Package 'TeachingDemos'

February 17, 2024

Title Demonstrations for Teaching and Learning

Version 2.13

Author Greg Snow

Description Demonstration functions that can be used in a classroom to demonstrate statistical concepts, or on your own to better understand the concepts or the programming.

Maintainer Greg Snow <538280@gmail.com>

License Artistic-2.0

Date 2024-02-13

Suggests tkrplot, lattice, MASS, rgl, tcltk, tcltk2, png, ggplot2, logspline, spData, sf

Enhances manipulate, R2wd

LazyData true

KeepSource true

NeedsCompilation no

Repository CRAN

Date/Publication 2024-02-16 23:10:11 UTC

R topics documented:

`TeachingDemos-package` *Various functions for demonstration and learning.*

Description

This package provides various demonstrations that can be used in classes or by individuals to better learn statistical concepts and usage of R. Various utility functions are also included

Details

Package:	TeachingDemos
Type:	Package
License:	Artistic-2.0

Demonstration functions in this package include:

ci.examp, run.ci.examp	Confidence Interval Examples
clt.examp	Central Limit Theorem Example
dice, plot.dice	Roll and Plot dice (possibly loaded)
faces, faces2	Chernoff face plots
fagan.plot	Fagan plot for screening designs
lattice.demo	The 3d slicing idea behind lattice/trellis graphics
loess.demo	Interactive demo to show ideas of loess smooths
mle.demo	Interactive demo of Maximum Likelihood Estimation
plot.rgl.coin, plot.rgl.die	Animate flipping a coin or rolling a die
power.examp	Demonstrate concepts of Power.
put.points.demo	Add/move points on a plot and see the effect on correlation and regression.
roc.demo	Interactive demo of ROC curves.

rotate.cloud	Interactively rotate 3d plots.
run.cor.examp	Show plots representing different correlations.
run.hist.demo	Interactively change parameters for histograms.
SensSpec.demo	Show relationship between Sensitivity, Specificity, Prevalence and PPV and NPV.
TkApprox	Interactive linear interpolations of data.
tkBrush	Brush points in a scatterplot matrix.
TkSpline	Interactive spline interpolations of data.
tree.demo	Interactively Recursive partition data (create trees).
vis.binom	Plot various probability distributions and interactively change parameters.
vis.boxcox	Interactively change lambda for Box Cox Transforms.
z.test	Z-test similar to t.test for students who have not learned t tests yet.
Pvalue.norm.sim	
Pvalue.binom.sim	Simulate P-values to see how they are distributed.
run.Pvalue.norm.sim	GUI for above.
run.Pvalue.binom.sim	
HWidentify	
HTKidentify	Identify the point Hovered over with the mouse.
vis.test	test a null hypothesis by comparing graphs.

Utility functions include:

bct	Box-Cox Transforms.
char2seed	set or create the random number seed using a character string
clipplot	clip a plot to a rectangular region within the plot
col2grey	convert colors to greyscale
cnvrt.coords	Convert between the different coordinate systems
dynIdentify	Scatterplot with point labels that can be dragged to a new position
TkIdentify	Scatterplot with lables that can be dragged to new positions
gp.plot gp.splot	send commands to gnuplot
hpd	Highest Posterior Density intervals
my.symbols	Create plots using user defined symbols.
panel.my.symbols	Create lattice plots using user defined symbols.
plot2script	Create a script file that recreates the current plot.
shadowtext	plot text with contrasting shadow for better readability.
squishplot	Set the margins so that a plot has a specific aspect ratio without large white space inside.
spread.labs	Spread out coordinates so that labels do not overlap.
subplot	create a plot inside of an existing plot.
tkexamp	create plots that can have parameters adjusted interactively.
triplot	Trilinear plot for 3 proportions.
txtStart/etxtStart/wdtxtStart	Save commands and output to a text file (possibly for post processing with enscript).
zoomplot	recreate the current plot with different x/y limits (zoom in out).

Author(s)

Greg Snow <538280@gmail.com>

See Also

The tkrplot package

Examples

```
ci.examp()

clt.examp()
clt.examp(5)

plot.dice( expand.grid(1:6,1:6), layout=c(6,6) )

faces(rbind(1:3,5:3,3:5,5:7))

plot(1:10, 1:10)
my.symbols( 1:10, 1:10, ms.polygram, n=1:10, inches=0.3 )

x <- seq(1,100)
y <- rnorm(100)
plot(x,y, type='b', col='blue')
clipplot( lines(x,y, type='b', col='red'), ylim=c(par('usr')[3],0))

power.examp()
power.examp(n=25)
power.examp(alpha=0.1)
```

---

bct *Box-Cox Transforms*

Description

Computes the Box-Cox transform of the data for a given value of lambda. Includes the scaling factor.

Usage

```
bct(y, lambda)
```

Arguments

y	Vector of data to be transformed.
lambda	Scalar exponent for transform (1 is linear, 0 is log).

Details

bct computes the Box-Cox family of transforms: y = (y^lambda - 1)/(lambda*gm^(lambda-1)), where gm is the geometric mean of the y's. returns log(y)*gm when lambda equals 0.

Value

A vector of the same length as y with the corresponding transformed values.

Author(s)

Greg Snow <538280@gmail.com>

See Also

vis.boxcox, vis.boxcoxu, boxcox in package MASS, other implementations in various packages

Examples

```
y <- rlnorm(500, 3, 2)
par(mfrow=c(2,2))
qqnorm(y)
qqnorm(bct(y,1/2))
qqnorm(bct(y,0))
hist(bct(y,0))
```

---

cal *Plot a month or year calendar*

Description

Plot a calendar of the specified year or month. Monthly calendars can have additional information (text/plots) added to the individual cells.

Usage

```
cal(month, year)
```

Arguments

month	The month for the calendar, if ommitted will do a yearly calendar, can either be a number from 1 to 12 or a character string that will be matched (using pmatch) against month.name.
year	The year for the calendar. If ommitted and month is an integer less than or equal to 12 then month will be used as the year.

Details

This function plots on the current (or default) graphics device a yearly or monthly calendar. It tries to guess what you want, if both `year` and `month` are ommitted then it will plot the current month. If `month` is an integer greater than 12 and no `year` is specified then that value will be used as the year for a yearly calendar. The `month` can be either an integer from 1 to 12 or a character string that will be matched against `month.name` using `pmatch`.

Each day of the monthly calendar is a plotting frame that can be added to using stardard low level functions, the coordinates of the plotting region (the entire box) are from 0 to 1 in both dimensions. The `updateusr` function can be used to change the coordinates. The return from the function (when creating a monthly calendar) can be used to select the day.

Value

Nothing is returned when a whole year calendar is created. When the month calendar is created a function is returned invisibly that if passed an integer corresponding to a day of the month will set the graphics parameters so the corresponding day in the calendar becomes the current plotting figure. See the examples below.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

`Sys.time`, `as.POSIXlt`, `par`, `updateusr`

Examples

```
cal(2011)

cal('May')

setday <- cal(11, 2011)

setday(3)
text(0.5,0.5, 'Some\nCentered\nText')

setday(8)
text(1,1,'Top Right',adj=c(1,1))

setday(18)
text(0,0,'Bottom Left', adj=c(0,0) )

setday(21)
tmp.x <- runif(25)
tmp.y <- rnorm(25, tmp.x, .1)
mrgn.x <- 0.04*diff(range(tmp.x))
mrgn.y <- 0.04*diff(range(tmp.y))
updateusr( 0:1, 0:1, range(tmp.x)+c(-1,1)*mrgn.x, range(tmp.y)+c(-1,1)*mrgn.y)
points(tmp.x, tmp.y)
```

```
setday(30)
tmp <- hist(rnorm(100), plot=FALSE)
updateusr( 0:1, 0:1, range(tmp$breaks), range(tmp$counts*1.1,0) )
lines(tmp)
```

`char2seed` *Convert a character string into a random seed*

Description

This function creates a seed for the random number generator from a character string. Character strings can be based on student names so that every student has a different random sample, but the teacher can generate the same datasets.

