(a=1:3, d=c('01/02/2004',' 1/3/04',''))
```

```
cleanup.import(dat, datevars='d', dateformat='%m/%d/%y', fixdates='year')

dat <- data.frame(a=(1:3)/7, y=c('a','b1','b2'), z=1:3)
dat2 <- upData(dat, x=x^2, x=x-5, m=x/10,
               rename=c(a='x'), drop='z',
               labels=c(x='X', y='test'),
               levels=list(y=list(a='a',b=c('b1','b2'))))
dat2
describe(dat2)
dat <- dat2    # copy to original name and delete dat2 if OK
rm(dat2)

# Remove hard to analyze variables from a redundancy analysis of all
# variables in the data frame
d <- dataframeReduce(dat, fracmiss=.1, minprev=.05, maxlevels=5)
# Could run redun(~., data=d) at this point or include dataframeReduce
# arguments in the call to redun

# If you import a SAS dataset created by PROC CONTENTS CNTLOUT=x.datadict,
# the LABELs from this dataset can be added to the data.  Let's also
# convert names to lower case for the main data file
## Not run:
mydata2 <- cleanup.import(mydata2, lowernames=TRUE, sasdict=datadict)
## End(Not run)
```

valueTags *Store Discriptive Information About an Object*

Description

Functions get or set useful information about the contents of the object for later use.

Usage

```
valueTags(x)
valueTags(x) <- value

valueLabel(x)
valueLabel(x) <- value

valueName(x)
valueName(x) <- value

valueUnit(x)
valueUnit(x) <- value
```

Arguments

`x`	an object
`value`	for `valueTags<-` a named list of value tags. a character vector of length 1, or `NULL`.

Details

These functions store the a short name of for the contents, a longer label that is useful for display, and the units of the contents that is useful for display.

`valueTag` is an accessor, and `valueTag<-` is a replacement function for all of the value's information.

`valueName` is an accessor, and `valueName<-` is a replacement function for the value's name. This name is used when a plot or a latex table needs a short name and the variable name is not useful.

`valueLabel` is an accessor, and `valueLabel<-` is a replacement function for the value's label. The label is used in a plots or latex tables when they need a descriptive name.

`valueUnit` is an accessor, and `valueUnit<-` is a replacement function for the value's unit. The unit is used to add unit information to the R output.

Value

`valueTag` returns `NULL` or a named list with each of the named values `name`, `label`, `unit` set if they exists in the object.

For `valueTag<-` returns `list`

For `valueName`, `valueLable`, and `valueUnit` returns `NULL` or character vector of length 1.

For `valueName<-`, `valueLabel<-`, and `valueUnit` returns `value`

Author(s)

Charles Dupont

See Also

`names`, `attributes`

Examples

```
age <- c(21,65,43)
y   <- 1:3
valueLabel(age) <- "Age in Years"
plot(age, y, xlab=valueLabel(age))

x1 <- 1:10
x2 <- 10:1
valueLabel(x2) <- 'Label for x2'
valueUnit(x2) <- 'mmHg'
x2
```

```
x2[1:5]
dframe <- data.frame(x1, x2)
Label(dframe)

##In these examples of llist, note that labels are printed after
##variable names, because of print.labelled
a <- 1:3
b <- 4:6
valueLabel(b) <- 'B Label'
```

`varclus` *Variable Clustering*

Description

Does a hierarchical cluster analysis on variables, using the Hoeffding D statistic, squared Pearson or Spearman correlations, or proportion of observations for which two variables are both positive as similarity measures. Variable clustering is used for assessing collinearity, redundancy, and for separating variables into clusters that can be scored as a single variable, thus resulting in data reduction. For computing any of the three similarity measures, pairwise deletion of NAs is done. The clustering is done by `hclust()`. A small function `naclus` is also provided which depicts similarities in which observations are missing for variables in a data frame. The similarity measure is the fraction of `NAs` in common between any two variables. The diagonals of this `sim` matrix are the fraction of NAs in each variable by itself. `naclus` also computes `na.per.obs`, the number of missing variables in each observation, and `mean.na`, a vector whose ith element is the mean number of missing variables other than variable i, for observations in which variable i is missing. The `naplot` function makes several plots (see the `which` argument).

So as to not generate too many dummy variables for multi-valued character or categorical predictors, `varclus` will automatically combine infrequent cells of such variables using an auxiliary function `combine.levels` that is defined here.

`plotMultSim` plots multiple similarity matrices, with the similarity measure being on the x-axis of each subplot.

`na.pattern` prints a frequency table of all combinations of missingness for multiple variables. If there are 3 variables, a frequency table entry labeled `110` corresponds to the number of observations for which the first and second variables were missing but the third variable was not missing.

Usage

```
varclus(x, similarity=c("spearman","pearson","hoeffding","bothpos","ccbothpos"),
        type=c("data.matrix","similarity.matrix"),
        method=if(.R.)"complete" else "compact",
        data=NULL, subset=NULL, na.action=na.retain, ...)
## S3 method for class 'varclus':
print(x, abbrev=FALSE, ...)
## S3 method for class 'varclus':
plot(x, ylab, abbrev=FALSE, legend.=FALSE, loc, maxlen, labels, ...)
```

```
naclus(df, method)
naplot(obj, which=c('all','na per var','na per obs','mean na',
                    'na per var vs mean na'), ...)

combine.levels(x, minlev=.05)

plotMultSim(s, x=1:dim(s)[3],
            slim=range(pretty(c(0,max(s,na.rm=TRUE)))),
            slimds=FALSE,
            add=FALSE, lty=par('lty'), col=par('col'),
            lwd=par('lwd'), vname=NULL, h=.5, w=.75, u=.05,
            labelx=TRUE, xspace=.35)