Usage

```
char2seed(x, set = TRUE, ...)
```

Arguments

`x`	A character string
`set`	Logical, should the seed be set or just returned
`...`	Additional parameters passed on to `set.seed`

Details

Simulations or other situations call for the need to have repeatable random numbers, it is easier to remember a word or string than a number, so this function converts words or character strings to an integer and optionally sets the seed based on this.

Teachers can assign students to generate a random dataset using their name to seed the rng, this way each student will have a different dataset, but the teacher can generate the same set of data to check values.

Any characters other than letters (a-zA-Z) or digits (0-9) will be silently removed. This function is not case sensitive, so "ABC" and "abc" will generate the same seed.

This is a many to one function, so it is possible to find different words that generate the same seed, but this is unlikely by chance alone.

Value

This returns an integer (but mode numeric) to use as a seed for the RNG. If `set` is true then it is returned invisibly.

Author(s)

Greg Snow <538280@gmail.com>

See Also

set.seed

Examples

```
char2seed('Snow')
x <- rnorm(100)
rnorm(10)
tmp <- char2seed('Snow',set=FALSE)
set.seed(tmp)
y <- rnorm(100)

all.equal(x,y) # should be true
```

---

chisq.detail *Print details of a chi-squared test*

Description

Prints out the details of the computations involved in a chi-squared test on a table. Includes the expected values and the chi-squared contribution of each cell.

Usage

```
chisq.detail(tab)
```

Arguments

tab	Matrix or table to be analyzed

Details

This function prints out the input table along with the expected value for each cell under the null hypothesis. It also prints out the chi-squared contribution of each cell in the same pattern as the table. This shows the computations involved and one rule of thumb is to look for these values that are greater than 4 as a post-hoc analysis.

Value

This function is used primarily for its side effect of printing the results, but does return invisibly a list with the following components:

obs	A matrix of the observed values, same as tab.
expected	A matrix of the expected values under the null hypothesis.
chi.table	A matrix of the chi-squared contributions of each cell.
chi2	The chi-squared test statistic.

Author(s)

Greg Snow, <538280@gmail.com>

References

~put references to the literature/web site here ~ Moore, bps

See Also

chisq.test,loglin, xtabs, table, prop.table, CrossTable from the gmodels package.

Examples

```
chisq.detail(HairEyeColor[,,1])
chisq.detail(HairEyeColor[,,2])
```

---

ci.examp *Plot examples of Confidence Intervals*

Description

Generate reps samples from a normal distribution then compute and plot confidence intervals for each sample along with information about the population to demonstrate confidence intervals. Optionally change the confidence level using a Tk slider.

Usage

```
ci.examp(mean.sim = 100, sd = 10, n = 25, reps = 50, conf.level = 0.95,
  method = "z", lower.conf = (1 - conf.level)/2,
  upper.conf = 1 - (1 - conf.level)/2)
run.ci.examp(reps = 100, seed, method="z", n=25)
```

Arguments

mean.sim	The mean of the population.
sd	The standard deviation of the population.
n	The sample size for each sample.
reps	The number of samples/intervals to create.
conf.level	The confidence level of the intervals.
method	'z', 't', or 'both', should the intervals be based on the normal, the t, or both distributions.
lower.conf	Quantile for lower confidence bound.
upper.conf	Quantile for upper confidence bound.
seed	The seed to use for the random number generation.

Details

These functions demonstrate the concept of confidence intervals by taking multiple samples from a known normal distribution and calculating a confidence interval for each sample and plotting the interval relative to the true mean. Intervals that contain the true mean will be plotted in black and those that do not include the true mean will be plotted in different colors.

The `method` argument determines the type of interval: 'z' will use the normal distribution and the known population standard deviation, 't' will use the t distribution and the sample standard deviations, 'both' will compute both for each sample for easy comparison (it is best to reduce `reps` to about 25 when using 'both').

The optional arguments `lower.conf` and `upper.conf` can be used to plot non-symmetric or 1 sided confidence intervals.

The function `run.ci.examp` also creates a Tk slider that will allow you to interactively change the confidence level and replot the intervals to show how the interval widths change with the confidence level.

Value

These functions are run solely for the side effect of plotting the intervals, there is no meaningfull return value.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`z.test`, `t.test`

Examples

```
ci.examp()

if(interactive()) {
  run.ci.examp()
}

# 1 sided confidence intervals
ci.examp(lower.conf=0, upper.conf=0.95)

# non-symmetric intervals
ci.examp(lower.conf=0.02, upper.conf=0.97)
```

clipplot *Clip plotting to a rectangular region*

Description

Clip plotting to a rectangular region that is a subset of the plotting area

Usage

```
clipplot(fun, xlim = par("usr")[1:2], ylim = par("usr")[3:4])
```

Arguments

fun	The function or expression to do the plotting.
xlim	A vector of length 2 representing the x-limits to clip plotting to, defaults to the entire width of the plotting region.
ylim	A vector of length 2 representing the y-limits to clip the plot to, defaults to the entire height of the plotting region.

Details

This function resets the active region for plotting to a rectangle within the plotting area and turns on clipping so that any points, lines, etc. that are outside the rectange are not plotted.

A side effect of this function is a call to the box() command, it is called with a fully transparent color so if your graphics device honors transparency then you will probably see no effect.

Value

Nothing meaningful is returned

Note

This function abuses some of the intent of what par(plt=...) is supposed to mean. In R2.7.0 and beyond there is a new funcntion clip with the intended purpose of doing this in a more proper manner (however as of my last test it is not working perfectly either, so clipplot will remain undepricated for now).

It uses some hacks to make sure that the clipping region is set, but it does this by plotting some tranparent boxes, therefore you should not use this on devices where tranparency is not supported (or you may see extra boxes).

Author(s)

Greg Snow <538280@gmail.com>

See Also

par, lines, clip in R2.7.0 and later

Examples

```
x <- seq(1,100)
y <- rnorm(100)
plot(x,y, type='b', col='blue')
clipplot( lines(x,y, type='b', col='red'), ylim=c(par('usr')[3],0))


attach(iris)

tmp <- c('red','green','blue')
names(tmp) <- levels(Species)
plot(Petal.Width,Petal.Length, col=tmp[Species])
for(s in levels(Species)){
  clipplot( abline(
    lm(Petal.Length~Petal.Width, data=iris, subset=Species==s),
    col=tmp[s]),
    xlim=range(Petal.Width[Species==s]))
}

detach(iris)
```

`clt.examp` *Plot Examples of the Central Limit Theorem*

Description

Takes samples of size `n` from 4 different distributions and plots histograms of the means along with a normal curve with matching mean and standard deviation. Creating the plots for different values of `n` demonstrates the Central Limit Theorem.

Usage

```
clt.examp(n = 1, reps = 10000, nclass = 16, norm.param=list(mean=0,sd=1),
          gamma.param=list(shape=1, rate=1/3), unif.param=list(min=0,max=1),
          beta.param=list(shape1=0.35, shape2=0.25))
```

Arguments

`n`	size of the individual samples
`reps`	number of samples to take from each distribution
`nclass`	number of bars in the histograms
`norm.param`	List with parameters passed to `rnorm`
`gamma.param`	List with parameters passed to `rgamma`
`unif.param`	List with parameters passed to `runif`
`beta.param`	List with parameters passed to `rbeta`

Details

The 4 distributions sampled from are a Normal with defaults mean 0 and standard deviation 1, a gamma with defaults shape 1 (exponential) and lambda 1/3 (mean = 3), a uniform distribution from 0 to 1 (default), and a beta distribution with default alpha 0.35 and beta 0.25 (U shaped left skewed).

The `norm.param`, `gamma.param`, `unif.param`, and `beta.param` arguments can be used to change the parameters of the generating distributions.

Running the function with `n=1` will show the populations. Run the function again with `n` at higher values to show that the sampling distribution of the uniform quickly becomes normal and the exponential and beta distributions eventually become normal (but much slower than the uniform).

Value

This function is run for its side effect of creating plots. It returns NULL invisibly.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`rnorm`, `rexp`, `runif`, `rbeta`

Examples

```
clt.examp()
clt.examp(5)
clt.examp(30)
clt.examp(50)
```

---

cnvrt.coords *Convert between the 5 different coordinate sytems on a graphical device*

Description

Takes a set of coordinates in any of the 5 coordinate systems (usr, plt, fig, dev, or tdev) and returns the same points in all 5 coordinate systems.

Usage

```
cnvrt.coords(x, y = NULL, input = c("usr", "plt", "fig", "dev","tdev"))
```

Arguments

`x`	Vector, Matrix, or list of x coordinates (or x and y coordinates), NA's allowed.
`y`	y coordinates (if `x` is a vector), NA's allowed.
`input`	Character scalar indicating the coordinate system of the input points.

Details

Every plot has 5 coordinate systems:

usr (User): the coordinate system of the data, this is shown by the tick marks and axis labels.

plt (Plot): Plot area, coordinates range from 0 to 1 with 0 corresponding to the x and y axes and 1 corresponding to the top and right of the plot area. Margins of the plot correspond to plot coordinates less than 0 or greater than 1.

fig (Figure): Figure area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left edges of the figure (including margins, label areas) and 1 corresponds to the top and right edges. fig and dev coordinates will be identical if there is only 1 figure area on the device (layout, mfrow, or mfcol has not been used).

dev (Device): Device area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left of the device region within the outer margins and 1 is the top and right of the region withing the outer margins. If the outer margins are all set to 0 then tdev and dev should be identical.

tdev (Total Device): Total Device area, coordinates range from 0 to 1 with 0 corresponding to the bottom and left edges of the device (piece of paper, window on screen) and 1 corresponds to the top and right edges.

Value

A list with 5 components, each component is a list with vectors named x and y. The 5 sublists are:

`usr`	The coordinates of the input points in usr (User) coordinates.
`plt`	The coordinates of the input points in plt (Plot) coordinates.
`fig`	The coordinates of the input points in fig (Figure) coordinates.
`dev`	The coordinates of the input points in dev (Device) coordinates.
`tdev`	The coordinates of the input points in tdev (Total Device) coordinates.

Note

You must provide both x and y, but one of them may be `NA`.

This function is now depricated with the new functions `grconvertX` and `grconvertY` in R version 2.7.0 and beyond. These new functions use the correct coordinate system names and have more coordinate systems available, you should start using them instead.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`par` specifically 'usr','plt', and 'fig'. Also 'xpd' for plotting outside of the plotting region and 'mfrow' and 'mfcol' for multi figure plotting. `subplot`, `grconvertX` and `grconvertY` in R2.7.0 and later

Examples

```
old.par <- par(no.readonly=TRUE)

par(mfrow=c(2,2),xpd=NA)

# generate some sample data
tmp.x <- rnorm(25, 10, 2)
tmp.y <- rnorm(25, 50, 10)
tmp.z <- rnorm(25, 0, 1)

plot( tmp.x, tmp.y)

# draw a diagonal line across the plot area
tmp1 <- cnvrt.coords( c(0,1), c(0,1), input='plt' )
lines(tmp1$usr, col='blue')

# draw a diagonal line accross figure region
tmp2 <- cnvrt.coords( c(0,1), c(1,0), input='fig')
lines(tmp2$usr, col='red')

# save coordinate of point 1 and y value near top of plot for future plots
tmp.point1 <- cnvrt.coords(tmp.x[1], tmp.y[1])
tmp.range1 <- cnvrt.coords(NA, 0.98, input='plt')

# make a second plot and draw a line linking point 1 in each plot
plot(tmp.y, tmp.z)

tmp.point2 <- cnvrt.coords( tmp.point1$dev, input='dev' )
arrows( tmp.y[1], tmp.z[1], tmp.point2$usr$x, tmp.point2$usr$y,
 col='green')

# draw another plot and add rectangle showing same range in 2 plots

plot(tmp.x, tmp.z)
tmp.range2 <- cnvrt.coords(NA, 0.02, input='plt')
tmp.range3 <- cnvrt.coords(NA, tmp.range1$dev$y, input='dev')
rect( 9, tmp.range2$usr$y, 11, tmp.range3$usr$y, border='yellow')

# put a label just to the right of the plot and
#  near the top of the figure region.
text( cnvrt.coords(1.05, NA, input='plt')$usr$x,
cnvrt.coords(NA, 0.75, input='fig')$usr$y,
"Label", adj=0)

par(mfrow=c(1,1))

## create a subplot within another plot (see also subplot)

plot(1:10, 1:10)

tmp <- cnvrt.coords( c( 1, 4, 6, 9), c(6, 9, 1, 4) )
```

```
par(plt = c(tmp$dev$x[1:2], tmp$dev$y[1:2]), new=TRUE)
hist(rnorm(100))

par(fig = c(tmp$dev$x[3:4], tmp$dev$y[3:4]), new=TRUE)
hist(rnorm(100))

par(old.par)
```

`coin.faces` *Designs for coin faces for use with plot.rgl.coin*

Description

This is a list of matricies where each matrix represents a design for drawing lines on the face of a coin.

Usage

```
data(coin.faces)
```

Format

The format is: List of 4 $ qh: num [1:57, 1:2] 0.387 0.443 0.515 0.606 0.666 ... $ qt: num [1:62, 1:2] 0.862 0.873 0.875 0.857 0.797 ... $ H : num [1:28, 1:2] 0.503 0.506 0.548 0.548 0.500 ... $ T : num [1:18, 1:2] 0.506 0.520 0.569 0.626 0.626 ...

Details

The current options are a capitol "H", a capitol "T", a design representing George Washingtons head traced from the heads of a US quarter, and a design representing an eagle traced from the tails of a US quarter.

The tracings here have pretty much exhausted my artistic ability, if you can do better, please do, I will be happy to include it in future versions. It would also be nice to include some designs representing faces of non-US coins, please submit your contributions (the design should fit within a circle inscribed within the unit square).

Examples

```
## Not run:
plot.rgl.coin(heads=coin.faces$H, tails=coin.faces$T)

## End(Not run)
```

col2grey *Convert colors to grey/grayscale*

Description

Convert colors to grey/grayscale so that you can see how your plot will look after photocopying or printing to a non-color printer.