na.pattern(x)
```

Arguments

`x`	a formula, a numeric matrix of predictors, or a similarity matrix. If `x` is a formula, `model.matrix` is used to convert it to a design matrix. If the formula excludes an intercept (e.g., `~ a + b -1`), the first categorical (`factor`) variable in the formula will have dummy variables generated for all levels instead of omitting one for the first level. For `combine.levels`, `x` is a character, category, or factor vector (or other vector that is converted to factor). For `plot` and `print`, `x` is an object created by `varclus`. For `na.pattern`, `x` is a list, data frame, or numeric matrix. For `plotMultSim`, is a numeric vector specifying the ordered unique values on the x-axis, corresponding to the third dimension of `s`.
`df`	a data frame
`s`	an array of similarity matrices. The third dimension of this array corresponds to different computations of similarities. The first two dimensions come from a single similarity matrix. This is useful for displaying similarity matrices computed by `varclus`, for example. A use for this might be to show pairwise similarities of variables across time in a longitudinal study (see the example below). If `vname` is not given, `s` must have `dimnames`.
`similarity`	the default is to use squared Spearman correlation coefficients, which will detect monotonic but nonlinear relationships. You can also specify linear correlation or Hoeffding's (1948) D statistic, which has the advantage of being sensitive to many types of dependence, including highly non-monotonic relationships. For binary data, or data to be made binary, `similarity="bothpos"` uses as a similarity measure the proportion of observations for which two variables are both positive. `similarity="ccbothpos"` uses a chance-corrected measure which is the proportion of observations for which both variables are positive minus the product of the two marginal proportions. This difference is expected to be zero under independence. For diagonals, `"ccbothpos"` still uses the proportion of positives for the single variable. So `"ccbothpos"` is not really a similarity measure, and clustering is not done. This measure is useful for plotting with `plotMultSim` (see the last example).

`type`	if `x` is not a formula, it may be a data matrix or a similarity matrix. By default, it is assumed to be a data matrix.
`method`	see `hclust`. The default, for both `varclus` and `naclus`, is `"compact"` (for R it is `"complete"`).
`data`	
`subset`	
`na.action`	These may be specified if `x` is a formula. The default `na.action` is `na.retain`, defined by `varclus`. This causes all observations to be kept in the model frame, with later pairwise deletion of `NA`s.
`...`	for `varclus` these are optional arguments to pass to the `dataframeReduce` function. Otherwise, passed to `plclust` (or to `dotchart` or `dotchart2` for `naplot`).
`ylab`	y-axis label. Default is constructed on the basis of `similarity`.
`legend.`	set to `TRUE` to plot a legend defining the abbreviations
`loc`	a list with elements `x` and `y` defining coordinates of the upper left corner of the legend. Default is `locator(1)`.
`maxlen`	if a legend is plotted describing abbreviations, original labels longer than `maxlen` characters are truncated at `maxlen`.
`labels`	a vector of character strings containing labels corresponding to columns in the similar matrix, if the column names of that matrix are not to be used
`obj`	an object created by `naclus`
`which`	defaults to `"all"` meaning to have `naplot` make 4 separate plots. To make only one of the plots, use `which="na per var"` (dot chart of fraction of NAs for each variable), ,`"na per obs"` (dot chart showing frequency distribution of number of variables having NAs in an observation), `"mean na"` (dot chart showing mean number of other variables missing when the indicated variable is missing), or `"na per var vs mean na"`, a scatterplot showing on the x-axis the fraction of NAs in the variable and on the y-axis the mean number of other variables that are NA when the indicated variable is NA.
`minlev`	the minimum proportion of observations in a cell before that cell is combined with one or more cells. If more than one cell has fewer than minlev*n observations, all such cells are combined into a new cell labeled `"OTHER"`. Otherwise, the lowest frequency cell is combined with the next lowest frequency cell, and the level name is the combination of the two old level levels.
`abbrev`	set to `TRUE` to abbreviate variable names for plotting or printing. Is set to `TRUE` automatically if `legend=TRUE`.
`slim`	2-vector specifying the range of similarity values for scaling the y-axes. By default this is the observed range over all of `s`.
`slimds`	set to `slimds` to `TRUE` to scale diagonals and off-diagonals separately
`add`	set to `TRUE` to add similarities to an existing plot (usually specifying `lty` or `col`)
`lty`	
`col`	

`lwd`	line type, color, or line thickness for `plotMultSim`
`vname`	optional vector of variable names, in order, used in `s`
`h`	relative height for subplot
`w`	relative width for subplot
`u`	relative extra height and width to leave unused inside the subplot. Also used as the space between y-axis tick mark labels and graph border.
`labelx`	set to `FALSE` to suppress drawing of labels in the x direction
`xspace`	amount of space, on a scale of 1:`n` where `n` is the number of variables, to set aside for y-axis labels

Details

`options(contrasts= c("contr.treatment", "contr.poly"))` is issued temporarily by `varclus` to make sure that ordinary dummy variables are generated for `factor` variables. Pass arguments to the `dataframeReduce` function to remove problematic variables (especially if analyzing all variables in a data frame).

Value

for `varclus` or `naclus`, a list of class `varclus` with elements `call` (containing the calling statement), `sim` (similarity matrix), `n` (sample size used if `x` was not a correlation matrix already - `n` is a matrix), `hclust`, the object created by `hclust`, `similarity`, and `method`. For `plot`, returns the object created by `plclust`. `naclus` also returns the two vectors listed under description, and `naplot` returns an invisible vector that is the frequency table of the number of missing variables per observation. `plotMultSim` invisibly returns the limits of similarities used in constructing the y-axes of each subplot. For `similarity="ccbothpos"` the `hclust` object is `NULL`.

`na.pattern` creates an integer vector of frequencies.

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

References

Sarle, WS: The VARCLUS Procedure. SAS/STAT User's Guide, 4th Edition, 1990. Cary NC: SAS Institute, Inc.

Hoeffding W. (1948): A non-parametric test of independence. Ann Math Stat 19:546–57.

See Also

`hclust`, `plclust`, `hoeffd`, `rcorr`, `cor`, `model.matrix`, `locator`, `na.pattern`

Examples

```
set.seed(1)
x1 <- rnorm(200)
x2 <- rnorm(200)
x3 <- x1 + x2 + rnorm(200)
x4 <- x2 + rnorm(200)
x <- cbind(x1,x2,x3,x4)
v <- varclus(x, similarity="spear")  # spearman is the default anyway
v    # invokes print.varclus
print(round(v$sim,2))
plot(v)

# plot(varclus(~ age + sys.bp + dias.bp + country - 1), abbrev=TRUE)
# the -1 causes k dummies to be generated for k countries
# plot(varclus(~ age + factor(disease.code) - 1))
#
#
# use varclus(~., data= fracmiss= maxlevels= minprev=) to analyze all
# "useful" variables - see dataframeReduce for details about arguments

df <- data.frame(a=c(1,2,3),b=c(1,2,3),c=c(1,2,NA),d=c(1,NA,3),
                 e=c(1,NA,3),f=c(NA,NA,NA),g=c(NA,2,3),h=c(NA,NA,3))
par(mfrow=c(2,2))
for(m in if(.R.)c("ward","complete","median") else
                c("compact","connected","average")) {
  plot(naclus(df, method=m))
  title(m)
}
naplot(naclus(df))
n <- naclus(df)
plot(n); naplot(n)
na.pattern(df)      # builtin function

x <- c(1, rep(2,11), rep(3,9))
combine.levels(x)
x <- c(1, 2, rep(3,20))
combine.levels(x)

# plotMultSim example: Plot proportion of observations
# for which two variables are both positive (diagonals
# show the proportion of observations for which the
# one variable is positive).  Chance-correct the
# off-diagonals by subtracting the product of the
# marginal proportions.  On each subplot the x-axis
# shows month (0, 4, 8, 12) and there is a separate
# curve for females and males
d <- data.frame(sex=sample(c('female','male'),1000,TRUE),
                month=sample(c(0,4,8,12),1000,TRUE),
                x1=sample(0:1,1000,TRUE),
                x2=sample(0:1,1000,TRUE),
                x3=sample(0:1,1000,TRUE))
s <- array(NA, c(3,3,4))
```