Usage

```
col2grey(cols)
col2gray(cols)
```

Arguments

cols	Colors to convert.

Details

converts colors to greyscale using the formula grey=0.3*red + 0.59*green + 0.11*blue. This allows you to see how your color plot will approximately look when printed on a non-color printer or photocopied.

Value

A vector of colors (greys) corresponding to the input colors.

Author(s)

Greg Snow <538280@gmail.com>

See Also

grey, col2rgb, dichromat package

Examples

```
par(mfcol=c(2,2))
tmp <- 1:3
names(tmp) <- c('red','green','blue')

barplot( tmp, col=c('red','green','blue') )
barplot( tmp, col=col2gray( c('red','green','blue') ) )

barplot( tmp, col=c('red','#008100','#3636ff') )
barplot( tmp, col=col2grey( c('red','#008100','#3636ff') ) )
```

`cor.rect.plot` *Plot a visualization of the correlation using colored rectangles*

Description

This function creates a scatterplot of the data, then adds colored rectangles between the points and the mean of x and y to represent the idea of the correlation coefficient.

Usage

```
cor.rect.plot(x, y, corr = TRUE, xlab = deparse(substitute(x)),
  ylab = deparse(substitute(y)), col = c("#ff000055", "#0000ff55"),
  ...)
```

Arguments

`x`	The `x` value or any object that can be interpreted by `xy.coords`
`y`	The `y` value
`corr`	Should the standardized axes (right and top) show the values divided by the standard deviation (TRUE, which shows correlation ideas) or not (FALSE, shows covariance idea)
`xlab`	The label for the `x` axis
`ylab`	The label for the `y` axis
`col`	A vector of length 2 with the colors to use for the fill of the rectangles, the 1st value will be used for "positive" rectangles and the 2nd value will be used for the "negative" rectangles.
`...`	Possible further arguments, currently ignored

Details

This will create a scatterplot of the data, draw refrence lines at the mean of `x` and the mean of `y`, then draw rectangles from the mean point to the data points. The right and top axes will show the centered (and possibly scaled if `corr=TRUE`) values.

The idea is that the correlation/covariance is based on summing the area of the "positive" rectangles and subtracting the sum of the areas of the "negative" rectangles (then dividing by n-1). If the positive and negative areas are about the same then the correlation/covariance is near 0, if there is more area in the positive rectangles then the correlation/covariance will be positive.

Value

This function returns an invisible NULL, it is run for its side effects.

Author(s)

Greg Snow, `<538280@gmail.com>`

See Also

cor

Examples

```
## low correlation
x <- rnorm(25)
y <- rnorm(25)
cor(x,y)
cor.rect.plot(x,y)

## Positive correlation
x <- rnorm(25)
y <- x + rnorm(25,3, .5)
cor(x,y)
cor.rect.plot(x,y)

## negative correlation
x <- rnorm(25)
y <- rnorm(25,10,1.5) - x
cor(x,y)
cor.rect.plot(x,y)

## zero correlation but a definite relationship
x <- -5:5
y <- x^2
cor(x,y)
cor.rect.plot(x,y)
```

---

dice *Simulate rolling dice*

Description

Simulate and optionally plot rolls of dice.

Usage

```
dice(rolls = 1, ndice = 2, sides = 6, plot.it = FALSE, load = rep(1, sides))
## S3 method for class 'dice'
plot(x, ...)
```

Arguments

rolls	Scalar, the number of times to roll the dice.
ndice	Scalar, the number of dice to roll each time.
sides	Scalar, the number of sides per die.
plot.it	Logical, Should the results be plotted.

load	Vector of length sides, how the dice should be loaded.
x	Data frame, return value from dice.
...	Additional arguments passed to lattice plotting function.

Details

Simulates the rolling of dice. By default it will roll 2 dice 1 time and the dice will be fair. Internally the sample function is used and the load option is passed to sample. load is not required to sum to 1, but the elements will be divided by the sum of all the values.

Value

A data frame with rolls rows and ndice columns representing the results from rolling the dice.

If only 1 die is rolled, then the return value will be a vector.

If plot.it is TRUE, then the return value will be invisible.

Note

If the plot function is used or if plot.it is TRUE, then a plot will be created on the current graphics device.

Author(s)

Greg Snow <538280@gmail.com>

See Also

sample

Examples

```
# 10 rolls of 4 fair dice
dice(10,4, plot.it=TRUE)

# or

plot(dice(10,4))

# or

tmp <- dice(10,4)
plot(tmp)

# a loaded die
table(tmp <- dice(100,1,plot.it=TRUE, load=6:1 ) )
colMeans(tmp)

# Efron's dice
ed <- list( rep( c(4,0), c(4,2) ),
```

```
  rep(3,6), rep( c(6,2), c(2,4) ),
  rep( c(5,1), c(3,3) ) )

tmp <- dice( 10000, ndice=4 )
ed.out <- sapply(1:4, function(i) ed[[i]][ tmp[[i]] ] )

mean(ed.out[,1] > ed.out[,2])
mean(ed.out[,2] > ed.out[,3])
mean(ed.out[,3] > ed.out[,4])
mean(ed.out[,4] > ed.out[,1])


## redo De Mere's question

demere1 <- dice(10000,4)
demere2 <- dice(10000,24,sides=36)

mean(apply( demere1, 1, function(x) 6 %in% x ))

mean(apply( demere2, 1, function(x) 36 %in% x))

plot(demere1[1:10,])

## plot all possible combinations of 2 dice

plot.dice( expand.grid(1:6,1:6), layout=c(6,6) )
```

`digits` *Return the digits that make up an integer*

Description

Takes an integer or vector of integers and returns a vector, list, or matrix of the individual digits (decimal) that make up that number.

Usage

```
digits(x, n = NULL, simplify = FALSE)
```

Arguments

`x`	An integer or vector of integers (if not integer, the fractional part will be ignored)
`n`	The number of digits to be returned, this can be used to place 0's in front of smaller numbers. If this is less than the number of digits then the last `n` digits are returned.
`simplify`	Should `sapply` simplify the list into a matrix

Details

This function transforms an integer (or real ignoring the fractional part) into the decimal digits that make of the decimal representation of the number using modular mathematics rather than converting to character, splitting the string, and converting back to numeric.

Value

If `x` is of length 1 then a vector of the digits is returned.

If `x` is a vector and `simplify` is `FALSE` then a list of vectors is returned, one element for each element of `x`.

If `x` is a vector and `simplify` is `TRUE` then a matrix with 1 column for each element of `x`.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`%%`, `%/%`, `strsplit`

Examples

```
digits( 12345 )
digits( 567, n=5 )

x <- c(1, 23, 456, 7890)

digits(x)
digits(x, simplify=TRUE)
```

`dots` *Create a quick dotchart (histogram)*

Description

Create a quick dotchart of 1 or 2 datasets. These dotcharts are a poor man's histogram, not the trellis dotplot.

Usage

```
dots(x,...)
dots2(x, y, colx = "green", coly = "blue", lab1 =
deparse(substitute(x)), lab2 = deparse(substitute(y)),...)
```

Arguments

x	Vector, data to be plotted (should be rounded).
y	Vector, second dataset to be plotted.
colx	Color of points for x.
coly	Color of points for y.
lab1	Label for x.
lab2	Label for y.
...	Additional arguments passed to plotting functions.