```
opar <- par(mar=c(0,0,4.1,0))  # waste less space
for(sx in c('female','male')) {
  for(i in 1:4) {
    mon <- (i-1)*4
    s[,,i] <- varclus(~x1 + x2 + x3, sim='ccbothpos', data=d,
                      subset=d$month==mon & d$sex==sx)$sim
    }
  plotMultSim(s, c(0,4,8,12), vname=c('x1','x2','x3'),
              add=sx=='male', slimds=TRUE,
              lty=1+(sx=='male'))
  # slimds=TRUE causes separate  scaling for diagonals and
  # off-diagonals
}
par(opar)
```

---

wtd.stats *Weighted Statistical Estimates*

Description

These functions compute various weighted versions of standard estimators. In most cases the `weights` vector is a vector the same length of `x`, containing frequency counts that in effect expand `x` by these counts. `weights` can also be sampling weights, in which setting `normwt` to `TRUE` will often be appropriate. This results in making `weights` sum to the length of the non-missing elements in `x`. `normwt=TRUE` thus reflects the fact that the true sample size is the length of the `x` vector and not the sum of the original values of `weights` (which would be appropriate had `normwt=FALSE`). When `weights` is all ones, the estimates are all identical to unweighted estimates (unless one of the non-default quantile estimation options is specified to `wtd.quantile`). When missing data have already been deleted for, `x`, `weights`, and (in the case of `wtd.loess.noiter`) `y`, specifying `na.rm=FALSE` will save computation time. Omitting the `weights` argument or specifying `NULL` or a zero-length vector will result in the usual unweighted estimates.

`wtd.mean`, `wtd.var`, and `wtd.quantile` compute weighted means, variances, and quantiles, respectively. `wtd.Ecdf` computes a weighted empirical distribution function. `wtd.table` computes a weighted frequency table (although only one stratification variable is supported at present). `wtd.rank` computes weighted ranks, using mid–ranks for ties. This can be used to obtain Wilcoxon tests and rank correlation coefficients. `wtd.loess.noiter` is a weighted version of `loess.smooth` when no iterations for outlier rejection are desired. This results in especially good smoothing when `y` is binary.

`num.denom.setup` is a utility function that allows one to deal with observations containing numbers of events and numbers of trials, by outputting two observations when the number of events and non-events (trials - events) exceed zero. A vector of subscripts is generated that will do the proper duplications of observations, and a new binary variable `y` is created along with usual cell frequencies (`weights`) for each of the `y=0`, `y=1` cells per observation.

Usage

```
wtd.mean(x, weights=NULL, normwt="ignored", na.rm=TRUE)
wtd.var(x, weights=NULL, normwt=FALSE, na.rm=TRUE)
wtd.quantile(x, weights=NULL, probs=c(0, .25, .5, .75, 1),
             type=c('quantile','(i-1)/(n-1)','i/(n+1)','i/n'),
             normwt=FALSE, na.rm=TRUE)
wtd.Ecdf(x, weights=NULL,
         type=c('i/n','(i-1)/(n-1)','i/(n+1)'),
         normwt=FALSE, na.rm=TRUE)
wtd.table(x, weights=NULL, type=c('list','table'),
          normwt=FALSE, na.rm=TRUE)
wtd.rank(x, weights=NULL, normwt=FALSE, na.rm=TRUE)
wtd.loess.noiter(x, y, weights=rep(1,n), robust=rep(1,n),
                 span=2/3, degree=1, cell=.13333,
                 type=c('all','ordered all','evaluate'),
                 evaluation=100, na.rm=TRUE)
num.denom.setup(num, denom)
```

Arguments

`x`	a numeric vector (may be a character or `category` or `factor` vector for `wtd.table`)
`num`	vector of numerator frequencies
`denom`	vector of denominators (numbers of trials)
`weights`	a numeric vector of weights
`normwt`	specify `normwt=TRUE` to make `weights` sum to `length(x)` after deletion of NAs
`na.rm`	set to `FALSE` to suppress checking for NAs
`probs`	a vector of quantiles to compute. Default is 0 (min), .25, .5, .75, 1 (max).
`type`	For `wtd.quantile`, `type` defaults to `quantile` to use the same interpolated order statistic method as `quantile`. Set `type` to `"(i-1)/(n-1)"`,`"i/(n+1)"`, or `"i/n"` to use the inverse of the empirical distribution function, using, respectively, (wt - 1)/T, wt/(T+1), or wt/T, where wt is the cumulative weight and T is the total weight (usually total sample size). These three values of `type` are the possibilities for `wtd.Ecdf`. For `wtd.table` the default `type` is `"list"`, meaning that the function is to return a list containing two vectors: `x` is the sorted unique values of `x` and `sum.of.weights` is the sum of weights for that `x`. This is the default so that you don't have to convert the `names` attribute of the result that can be obtained with `type="table"` to a numeric variable when `x` was originally numeric. `type="table"` for `wtd.table` results in an object that is the same structure as those returned from `table`. For `wtd.loess.noiter` the default `type` is `"all"`, indicating that the function is to return a list containing all the original values of `x` (including duplicates and without sorting) and the smoothed `y` values corresponding to them. Set `type="ordered all"` to sort by `x`, and `type="evaluate"` to evaluate the smooth only at `evaluation` equally spaced points between the observed limits of `x`.

`y`	a numeric vector the same length as `x`
`robust, span, degree, cell, evaluation`	see `loess.smooth`. The default is linear (`degree`=1) and 100 points to evaluation (if `type="evaluate"`).

Details

The functions correctly combine weights of observations having duplicate values of `x` before computing estimates.

`wtd.rank` does not handle NAs as elegantly as `rank` if `weights` is specified.

Value

`wtd.mean` and `wtd.var` return scalars. `wtd.quantile` returns a vector the same length as `probs`. `wtd.Ecdf` returns a list whose elements `x` and `Ecdf` correspond to unique sorted values of `x`. If the first CDF estimate is greater than zero, a point (min(x),0) is placed at the beginning of the estimates. See above for `wtd.table`. `wtd.rank` returns a vector the same length as `x` (after removal of NAs, depending on `na.rm`). See above for `wtd.loess.noiter`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

References

Research Triangle Institute (1995): SUDAAN User's Manual, Release 6.40, pp. 8–16 to 8–17.

See Also

`mean`, `var`, `quantile`, `table`, `rank`, `loess.smooth`, `lowess`, `plsmo`, `Ecdf`, `somers2`, `describe`

Examples

```
set.seed(1)
x <- runif(500)
wts <- sample(1:6, 500, TRUE)
std.dev <- sqrt(wtd.var(x, wts))
wtd.quantile(x, wts)
death <- sample(0:1, 500, TRUE)
plot(wtd.loess.noiter(x, death, wts, type='evaluate'))
describe(~x, weights=wts)
# describe uses wtd.mean, wtd.quantile, wtd.table
xg <- cut2(x,g=4)
table(xg)
wtd.table(xg, wts, type='table')
```

```
# Here is a method for getting stratified weighted means
y <- runif(500)
g <- function(y) wtd.mean(y[,1],y[,2])
summarize(cbind(y, wts), llist(xg), g, stat.name='y')