Details

These functions create basic dotcharts that are quick "back of the envelope" approximations to histograms. Mainly intended for demonstration.

Value

No meaninful value. These functions are run for the side effect of creating a plot.

Author(s)

Greg Snow <538280@gmail.com >

See Also

dotplot in the lattice package, hist

Examples

```
dots( round( rnorm(50, 10,3) ) )
dots2( round( rnorm(20, 10,3) ), round(rnorm(20,12,2)) )
```

---

dynIdentify *Interacively place labels for points in a plot*

Description

These functions create a scatterplot of your points and place labels for the points on them. You can then use the mouse to click and drag the labels to new positions with a line stretching between the point and label.

Usage

```
dynIdentify(x, y, labels = seq_along(x),
  corners = cbind(c(-1, 0, 1, -1, 1, -1, 0, 1),
    c(1, 1, 1, 0, 0, -1, -1, -1)), ...)
TkIdentify(x, y, labels=seq_along(x), hscale=1.75, vscale=1.75,
  corners = cbind( c(-1,0,1,-1,1,-1,0,1), c(1,1,1,0,0,-1,-1,-1) ),...)
```

Arguments

x	x-values to plot
y	y-values to plot
labels	Labels for the points, defaults to a sequence of integers
corners	2 column matrix of locations where the line can attach to the label, see below
hscale,vscale	Scaling passed to tkrplot
...	Additional parameters passed to plot

Details

These functions create a scatterplot of the x and y points with the labels (from the argument above) plotted on top. You can then use the mouse to click and drag the labels to new locations. The Tk version shows the labels being dragged, dynIdentify does not show the labels being dragged, but the label will jump to the new location as soon as you release the mouse button.

The corners argument is a 2 column matrix that gives the allowable points at which the line from the point can attach to the label (so the line does not cover thelabel). The first column represents the x-coordinates and the 2nd column the y-coordinates. A 1 represents the right/top of the label, A -1 is the left/bottom and a 0 is the center. The default values allow attachments at the 4 corners and the centers of the 4 sides of the rectangle bounding the label.

Value

A list of lists with the coordinates of the final positions of the labels and the line ends.

Note

The dynIdentify function only works on windows, TkIdentify should work on any platform with tcltk.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

identify

Examples

```
if(interactive()) {
  tmp <- TkIdentify(state.x77[,'Frost'], state.x77[,'Murder'],
  state.abb)
     ### now move the labels

     ### recreate the graph on the current device
  plot( state.x77[,'Frost'], state.x77[,'Murder'],
        xlab='Frost', ylab='Frost')
  text( tmp$labels$x, tmp$labels$y, state.abb )
  segments( state.x77[,'Frost'], state.x77[,'Murder'],
            tmp$lineends$x, tmp$lineends$y )
}
```

---

evap *Data on soil evaporation.*

Description

Data from 46 consecutive days on weather variables used to estimate amount of evaporation from the soil.

Usage

```
data(evap)
```

Format

A data frame with 46 observations on the following 14 variables.

`Obs` Observation number

`Month` Month (6-June, 7-July)

`day` Day of the month

`MaxST` Maximum Soil Temperature

`MinST` Minimum Soil Temperature

`AvST` Average (integrated) Soil Temperature

`MaxAT` Maximum Air Temperature

`MinAT` Minimum Air Temperature

`AvAT` Average (integrated) Air Temperature

`MaxH` Maximum Relative Humidity

`MinH` Minimum Relative Humidity

`AvH` Average (integrated) Relative Humidity

`Wind` Total Wind

`Evap` Total evoporation from the soil

Details

The idea of the data is to predict the amount of evaporation given the other variables. Note that the "average" values are scaled differently from the others, this is more an area under the curve measure representing the total/average value.

This dataset was entered by hand from a low quality copy of the paper. If you find any typos, please e-mail them to the package maintainer.

Source

Freund, R.J. (1979) Multicollinearity etc., Some "New" Examples. Proceedings of the Statistical Computing Section, *4*, 111-112.

Examples

```
data(evap)
pairs(evap[,-c(1,2,3)], panel=panel.smooth)
## maybe str(evap) ; plot(evap) ...
```

---

faces *Chernoff Faces*

Description

faces represent the rows of a data matrix by faces

Usage

```
faces(xy, which.row, fill = FALSE, nrow, ncol, scale = TRUE, byrow = FALSE, main, labels)
```

Arguments

xy	xy data matrix, rows represent individuals and columns attributes
which.row	defines a permutation of the rows of the input matrix
fill	if(fill==TRUE), only the first nc attributes of the faces are transformed, nc is the number of columns of xy
nrow	number of columns of faces on graphics device
ncol	number of rows of faces
scale	if(scale==TRUE), attributes will be normalized
byrow	if(byrow==TRUE), xy will be transposed
main	title
labels	character strings to use as names for the faces

Details

The features paramters of this implementation are: 1-height of face, 2-width of face, 3-shape of face, 4-height of mouth, 5-width of mouth, 6-curve of smile, 7-height of eyes, 8-width of eyes, 9-height of hair, 10-width of hair, 11-styling of hair, 12-height of nose, 13-width of nose, 14-width of ears, 15-height of ears. For details look at the literate program of `faces`

Value

a plot of faces is created on the graphics device, no numerical results

Note

version 12/2003

Author(s)

H. P. Wolf

References

Chernoff, H. (1973): The use of faces to represent statistiscal assoziation, JASA, 68, pp 361–368. The smooth curves are computed by an algorithm found in Ralston, A. and Rabinowitz, P. (1985): A first course in numerical analysis, McGraw-Hill, pp 76ff. http://www.uni-bielefeld.de/fakultaeten/wirtschaftswissenschaften : S/R - functions : faces

See Also

—

Examples

```
faces(rbind(1:3,5:3,3:5,5:7))

data(longley)
faces(longley[1:9,])

set.seed(17)
faces(matrix(sample(1:1000,128,),16,8),main="random faces")

if(interactive()){
  tke1 <- rep( list(list('slider',from=0,to=1,init=0.5,resolution=0.1)), 15)
  names(tke1) <- c('FaceHeight','FaceWidth','FaceShape','MouthHeight',
'MouthWidth','SmileCurve','EyesHeight','EyesWidth','HairHeight',
'HairWidth','HairStyle','NoseHeight','NoseWidth','EarWidth','EarHeight')
  tkfun1 <- function(...){
tmpmat <- rbind(Min=0,Adjust=unlist(list(...)),Max=1)
faces(tmpmat, scale=FALSE)
  }

  tkexamp( tkfun1, list(tke1), plotloc='left', hscale=2, vscale=2 )
}
```

faces2 *Chernoff Faces*

Description

Plot Chernoff Faces of the dataset, rows represent subjects/observations, columns represent variables.

Usage

```
faces2(mat, which = 1:ncol(mat), labels = rownames(mat),
  nrows = ceiling(nrow(mat)/ncols), ncols = ceiling(sqrt(nrow(mat))),
  byrow = TRUE, scale = c("columns", "all", "center", "none"),
  fill = c(0.5, 0.5, 1, 0.5, 0.5, 0.3, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
           0.5, 0.5, 0.5, 0.5, 1, 0.5), ...)
```

Arguments

mat	Matrix containing the data to plot.
which	Which columns correspond to which features (see details).
labels	Labels for the individual faces
nrows	Number of rows in the graphical layout
ncols	Number of columns in the graphical layout
byrow	Logical, should the faces be drawn rowwise or columnwise.
scale	Character, how should the data be scaled.
fill	What value to use for features not assocaiated with a column of data.
...	Additional arguments passed on to plotting functions.