# Restructure data to generate a dichotomous response variable
# from records containing numbers of events and numbers of trials
num   <- c(10,NA,20,0,15)   # data are 10/12 NA/999 20/20 0/25 15/35
denom <- c(12,999,20,25,35)
w     <- num.denom.setup(num, denom)
w
# attach(my.data.frame[w$subs,])
```

`xYplot` *xyplot and dotplot with Matrix Variables to Plot Error Bars and Bands*

Description

A utility function `Cbind` returns the first argument as a vector and combines all other arguments into a matrix stored as an attribute called `"other"`. The arguments can be named (e.g., `Cbind(pressure=y,ylow,yhigh)`) or a `label` attribute may be pre-attached to the first argument. In either case, the name or label of the first argument is stored as an attribute `"label"` of the object returned by `Cbind`. Storing other vectors as a matrix attribute facilitates plotting error bars, etc., as `trellis` really wants the x- and y-variables to be vectors, not matrices. If a single argument is given to `Cbind` and that argument is a matrix with column dimnames, the first column is taken as the main vector and remaining columns are taken as `"other"`. A subscript method for `Cbind` objects subscripts the `other` matrix along with the main `y` vector.

The `xYplot` function is a substitute for `xyplot` that allows for simulated multi-column `y`. It uses by default the `panel.xYplot` and `prepanel.xYplot` functions to do the actual work. The `method` argument passed to `panel.xYplot` from `xYplot` allows you to make error bars, the upper-only or lower-only portions of error bars, alternating lower-only and upper-only bars, bands, or filled bands. `panel.xYplot` decides how to alternate upper and lower bars according to whether the median `y` value of the current main data line is above the median `y` for all `groups` of lines or not. If the median is above the overall median, only the upper bar is drawn. For `bands` (but not 'filled bands'), any number of other columns of `y` will be drawn as lines having the same thickness, color, and type as the main data line. If plotting bars, bands, or filled bands and only one additional column is specified for the response variable, that column is taken as the half width of a precision interval for `y`, and the lower and upper values are computed automatically as `y` plus or minus the value of the additional column variable.

When a `groups` variable is present, `panel.xYplot` will create a function in frame 0 (`.GlobalEnv` in R) called `Key` that when invoked will draw a key describing the `groups` labels, point symbols, and colors. By default, the key is outside the graph. For S-Plus, if `Key(locator(1))` is specified, the key will appear so that its upper left corner is at the coordinates of the mouse click. For R/Lattice the first two arguments of `Key` (`x` and `y`) are fractions of the page, measured from the lower left corner, and the default placement is at `x=0.05, y=0.95`. For R, an optional argument to `sKey`, `other`, may contain a list of arguments to pass to `draw.key` (see `xyplot` for a list of possible arguments, under the `key` option).

When method="quantile" is specified, xYplot automatically groups the x variable into intervals containing a target of nx observations each, and within each x group computes three quantiles of y and plots these as three lines. The mean x within each x group is taken as the x-coordinate. This will make a useful empirical display for large datasets in which scatterdiagrams are too busy to see patterns of central tendency and variability. You can also specify a general function of a data vector that returns a matrix of statistics for the method argument. Arguments can be passed to that function via a list methodArgs. The statistic in the first column should be the measure of central tendency. Examples of useful method functions are those listed under the help file for summary.formula such as smean.cl.normal.

xYplot can also produce bubble plots. This is done when size is specified to xYplot. When size is used, a function sKey is generated for drawing a key to the character sizes. See the bubble plot example. size can also specify a vector where the first character of each observation is used as the plotting symbol, if rangeCex is set to a single cex value. An optional argument to sKey, other, may contain a list of arguments to pass to draw.key (see xyplot for a list of possible arguments, under the key option). See the bubble plot example.

Dotplot is a substitute for dotplot allowing for a matrix x-variable, automatic superpositioning when groups is present, and creation of a Key function. When the x-variable (created by Cbind to simulate a matrix) contains a total of 3 columns, the first column specifies where the dot is positioned, and the last 2 columns specify starting and ending points for intervals. The intervals are shown using line type, width, and color from the trellis plot.line list. By default, you will usually see a darker line segment for the low and high values, with the dotted reference line elsewhere. A good choice of the pch argument for such plots is 3 (plus sign) if you want to emphasize the interval more than the point estimate. When the x-variable contains a total of 5 columns, the 2nd and 5th columns are treated as the 2nd and 3rd are treated above, and the 3rd and 4th columns define an inner line segment that will have twice the thickness of the outer segments. In addition, tick marks separate the outer and inner segments. This type of display (an example of which appeared in *The Elements of Graphing Data* by Cleveland) is very suitable for displaying two confidence levels (e.g., 0.9 and 0.99) or the 0.05, 0.25, 0.75, 0.95 sample quantiles, for example. For this display, the central point displays well with a default circle symbol.

setTrellis sets nice defaults for Trellis graphics, assuming that the graphics device has already been opened if using postscript, etc. By default, it sets panel strips to blank and reference dot lines to thickness 1 instead of the Trellis default of 2.

numericScale is a utility function that facilitates using xYplot to plot variables that are not considered to be numeric but which can readily be converted to numeric using as.numeric(). A good example of this is timeDate variables in S-Plus 5 and 6. numericScale converts the variable into an ordinary numeric variable. If it is a timeDate variable, two attributes are added to the resulting variable: scales.major and scales.minor. These are each lists with elements at to specify a vector of numeric values for tick marks, and a corresponding character vector labels with formatted values (e.g., using time or date formats). When you use such a variable with xYplot and do not specify a corresponding scales element, tick marks and scale labeling are taken from scales.major. The at element for scales.minor is used by panel.xYplot to add minor tick marks. numericScale by default will keep the name of the input variable as a label attribute for the new numeric variable.

Usage

```
Cbind(...)
```

```
xYplot(formula, data = sys.frame(sys.parent()), groups,
       subset, xlab=NULL, ylab=NULL, ylim=NULL,
       panel=panel.xYplot, prepanel=prepanel.xYplot, scales=NULL,
       minor.ticks=NULL, ...)

panel.xYplot(x, y, subscripts, groups=NULL,
             type=if(is.function(method) || method=='quantiles')
               'b' else 'p',
             method=c("bars", "bands", "upper bars", "lower bars",
                      "alt bars", "quantiles", "filled bands"),
             methodArgs=NULL, label.curves=TRUE, abline,
             probs=c(.5,.25,.75), nx=NULL,
             cap=0.015, lty.bar=1,
             lwd=plot.line$lwd, lty=plot.line$lty, pch=plot.symbol$pch,
             cex=plot.symbol$cex, font=plot.symbol$font, col=NULL,
             lwd.bands=NULL, lty.bands=NULL, col.bands=NULL,
             minor.ticks=NULL, col.fill=NULL,
             size=NULL, rangeCex=c(.5,3), ...)

prepanel.xYplot(x, y, ...)

Dotplot(formula, data = sys.frame(sys.parent()), groups, subset,
        xlab = NULL, ylab = NULL, ylim = NULL,
        panel=panel.Dotplot, prepanel=prepanel.Dotplot,
        scales=NULL, xscale=NULL, ...)

prepanel.Dotplot(x, y, ...)

panel.Dotplot(x, y, groups = NULL,
              pch  = dot.symbol$pch,
              col  = dot.symbol$col, cex = dot.symbol$cex,
              font = dot.symbol$font, abline, ...)