Details

The features are: 1 Width of center 2 Top vs. Bottom width (height of split) 3 Height of Face 4 Width of top half of face 5 Width of bottom half of face 6 Length of Nose 7 Height of Mouth 8 Curvature of Mouth (abs < 9) 9 Width of Mouth 10 Height of Eyes 11 Distance between Eyes (.5-.9) 12 Angle of Eyes/Eyebrows 13 Circle/Ellipse of Eyes 14 Size of Eyes 15 Position Left/Right of Eyeballs/Eyebrows 16 Height of Eyebrows 17 Angle of Eyebrows 18 Width of Eyebrows

The face plotting routine needs the data values to be between 0 and 1 (inclusive). The `scale` option controls how scaling will be done on `mat`: "columns" scales each column to range from 0 to 1, "all" scales the entire dataset to vary from 0 to 1, "center" scales each column so that the mean of the column becomes 0.5 and all other values are between 0 and 1, and "none" does no scaling assuming that the data has already been scaled.

Value

This function is run for its side effect of plotting and does not return anything.

Note

If you choose to not scale the data and any data values are outside of the 0 to 1 range, then strange things may happen.

This function is based on code found on the internet, the good things come from there, any problems are likely due to my (Greg's) tweaking.

Author(s)

Original code by ; current implementation by Greg Snow <538280@gmail.com>

References

Chernoff, H. (1973): The use of faces to represent statistiscal assoziation, JASA, 68, pp 361–368.

See Also

faces

Examples

```
  faces2(matrix( runif(18*10), nrow=10), main='Random Faces')

if(interactive()){
  tke2 <- rep( list(list('slider',from=0,to=1,init=0.5,resolution=0.1)), 18)
  names(tke2) <- c('CenterWidth','TopBottomWidth','FaceHeight','TopWidth',
'BottomWidth','NoseLength','MouthHeight','MouthCurve','MouthWidth',
'EyesHeight','EyesBetween','EyeAngle','EyeShape','EyeSize','EyeballPos',
'EyebrowHeight','EyebrowAngle','EyebrowWidth')
  tkfun2 <- function(...){
tmpmat <- rbind(Min=0,Adjust=unlist(list(...)),Max=1)
faces2(tmpmat, scale='none')
  }

  tkexamp( tkfun2, list(tke2), plotloc='left', hscale=2, vscale=2 )
}
```

---

fagan.plot *Create a Fagan plot to demonstrate Bayes Theorem and screening tests*

Description

These functions create a plot showing the relationship between the prior probability, the LR (combination of sensitivity and specificity), and the posterior probability.

Usage

```
fagan.plot(probs.pre.test, LR, test.result="+")
plotFagan(hscale=1.5, vscale=1.5, wait=FALSE)
plotFagan2(hscale=1.5, vscale=1.5, wait=FALSE)
plotFagan.old()
plotFagan2.old()
```

Arguments

probs.pre.test	The prior probability
LR	the likelihood ratio (sensitivity/(1-specificity))
test.result	either '+' or '-' indicating whether you want the probability of the event or of not seeing the event
hscale	Horizontal scale, passed to tkrplot
vscale	Vertical scale, passed to tkrplot
wait	Should the R session wait for the window to close

Details

When Bayes theorem is expressed in terms of log-odds it turns out that the posterior log-odds are a linear function of the prior log-odds and the log likelihood ratio. These functions plot an axis on the left with the prior log-odds, an axis in the middle representing the log likelihood ratio and an axis on the right representing the posterior log-odds. A line is then drawn from the prior probability on the left through the LR in the center and extended to the posterior probability on the right. The fagan.plot creates the plot based on input to the function. The plotFagan and plotFagan2 functions set up Tk windows with sliders representing the possible inputs and show how the plot and the posterior probability changes when you adjust the inputs. The plotFagan function creates sliders for the prior probability and the LR, while the plotFagan2 function replaces the LR slider with 2 sliders for the sensitivity and specificity.

More detail on the plots and the math behind them can be found at the websites below.

Value

The old functions are run for their side effects and do not return a meaningful value. If wait is FALSE then NULL is returned, if wait is TRUE, then a list with the current values is returned.

Author(s)

Guazzetti Stefano and Greg Snow <538280@gmail.com>

References

Fagan TJ. Nomogram for Bayes theorem. N Engl J Med 1975;293(5):257-61. https://ebm.bmj.com/content/6/6/164.full

See Also

slider

Examples

```
 fagan.plot(0.8, 2)

 fagan.plot(0.8, 0.95/(1-0.90) )

if(interactive()) {
 plotFagan()

 plotFagan2()
}
```

---

gp.open *Alpha version functions to send plotting commands to GnuPlot*

Description

These functions allow you to open a connection to a gnuplot process, send data and possibly other information to gnuplot for it to plot, then close gnuplot and clean up temporary files and variables. These functions are alpha level at best, use at your own risk.

Usage

```
gp.open(where='c:/progra~1/GnuPlot/bin/pgnuplot.exe')
gp.close(pipe=gpenv$gp)
gp.send(cmd='replot',pipe=gpenv$gp)
gp.plot(x,y,type='p',add=FALSE, title=deparse(substitute(y)),pipe=gpenv$gp)
gp.splot(x,y,z, add=FALSE, title=deparse(substitute(z)), pipe=gpenv$gp,
  datafile=tempfile())
```

Arguments

where	Path to GnuPlot Executable
pipe	The pipe object connected to GnuPlot (returned from gp.open), warning: changing this from the default will probably break things
cmd	Text string, the command to be sent verbatim to the GnuPlot process
x	The x coordinates to plot
y	the y coordinates to plot
z	the z coordinates to splot
type	Either 'p' or 'l' for plotting points or lines
add	Logical, should the data be added to the existing plot or start a new plot
title	The title or legend entry
datafile	The file to store the data in for transfer to gnuplot

Details

These functions provide a basic interface to the GnuPlot program (you must have GnuPlot installed (separate install)), `gp.open` runs GnuPlot and establishes a pipe connection, `gp.close` sends a quite command to gnuplot and cleans up temporary variables and files, `gp.send` sends a command to the GnuPlot process verbatim, and `gp.plot` sends data and commands to the process to create a standard scatterplot or line plot.

Value

`gp.open` returns and invisible copy of the pipe connection object (to pass to other functions, but don't do this because it doesn't work right yet).

The other 3 functions don't return anything meaningful. All functions are run for their side effects.

Note

These functions create some temporary files and 2 temporary global variables (.gp and .gp.tempfiles), running `gp.close` will clean these up (so use it).

These functions are still alpha level.

Author(s)

Greg Snow <538280@gmail.com>

References

http://www.gnuplot.info/

See Also

plot

Examples

```
 ## Not run:

x <- 1:10
y <- 3-2*x+x*x+rnorm(10)

gp.open()
gp.plot(x,y)
gp.send('replot 3-2*x+x**2')

tmp <- expand.grid(x=1:10, y=1:10)
tmp <- transform(tmp, z=(x-5)*(y-3))
gp.splot(tmp$x, tmp$y, tmp$z)

gp.close()

## End(Not run)
```

hpd *Compute Highest Posterior Density Intervals*

Description

Compute the Highest Posterior Density Interval (HPD) from an inverse density function (hpd) or a vector of realizations of the distribution (emp.hpd).