setTrellis(strip.blank=TRUE, lty.dot.line=2, lwd.dot.line=1)

numericScale(x, label=NULL, skip.weekends=FALSE, ...)
```

Arguments

`...`	for `Cbind ...` is any number of additional numeric vectors. Unless you are using `Dotplot` (which allows for either 2 or 4 "other" variables) or `xYplot` with `method="bands"`, vectors after the first two are ignored. If drawing bars and only one extra variable is given in `...`, upper and lower values are computed as described above. If the second argument to `Cbind` is a matrix, that matrix is stored in the `"other"` attribute and arguments after the second are ignored. For bubble plots, name an argument `cex`. Also can be other arguments to pass to `labcurve`. or extra arguments sent from `numericScale` to `axis.time`

`formula`	a `trellis` formula consistent with `xyplot` or `dotplot`
`x`	`x`-axis variable. For `numericScale` `x` is any vector such as `as.numeric(x)` returns a numeric vector suitable for x- or y-coordinates.
`y`	a vector, or an object created by `Cbind` for `xYplot`. `y` represents the main variable to plot, i.e., the variable used to draw the main lines. For `Dotplot` the first argument to `Cbind` will be the main `x`-axis variable.
`data`	
`subset`	
`ylim`	
`subscripts`	
`groups`	
`type`	
`scales`	
`panel`	
`prepanel`	
`xlab`	
`ylab`	see `trellis.args`. `xlab` and `ylab` get default values from `"label"` attributes.
`xscale`	allows one to use the default `scales` but specify only the `x` component of it for `Dotplot`
`method`	defaults to `"bars"` to draw error-bar type plots. See meaning of other values above. `method` can be a function. Specifying `method=quantile`, `methodArgs=list(probs=c(.5,.25,.75))` is the same as specifying `method="quantile"` without specifying `probs`.
`methodArgs`	a list containing optional arguments to be passed to the function specified in `method`
`label.curves`	set to `FALSE` to suppress invocation of `labcurve` to label primary curves where they are most separated or to draw a legend in an empty spot on the panel. You can also set `label.curves` to a list of options to pass to `labcurve`. These options can also be passed as `...` to `xYplot`. See the examples below.
`abline`	a list of arguments to pass to `panel.abline` for each panel, e.g. `list(a=0, b=1, col=3)` to draw the line of identity using color 3.
`probs`	a vector of three quantiles with the quantile corresponding to the central line listed first. By default `probs=c(.5, .25, .75)`. You can also specify `probs` through `methodArgs=list(probs=...)`.
`nx`	number of target observations for each `x` group (see `cut2` `m` argument). `nx` defaults to the minimum of 40 and the number of points in the current stratum divided by 4. Set `nx=FALSE` or `nx=0` if `x` is already discrete and requires no grouping.
`cap`	the half-width of horizontal end pieces for error bars, as a fraction of the length of the `x`-axis
`lty.bar`	line type for bars

`lwd, lty, pch, cex, font, col`
: see `trellis.args`. These are vectors when `groups` is present, and the order of their elements corresponds to the different `groups`, regardless of how many bands or bars are drawn. If you don't specify `lty.bands`, for example, all band lines within each group will have the same `lty`.

`lty.bands, lwd.bands, col.bands`
: used to allow `lty`, `lwd`, `col` to vary across the different band lines for different `groups`. These parameters are vectors or lists whose elements correspond to the added band lines (i.e., they ignore the central line, whose line characteristics are defined by `lty`, `lwd`, `col`). For example, suppose that 4 lines are drawn in addition to the central line. Specifying `lwd.bands=1:4` will cause line widths of 1:4 to be used for every group, regardless of the value of `lwd`. To vary characteristics over the `groups` use e.g. `lwd.bands=list(rep(1,4), rep(2,4))` or `list(c(1,2,1,2), c(3,4,3,4))`.

`minor.ticks`
: a list with elements `at` and `labels` specifying positions and labels for minor tick marks to be used on the x-axis of each panel, if any. This is intended for `timeDate` variables.

`col.fill`
: used to override default colors used for the bands in method='filled bands'. This is a vector when `groups` is present, and the order of the elements corresponds to the different `groups`, regardless of how many bands are drawn. The default colors for 'filled bands' are pastel colors matching the default colors superpose.line$col(plot.line$col)

`size`
: a vector the same length as `x` giving a variable whose values are a linear function of the size of the symbol drawn. This is used for example for bubble plots.

`rangeCex`
: a vector of two values specifying the range in character sizes to use for the `size` variable (lowest first, highest second). `size` values are linearly translated to this range, based on the observed range of `size` when `x` and `y` coordinates are not missing. Specify a single numeric `cex` value for `rangeCex` to use the first character of each observations's `size` as the plotting symbol.

`strip.blank`
: set to `FALSE` to not make the panel strip backgrounds blank

`lty.dot.line`
: line type for dot plot reference lines (default = 1 for dotted; use 2 for dotted)

`lwd.dot.line`
: line thickness for reference lines for dot plots (default = 1)

`label`
: a scalar character string to be used as a variable label after `numericScale` converts the variable to numeric form

`skip.weekends`
: see `axis.time`

Details

Unlike `xyplot`, `xYplot` senses the presence of a `groups` variable and automatically invokes `panel.superpose` instead of `panel.xyplot`. The same is true for `Dotplot` vs. `dotplot`.

Value

`Cbind` returns a matrix with attributes. Other functions return standard `trellis` results.

Side Effects

plots, and `panel.xYplot` may create temporary `Key` and `sKey` functions in the session frame.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu
Madeline Bauer
Department of Infectious Diseases
University of Southern California School of Medicine
mbauer@usc.edu

See Also

`xyplot`, `panel.xyplot`, `summarize`, `label`, `labcurve`, `errbar`, `dotplot`, `reShape`, `setps`, `cut2`, `panel.abline`

Examples

```
# Plot 6 smooth functions.  Superpose 3, panel 2.
# Label curves with p=1,2,3 where most separated
d <- expand.grid(x=seq(0,2*pi,length=150), p=1:3, shift=c(0,pi))
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d, type='l')
# Use a key instead, use 3 line widths instead of 3 colors
# Put key in most empty portion of each panel
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d,
       type='l', keys='lines', lwd=1:3, col=1)
# Instead of implicitly using labcurve(), put a
# single key outside of panels (for S-Plus) or at
# lower left corner (for R)
xYplot(sin(x+shift)^p ~ x | shift, groups=p, data=d,
       type='l', label.curves=FALSE, lwd=1:3, col=1, lty=1:3)
Key()

# Bubble plots
x <- y <- 1:8
x[2] <- NA
units(x) <- 'cm^2'
z <- 101:108
p <- factor(rep(c('a','b'),4))
g <- c(rep(1,7),2)
data.frame(p, x, y, z, g)
xYplot(y ~ x | p, groups=g, size=z)
 Key(other=list(title='g', cex.title=1.2))  # draw key for colors
sKey(.2,.85,other=list(title='Z Values', cex.title=1.2))
# draw key for character sizes

# Show the median and quartiles of height given age, stratified
# by sex and race.  Draws 2 sets (male, female) of 3 lines per panel.
# xYplot(height ~ age | race, groups=sex, method='quantiles')

# Examples of plotting raw data
dfr <- expand.grid(month=1:12, continent=c('Europe','USA'),
                   sex=c('female','male'))
set.seed(1)
dfr <- upData(dfr,
              y=month/10 + 1*(sex=='female') + 2*(continent=='Europe') +
                runif(48,-.15,.15),
              lower=y - runif(48,.05,.15),
              upper=y + runif(48,.05,.15))

xYplot(Cbind(y,lower,upper) ~ month,subset=sex=='male' & continent=='USA',
       data=dfr)
xYplot(Cbind(y,lower,upper) ~ month|continent, subset=sex=='male',data=dfr)
xYplot(Cbind(y,lower,upper) ~ month|continent, groups=sex, data=dfr); Key()
# add ,label.curves=FALSE to suppress use of labcurve to label curves where
# farthest apart

xYplot(Cbind(y,lower,upper) ~ month,groups=sex,
                              subset=continent=='Europe', data=dfr)
xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b',
                              subset=continent=='Europe', keys='lines',
                              data=dfr)
# keys='lines' causes labcurve to draw a legend where the panel is most empty

xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b', data=dfr,
                              subset=continent=='Europe',method='bands')
xYplot(Cbind(y,lower,upper) ~ month,groups=sex, type='b', data=dfr,
                              subset=continent=='Europe',method='upper')

label(dfr$y) <- 'Quality of Life Score'
# label is in Hmisc library = attr(y,'label') <- 'Quality...'; will be
# y-axis label
# can also specify Cbind('Quality of Life Score'=y,lower,upper)
xYplot(Cbind(y,lower,upper) ~ month, groups=sex,
       subset=continent=='Europe', method='alt bars',
        offset=if(.R.)unit(.1,'inches') else .4, type='b', data=dfr)
# offset passed to labcurve to label .4 y units away from curve
# for R (using grid/lattice), offset is specified using the grid
# unit function, e.g., offset=unit(.4,'native') or
# offset=unit(.1,'inches') or unit(.05,'npc')

# The following example uses the summarize function in Hmisc to
# compute the median and outer quartiles.  The outer quartiles are
# displayed using "error bars"
set.seed(111)
dfr <- expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
month <- dfr$month; year <- dfr$year
y <- abs(month-6.5) + 2*runif(length(month)) + year-1997
s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)
xYplot(Cbind(y,Lower,Upper) ~ month, groups=year, data=s,
       keys='lines', method='alt', type='b')
```

```
# Can also do:
s <- summarize(y, llist(month,year), quantile, probs=c(.5,.25,.75),
               stat.name=c('y','Q1','Q3'))
xYplot(Cbind(y, Q1, Q3) ~ month, groups=year, data=s,
       type='b', keys='lines')
# Or:
xYplot(y ~ month, groups=year, keys='lines', nx=FALSE, method='quantile',
       type='b')
# nx=FALSE means to treat month as a discrete variable

# To display means and bootstrapped nonparametric confidence intervals
# use:
s <- summarize(y, llist(month,year), smean.cl.boot)
s
xYplot(Cbind(y, Lower, Upper) ~ month | year, data=s, type='b')
# Can also use Y <- cbind(y, Lower, Upper); xYplot(Cbind(Y) ~ ...)
# Or:
xYplot(y ~ month | year, nx=FALSE, method=smean.cl.boot, type='b')

# This example uses the summarize function in Hmisc to
# compute the median and outer quartiles.  The outer quartiles are
# displayed using "filled bands"

s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)

# filled bands: default fill = pastel colors matching solid colors
# in superpose.line (this works differently in R)
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
     method="filled bands" , data=s, type="l")

# note colors based on levels of selected subgroups, not first two colors
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
     method="filled bands" , data=s, type="l",
     subset=(year == 1998 | year == 2000), label.curves=FALSE )

# filled bands using black lines with selected solid colors for fill
xYplot ( Cbind ( y, Lower, Upper ) ~ month, groups=year,
     method="filled bands" , data=s, label.curves=FALSE,
     type="l", col=1, col.fill = 2:3)
Key(.5,.8,col = 2:3) #use fill colors in key

# A good way to check for stable variance of residuals from ols
# xYplot(resid(fit) ~ fitted(fit), method=smean.sdl)
# smean.sdl is defined with summary.formula in Hmisc

# Plot y vs. a timeDate variable x
# xYplot(y ~ numericScale(x, label='Label for X') | country)
# For this example could omit label= and specify
#    y ~ numericScale(x) | country, xlab='Label for X'

# Here is an example of using xYplot with several options
# to change various Trellis parameters,
# xYplot(y ~ x | z, groups=v, pch=c('1','2','3'),
#        layout=c(3,1),     # 3 panels side by side
#        ylab='Y Label', xlab='X Label',
#        main=list('Main Title', cex=1.5),
#        par.strip.text=list(cex=1.2),
#        strip=function(...) strip.default(..., style=1),
#        scales=list(alternating=FALSE))

#
# Dotplot examples
#

s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)

setTrellis()            # blank conditioning panel backgrounds
Dotplot(month ~ Cbind(y, Lower, Upper) | year, data=s)
# or Cbind(...), groups=year, data=s

# Display a 5-number (5-quantile) summary (2 intervals, dot=median)
# Note that summarize produces a matrix for y, and Cbind(y) trusts the
# first column to be the point estimate (here the median)
s <- summarize(y, llist(month,year), quantile,
               probs=c(.5,.05,.25,.75,.95), type='matrix')
Dotplot(month ~ Cbind(y) | year, data=s)
# Use factor(year) to make actual years appear in conditioning title strips

# Plot proportions and their Wilson confidence limits
set.seed(3)
d <- expand.grid(continent=c('USA','Europe'), year=1999:2001,
                 reps=1:100)
# Generate binary events from a population probability of 0.2
# of the event, same for all years and continents
d$y <- ifelse(runif(6*100) <= .2, 1, 0)
s <- with(d,
          summarize(y, llist(continent,year),
                    function(y) {
                     n <- sum(!is.na(y))
                     s <- sum(y, na.rm=TRUE)
                     binconf(s, n)
                    }, type='matrix')
)

Dotplot(year ~ Cbind(y) | continent,  data=s, ylab='Year',
        xlab='Probability')

# Dotplot(z ~ x | g1*g2)
# 2-way conditioning
# Dotplot(z ~ x | g1, groups=g2); Key()
# Key defines symbols for g2

# If the data are organized so that the mean, lower, and upper
# confidence limits are in separate records, the Hmisc reShape
# function is useful for assembling these 3 values as 3 variables
# a single observation, e.g., assuming type has values such as
# c('Mean','Lower','Upper'):
# a <- reShape(y, id=month, colvar=type)
# This will make a matrix with 3 columns named Mean Lower Upper
# and with 1/3 as many rows as the original data
```

`xy.group` *Mean x vs. function of y in groups of x*

Description

Compute mean x vs. a function of y (e.g. median) by quantile groups of x or by x grouped to have a given average number of observations. Deletes NAs in x and y before doing computations.

Usage

```
xy.group(x, y, m=150, g, fun=mean, result="list")
```

Arguments

`x`	a vector, may contain NAs
`y`	a vector of same length as x, may contain NAs
`m`	number of observations per group
`g`	number of quantile groups
`fun`	function of y such as median or mean (the default)
`result`	"list" (the default), or "matrix"

Value

if result="list", a list with components x and y suitable for plotting. if result="matrix", matrix with rows corresponding to x-groups and columns named n, x, and y.

See Also

`cut2`, `tapply`

Examples

```
# plot(xy.group(x, y, g=10))    #Plot mean y by deciles of x
# xy.group(x, y, m=100, result="matrix")        #Print table, 100 obs/group
```

`yearDays` *Get Number of Days in Year or Month*

Description

Returns the number of days in a specific year or month.

Usage