Usage

```
hpd(posterior.icdf, conf=0.95, tol=0.00000001,...)

emp.hpd(x, conf=0.95, lower, upper)
```

Arguments

posterior.icdf	Function, the inverse cdf of the posterior distribution (usually a function whose name starts with 'q').
x	A vector of realizations from the posterior distribution.
conf	Scalar, the credible level desired.
tol	Scalar, the tolerance for optimize.
...	Additional arguments to posterior.icdf.
lower	Optional lower bound on support of x.
upper	Optional upper bound on support of x.

Details

These functions compute the highest posterior density intervals (sometimes called minimum length confidence intervals) for a Bayesian posterior distribution. The hpd function is used when you have a function representing the inverse cdf (the common case with conjugate families). The emp.hpd function is used when you have realizations of the posterior (when you have results from an MCMC run).

Value

A vector of length 2 with the lower and upper limits of the interval.

Note

These functions assume that the posterior distribution is unimodal, they compute only 1 interval, not the set of intervals that are appropriate for multimodal distributions.

Author(s)

Greg Snow <538280@gmail.com>

See Also

hdr in the hdrcde package.

Examples

```
hpd(qbeta, shape1=50, shape2=250)

tmp <- rbeta(10000, 50, 250)
emp.hpd(tmp)
```

HWidentify *Show label for point being Hovered over.*

Description

These functions create a scatterplot then you Hover the mouse pointer over a point in the plot and it will show an id label for that point.

Usage

```
HWidentify(x, y, label = seq_along(x), lab.col="darkgreen",
pt.col="red", adj=c(0,0), clean=TRUE, xlab = deparse(substitute(x)),
ylab = deparse(substitute(y)), ...)
HTKidentify(x, y, label = seq_along(x), lab.col="darkgreen",
pt.col="red", adj=c(0,0), xlab = deparse(substitute(x)),
ylab = deparse(substitute(y)), ...)
```

Arguments

x	x-coordinates to plot
y	y-coordinates to plot
label	Labels to show for each point
lab.col	The color to plot the labels
pt.col	The color of the highlighting point
adj	The adjustment of the labels relative to the cursor point. The default places the label so that its bottom left corner is at the curser, values below 0 or greater than 1 will move the label to not touch the cursor.
clean	Logical value, should any labels on the plot be removed at the end of the plotting.
xlab	Label for x-axis
ylab	Label for y-axis
...	additional arguments passed through to plot

Details

This is an alternative to the `identify` function. The label only shows up for the point currently closest to the mouse pointer. When the mouse pointer moves closer to a different point, the label changes to the one for the new point. The currently labeled point is also highlighted. HWidentify only works on windows, HTKidentify requires the tkrplot package.

Value

These functions are run for their side effects, nothing meaningful is returned.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

`identify`

Examples

```
if( interactive() ){
  tmpx <- runif(25)
  tmpy <- rnorm(25)
  HTKidentify(tmpx,tmpy, LETTERS[1:25], pch=letters)
}
```

`lattice.demo` *Interactively explore the conditioned panels in lattice plots.*

Description

Plot 1 panel from an xyplot, and optionally a 3d graph highligting the shown points, then allow you to interactively set the conditioning set of data to see the effects and help you better understand how xyplot works.

Usage

```
lattice.demo(x, y, z, show3d = TRUE)
```

Arguments

x	The x variable to plot (numeric).
y	The y variable to plot (numeric).
z	The variable to condition on (numeric).
show3d	Logical, should a 3D cloud be shown as well.

Details

This function is intended to for demonstration purposes to help understand what is happening in an `xyplot` (lattice). When you run the demo it will create a single panel from a conditioned `xyplot` and optionally a 3D cloud with the points included in the panel highlighted. The function then opens a tcl/tk dialog box that allows you to choose which points are included in the panel (based on the conditioning variable). You can choose the center and width of the shingle displayed and the graph will update to show the new selection.

The intent for this function is for a teacher to show a class how lattice graphics take slices of a 3d plot and show each slice seperately. Students could then work through some examples on their own to better understand what functions like `xyplot` are doing automatically.

Value

No meaningful return value, this function is run for the side effects.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`xyplot` in lattice package

Examples

```
if(interactive()){
require(stats)
lattice.demo(quakes$long, quakes$lat, quakes$depth)
}
```

---

ldsgrowth *Growth of The Church of Jesus Christ of Latter-day Saints.*

Description

Data on the Growth of The Church of Jesus Christ of Latter-day Saints (commonly known as the Mormon church (https://www.churchofjesuschrist.org/comeuntochrist)).

Usage

```
data(ldsgrowth)
```

Format

A data frame with 179 observations on the following 6 variables.

`Year` Year from 1830 to 2008

`Members` Total number of Members

`Wards` Number of Wards and Branches (individual congregations)

`Stakes` Number of Stakes (a group of wards/branches)

`Missions` Number of Missions

`Missionaries` Number of Missionaries called

Details

The data comes from the church records and are as of December 31st of each year.

The church was officially organized on 6 April 1830 (hence the starting year of 1830).

The `Missionaries` column represents the number of missionaries called each year. Missionaries generally serve for about 2 years.

Source

Deseret News 2010 Church News Almanac

Examples

```
data(ldsgrowth)
with(ldsgrowth, plot(Year, log(Members)))
```

---

loess.demo *Demonstrate the internals of loess curve fits*

Description

Creates a scatterplot with a loess fit, then interactively shows the window and case weights used to create the curve at the selected value of `x`.

Usage

```
loess.demo(x, y, span = 2/3, degree = 1, nearest = FALSE,
  xlim = numeric(0), ylim = numeric(0), verbose = FALSE)
```

Arguments

x	The x coordinates to be plotted.
y	The y coordinates to be plotted.
span	The relative width of the window, passed on to loess.
degree	Degree of polynomial to use (0, 1, or 2), passed on to loess.
nearest	Logical, should predictions be made at the point where you clicked (FALSE), or at the nearest x value of the data to where you clicked (TRUE).
xlim	Limits of the Horizonal axis.
ylim	Limits of the Vertical axis.
verbose	If true then print the x coordinate being predicted.

Details

This function demonstrates the underlying calculations of loess curves.

Given x and y vectors it will create a scatterplot and add 2 loess fit lines (one using straight loess smooth with linear interpolation and one that does a spline interpolation of the loess fit).

The function then waits for the user to click on the plot. The function then shows the window of points (centered at the x value clicked on) used in the weighting for predicting that point and shows a circle around each point in the window where the area of the circle is proportional to the weight of that point in the linear fit. The function also shows the linear (or quadratic) fit used to predict at the selected point.

The basic steps of the loess algorithm (as demonstrated by the function) is that to predict the y-value for a given x-value the computer:

1. Find all the points within a window around the x-value (the width of the window is based on the parameter span). 2. Weight the points in the window with points nearest the x-value having the highest weight. 3. Fit a weighted linear (quadratic) line to the points in the window. 4. Use the y-value of the fitted line (curve) at the x-value to give loess prediction at that x-value.

Clicking on another point in the graph will replot with the new situation.

Right click and select 'stop' to end the demonstration.

Value

This function does not return anything, it is run purely for its side effects.