```
yearDays(time)

monthDays(time)
```

Arguments

`time`	A POSIXt or Date object describing the month or year in question.

Author(s)

Charles Dupont

See Also

`POSIXt`, `Date`

`sas.get` *Convert a SAS Dataset to an S Data Frame*

Description

Converts a SAS dataset into an S data frame. You may choose to extract only a subset of variables or a subset of observations in the SAS dataset. You may have the function automatically convert PROC FORMAT-coded variables to factor objects. The original SAS codes are stored in an attribute called `sas.codes` and these may be added back to the `levels` of a `factor` variable using the `code.levels` function. Information about special missing values may be captured in an attribute of each variable having special missing values. This attribute is called `special.miss`, and such variables are given class `special.miss`. There are `print`, `[]`, `format`, and `is.special.miss` methods for such variables. The `chron` function is used to set up date, time, and date-time variables. If using S-Plus 5 or 6 or later, the `timeDate` function is used instead. Under R, `Dates` is used for dates and `chron` for date-times. For times without dates, these still need to be stored in date-time format in POSIX. Such SAS time variables are given a major class of `timePOSIXt` and a `format.timePOSIXt` function so that the date portion (which will always be 1/1/1970) will not print by default. If a date variable represents a partial date (.5 added if month missing, .25 added if day missing, .75 if both), an attribute `partial.date` is added to the variable, and the variable also becomes a class `imputed` variable. The `describe` function uses information about partial dates and special missing values. There is an option to automatically uncompress (or gunzip) compressed SAS datasets.

Usage

```
sas.get(library, member, variables=character(0), ifs=character(0),
     format.library=library, id,
     dates.=c("sas","yymmdd","yearfrac","yearfrac2"),
     keep.log=TRUE, log.file="_temp_.log", macro=sas.get.macro,
     data.frame.out=existsFunction("data.frame"), clean.up=!.R., quiet=FALSE,
     temp=tempfile("SaS"), formats=TRUE, recode=formats,
     special.miss=FALSE, sasprog="sas",
     as.is=.5, check.unique.id=TRUE, force.single=FALSE,
     where, uncompress=FALSE)

is.special.miss(x, code)

## S3 method for class 'special.miss':
x[..., drop=FALSE]

## S3 method for class 'special.miss':
print(x, ...)

## S3 method for class 'special.miss':
format(x, ...)

sas.codes(object)

code.levels(object)
```

Arguments

`library`	character string naming the directory in which the dataset is kept.
`member`	character string giving the second part of the two part SAS dataset name. (The first part is irrelevant here - it is mapped to the UNIX directory name.)
`x`	a variable that may have been created by `sas.get` with `special.miss=T` or with `recode` in effect.
`variables`	vector of character strings naming the variables in the SAS dataset. The S dataset will contain only those variables from the SAS dataset. To get all of the variables (the default), an empty string may be given. It is a fatal error if any one of the variables is not in the SAS dataset. You can use `sas.contents` to get the variables in the SAS dataset. If you have retrieved a subset of the variables in the SAS dataset and which to retrieve the same list of variables from another dataset, you can program the value of `variables` - see one of the last examples.
`ifs`	a vector of character strings, each containing one SAS "subsetting if" statement. These will be used to extract a subset of the observations in the SAS dataset.
`format.library`	The UNIX directory containing the file **formats.sct**, which contains the definitions of the user defined formats used in this dataset. By default, we look for the formats in the same directory as the data. The user defined formats must be available (so SAS can read the data).

`formats`	Set `formats` to `F` to keep `sas.get` from telling the SAS macro to retrieve value label formats from `format.library`. When you do not specify `formats` or `recode`, `sas.get` will set `format` to `T` if a SAS format catalog (`.sct` or `.sc2`) file exists in `format.library`. Value label formats if present are stored as the `formats` attribute of the returned object (see below). A format is used if it is referred to by one or more variables in the dataset, if it contains no ranges of values (i.e., it identifies value labels for single values), and if it is a character format or a numeric format that is not used just to label missing values. If you set `recode` to `TRUE`, 1, or 2, `formats` defaults to `TRUE`. To fetch the values and labels for variable `x` in the dataset `d` you could type:
	`f <- attr(d$x, "format")`
	`formats <- attr(d, "formats")`
	`formats$f$values; formats$f$labels`

`recode`	This parameter defaults to `TRUE` if `formats` is `TRUE`. If it is `TRUE`, variables that have an appropriate format (see above) are recoded as `factor` objects, which map the values to the value labels for the format. Alternatively, set `recode` to 1 to use labels of the form value:label, e.g. 1:good 2:better 3:best. Set `recode` to 2 to use labels such as good(1) better(2) best(3). Since `sas.codes` and `code.levels` add flexibility, the usual choice for `recode` is `T` or `TRUE`.

`special.miss`	For numeric variables, any missing values are stored as NA in S. You can recover special missing values by setting `special.miss` to `TRUE`. This will cause the `special.miss` attribute and the `special.miss` class to be added to each variable that has at least one special missing value. Suppose that variable `y` was .E in observation 3 and .G in observation 544. The `special.miss` attribute for `y` then has the value
	`list(codes=c("E","G"),obs=c(3,544))`
	To fetch this information for variable `y` you would say for example
	`s <- attr(y, "special.miss")`
	`s$codes; s$obs`
	or use `is.special.miss(x)` or the `print.special.miss` method, which will replace `NA` values for the variable with `E` or `G` if they correspond to special missing values. The describe function uses this information in printing a data summary.

`drop`	set `drop=FALSE` to keep unused factor levels as columns of the matrix produced by `mChoice`
`id`	The name of the variable to be used as the row names of the S dataset. The id variable becomes the `row.names` attribute of a data frame, but the id variable is still retained as a variable in the data frame. (if `data.frame.out` is `FALSE`, this will be the attribute `"id"` of the S dataset.) You can also specify a vector of variable names as the `id` parameter. After fetching the data from SAS, all these variables will be converted to character format and concatenated (with a space as a separator) to form a (hopefully) unique ID variable.
`dates.`	specifies the format for storing SAS dates in the resulting data frame
`as.is`	IF `data.frame.out=T`, SAS character variables are converted to S factor objects if `as.is=F` or if `as.is` is a number between 0 and 1 inclusive and the number of unique values of the variable is less than the number of observations

(`n`) times `as.is`. The default if `as.is` is .5, so character variables are converted to factors only if they have fewer than `n/2` unique values. The primary purpose of this is to keep unique identification variables as character values in the data frame instead of using more space to store both the integer factor codes and the factor labels.

`check.unique.id`	If `id` is specified, the row names are checked for uniqueness if `check.unique.id=T`. If any are duplicated, a warning is printed. Note that if a data frame is being created with duplicate row names, statements such as `my.data.frame["B23",]` will retrieve only the first row with a row name of `"B23"`.
`force.single`	By default, SAS numeric variables having `LENGTH`s > 4 are stored as S double precision numerics, which allow for the same precision as a SAS `LENGTH` 8 variable. Set `force.single=T` to store every numeric variable in single precision (7 digits of precision). This option is useful when the creator of the SAS dataset has failed to use a `LENGTH` statement. R does not have single precision, so no attempt is made to convert to single if running R.
`dates`	One of the character strings `"sas"`, `"yearfrac"`, `"yearfrac2"`, `"yymmdd"`. If a SAS variable has a date format (one of "DATE", "MMDDYY", "YYMMDD", "DDMMYY", "YYQ", "MONYY", "JULIAN"), it will be converted to the format specified by `dates` before being given to S. `"sas"` gives days from 1/1/1960 (from 1/1/1970 if using `chron`), `"yearfrac"` gives days from 1/1/1900 divided by 365.25, `"yearfrac2"` gives year plus fraction of current year, and `"yymmdd"` gives a 6 digit number YYMMDD (year%%100, month, day). Note that S will store these as numbers, not as character strings. If dates="sas" and a variable has one of the SAS date formats listed above, the variable will be given a class of "date" to work with Terry Therneau's implementation of the "date" class in S. If the `chron` package or `timeDate` function is available, these are used instead.
`keep.log`	logical flag: if `FALSE`, delete the SAS log file upon completion.
`log.file`	the name of the SAS log file.
`macro`	the name of an S object in the current search path that contains the text of the SAS macro called by S. The S object is a character vector that can be edited using for example sas.get.macro <- editor(sas.get.macro).
`data.frame.out`	logical flag: if `TRUE`, the return value will be an S data frame, otherwise it will be a list.
`clean.up`	logical flag: if `TRUE`, remove all temporary files when finished. You may want to keep these while debugging the SAS macro. Not needed for R.
`quiet`	logical flag: if `FALSE`, print the contents of the SAS log file if there has been an error.
`temp`	the prefix to use for the temporary files. Two characters will be added to this, the resulting name must fit on your file system.
`sasprog`	the name of the system command to invoke SAS
`uncompress`	set to `T` to automatically invoke the UNIX `gunzip` command (if `member.ssd01.gz` exists) or the `uncompress` command (if `member.ssd01.Z` exists) to un-

	compress the SAS dataset before proceeding. This assumes you have the file permissions to allow uncompressing in place. If the file is already uncompressed, this option is ignored.
`where`	by default, a list or data frame which contains all the variables is returned. If you specify `where`, each individual variable is placed into a separate object (whose name is the name of the variable) using the `assign` function with the `where` argument. For example, you can put each variable in its own file in a directory, which in some cases may save memory over attaching a data frame.
`code`	a special missing value code (A through Z or underscore) to check against. If `code` is omitted, `is.special.miss` will return a `T` for each observation that has any special missing value.
`object`	a variable in a data frame created by `sas.get`
`...`	ignored

Details

If you specify `special.miss=T` and there are no special missing values in the data SAS dataset, the SAS step will bomb.

For variables having a `PROC FORMAT VALUE` format with some of the levels undefined, `sas.get` will interpret those values as `NA` if you are using `recode`.

The SAS macro `sas_get` uses record lengths of up to 4096 in two places. If you are exporting records that are very long (because of a large number of variables and/or long character variables), you may want to edit these `LRECL`s to quadruple them, for example.

Value

if `data.frame.out` is `TRUE`, the output will be a data frame resembling the SAS dataset. If `id` was specified, that column of the data frame will be used as the row names of the data frame. Each variable in the data frame or vector in the list will have the attributes `label` and `format` containing SAS labels and formats. Underscores in formats are converted to periods. Formats for character variables have `$` placed in front of their names. If `formats` is `TRUE` and there are any appropriate format definitions in `format.library`, the returned object will have attribute `formats` containing lists named the same as the format names (with periods substituted for underscores and character formats prefixed by `$`). Each of these lists has a vector called `values` and one called `labels` with the `PROC FORMAT; VALUE ...` definitions.

If `data.frame.out` is `FALSE`, the output will be a list of vectors, each containing a variable from the SAS dataset. If `id` was specified, that element of the list will be used as the `id` attribute of the entire list.

Side Effects

if a SAS error occurs and `quiet` is `FALSE`, then the SAS log file will be printed under the control of the **less** pager.

BACKGROUND

The references cited below explain the structure of SAS datasets and how they are stored under UNIX. See *SAS Language* for a discussion of the "subsetting if" statement.

Note

You must be able to run SAS (by typing **sas**) on your system. If the S command `!sas` does not start SAS, then this function cannot work.

If you are reading time or date-time variables, you will need to execute the command `library(chron)` to print those variables or the data frame if the `timeDate` function is not available.

Author(s)

Terry Therneau, Mayo Clinic
Frank Harrell, Vanderbilt University
Bill Dunlap, University of Washington and Insightful Corporation
Michael W. Kattan, Cleveland Clinic Foundation

References

SAS Institute Inc. (1990). *SAS Language: Reference, Version 6.* First Edition. SAS Institute Inc., Cary, North Carolina.

SAS Institute Inc. (1988). SAS Technical Report P-176, *Using the SAS System, Release 6.03, under UNIX Operating Systems and Derivatives.* SAS Institute Inc., Cary, North Carolina.

SAS Institute Inc. (1985). *SAS Introductory Guide.* Third Edition. SAS Institute Inc., Cary, North Carolina.

See Also

`data.frame`, `describe`, `label`, `upData`, `cleanup.import`

Examples

```
## Not run:
sas.contents("saslib", "mice")
# [1] "dose"  "ld50"  "strain"  "lab_no"
attr(, "n"):
# [1] 117
mice <- sas.get("saslib", mem="mice", var=c("dose", "strain", "ld50"))
plot(mice$dose, mice$ld50)

nude.mice <- sas.get(lib=unix("echo $HOME/saslib"), mem="mice",
        ifs="if strain='nude'")

nude.mice.dl <- sas.get(lib=unix("echo $HOME/saslib"), mem="mice",
        var=c("dose", "ld50"), ifs="if strain='nude'")

# Get a dataset from current directory, recode PROC FORMAT; VALUE ...
# variables into factors with labels of the form "good(1)" "better(2)",
# get special missing values, recode missing codes .D and .R into new
# factor levels "Don't know" and "Refused to answer" for variable q1
d <- sas.get(".", "mydata", recode=2, special.miss=TRUE)
attach(d)
nl <- length(levels(q1))
lev <- c(levels(q1), "Don't know", "Refused")
```

```
q1.new <- as.integer(q1)
q1.new[is.special.miss(q1,"D")] <- nl+1
q1.new[is.special.miss(q1,"R")] <- nl+2
q1.new <- factor(q1.new, 1:(nl+2), lev)
# Note: would like to use factor() in place of as.integer ... but
# factor in this case adds "NA" as a category level

d <- sas.get(".", "mydata")
sas.codes(d$x)    # for PROC FORMATted variables returns original data codes
d$x <- code.levels(d$x)   # or attach(d); x <- code.levels(x)
# This makes levels such as "good" "better" "best" into e.g.
# "1:good" "2:better" "3:best", if the original SAS values were 1,2,3

# Retrieve the same variables from another dataset (or an update of
# the original dataset)
mydata2 <- sas.get('mydata2', var=names(d))
# This only works if none of the original SAS variable names contained _
mydata2 <- cleanup.import(mydata2) # will make true integer variables

# Code from Don MacQueen to generate SAS dataset to test import of
# date, time, date-time variables
# data ssd.test;
#     d1='3mar2002'd ;
#     dt1='3mar2002 9:31:02'dt;
#     t1='11:13:45't;
#     output;
#
#     d1='3jun2002'd ;
#     dt1='3jun2002 9:42:07'dt;
#     t1='11:14:13't;
#     output;
#     format d1 mmddyy10. dt1 datetime. t1 time.;
# run;
## End(Not run)
```

Index