Author(s)

Greg Snow <538280@gmail.com>

See Also

loess, locator

Examples

```
if(interactive()){
data(ethanol, package='lattice')
attach(ethanol)
loess.demo(E, NOx)
# now click a few places, right click to end
loess.demo(E, NOx, span=1.5)
loess.demo(E, NOx, span=0.25)
loess.demo(E, NOx, degree=0)
loess.demo(E, NOx, degree=2)
detach()
}
```

---

mle.demo *Demonstrate the basic concept of Maximum Likelihood Estimation*

Description

This function graphically shows log likelihoods for a set of data and the normal distribution and allows you to interactively change the parameter estimates to see the effect on the log likelihood.

Usage

```
mle.demo(x = rnorm(10, 10, 2), start.mean = mean(x) - start.sd,
  start.sd = 1.2 * sqrt(var(x)))
```

Arguments

x	A vector of data (presumably from a normal distribution).
start.mean	The initial value for estimating the mean.
start.sd	The initial value for estimating the standard deviation.

Details

The function creates a plot with 3 panels: the top panel shows a normal curve based on the current values of the mean and standard deviation along with a vertical line for each point in x (the product of the heights of these lines is the likelihood, the sum of the logs of their heights is the log likelihood).

The lower 2 plots show the profiles of the mean and standard deviation. The y-axis is the likelihoods of the parameters tried so far, and the x-axes are the mean and standard deviation tried. The point corresponding to the current parameter estimates will be solid red.

A Tk slider box is also created that allows you to change the current estimates of the mean and standard deviation to show the effect on the log likelihood and find the maximum likelihood estimate.

Value

This function is run for its side effects and returns NULL.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`fitdistr` in package MASS, `mle` in package stats4, `slider`

Examples

```
if(interactive()){
mle.demo()

m <- runif(1, 50,100)
s <- runif(1, 1, 10)
x <- rnorm(15, m, s)

mm <- mean(x)
ss <- sqrt(var(x))
ss2 <- sqrt(var(x)*11/12)
mle.demo(x)
# now find the mle from the graph and compare it to mm, ss, ss2, m, and s
}
```

`ms.polygram` *Symbol functions/data to be passed as symb argument to my.symbols*

Description

These functions/data matricies are examples of what can be passed as the `symb` argument in the `my.symbols` function. They are provided both to be used for some common symbols and as examples of what can be passed as the `symb` argument.

Usage

```
ms.polygram(n, r=1, adj=pi/2, ...)
ms.polygon(n, r=1, adj=pi/2, ...)
ms.filled.polygon(n, r=1, adj=pi/2, fg=par('fg'), bg=par('fg'), ... )
ms.male
ms.female
ms.arrows(angle, r=1, adj=0.5, length=0.1, ...)
ms.sunflowers(n,r=0.3,adj=pi/2, ...)
ms.image(img, transpose=TRUE, ...)
ms.face(features, ...)
```

Arguments

n	The number of sides for polygons and polygrams, the number of petals(lines) for sunflowers.
r	The radius of the enclosing circle for polygons and polygrams (1 means that it will pretty much fill the bounding square). For sunflowers this is the radius (relative to the inches square) of the inner circle. For arrows this controls the length of the arrow, a value of 2 means the length of the arrow will be the same as inches (but it may then stick out of the box if adj != 1).
adj	For polygons, polygrams, and sunflowers this is the angle in radians that the first corner/point will be. The default puts a corner/point straight up, this can be used to rotate the symbols. For arrows, this determines the positioning of the arrow, a value of 0 means the arrow will start at the x,y point and point away from it, 0.5 means the arrow will be centered at x,y and 1 means that the arrow will end (point at) x,y.
fg, bg	Colors for the filled polygons. fg is the color of the line around the polygon and bg is the fill color, see polygon.
angle	The angle in radians that the arrow will point.
length	The length of the arrow head (see arrows).
img	A 3 dimensional array representing an image such as produced by the png or EBImage packages.
transpose	Should the image be tranposed, use TRUE for images imported using package png and FALSE for images imported using EBImage.
features	A list of data representing the features of the faces, each element represents 1 face and the values need to be scaled between 0 and 1, see faces for details on which elements match which features.
...	additional parameters that will be passed to plotting functions or be ignored.

Details

These functions/matricies can be passed as the symb argument to the my.symbols function. The represent examples that can be used to create your own symbols or may be used directly.

Value

These functions either return a 2 column matrix of points to be passed to lines or NULL.

Author(s)

Greg Snow <538280@gmail.com>

See Also

my.symbols, polygon, arrows, lines, faces, also see rasterImage for an alternative to ms.image

Examples

```
plot(1:10,1:10)
my.symbols(1:10,1:10, ms.polygram, n=1:10, r=seq(0.5,1,length.out=10),
inches=0.3)

my.symbols(1:10,1:10, ms.polygon, n=1:10, add=FALSE, inches=0.3)

my.symbols(1:5, 5:1, ms.filled.polygon, add=FALSE, n=3:7, fg='green',
  bg=c('red','blue','yellow','black','white'), inches=0.3 )

my.symbols( 1:10, 1:10, ms.female, inches=0.3, add=FALSE)
my.symbols( 1:10, 10:1, ms.male, inches=0.3, add=TRUE)

plot(1:10, 1:10)
my.symbols(1:10, 1:10, ms.arrows, angle=runif(10)*2*pi, inches=0.5,
adj=seq(0,1,length.out=10), symb.plots=TRUE)

my.symbols(1:10, 1:10, ms.sunflowers, n=1:10, inches=0.3, add=FALSE)

if( require(png) ) {
  img <- readPNG(system.file("img", "Rlogo.png", package="png"))

  my.symbols( runif(10), runif(10), ms.image, MoreArgs=list(img=img),
                inches=0.5, symb.plots=TRUE, add=FALSE)
}

tmp.mtcars <- scale(mtcars, center=sapply(mtcars,min),
scale=sapply(mtcars,function(x) diff(range(x))) )
tmp2.mtcars <- lapply( seq_len(nrow(tmp.mtcars)), function(i) tmp.mtcars[i,] )
my.symbols(mtcars$wt, mtcars$mpg, ms.face, inches=0.3, features=tmp2.mtcars,
add=FALSE)
```

`my.symbols` *Draw Symbols (User Defined) on a Plot*

Description

This function draws symbols on a plot. It is similar to the builtin `symbols` function with the difference that it plots symbols defined by the user rather than a prespecified set of symbols.

Usage

```
my.symbols(x, y=NULL, symb, inches=1, xsize, ysize,
                       add=TRUE,
                       vadj=0.5, hadj=0.5,
                       symb.plots=FALSE,
```

```
                xlab=deparse(substitute(x)),
                ylab=deparse(substitute(y)), main=NULL,
                xlim=NULL, ylim=NULL, linesfun=lines,
                ..., MoreArgs)
```

Arguments

x, y	The x and y coordinates for the position of the symbols to be plotted. These can be specified in any way which is accepted by xy.coords.
symb	Either a matrix, list, or function defining the symbol to be plotted. If it is a matrix or list it needs to be formatted that it can be passed directly to the lines function. It then defines the shape of the symbol on on a range/domain of -1 to 1. If this is a function it can either return a matrix or list as above (points on the range/domain of -1 to 1), or it can do the plotting itself.
inches	The size of the square containing the symbol in inches (note: unlike symbols this cannot be FALSE). This is ignored if xsize or ysize is specified.
xsize	The width of the bounding box(s) of the symbols in the same units as the x variable. Computed from ysize or inches if not specified. Can be a single value or a vector.
ysize	The height of the bounding box(s) of the symbols in the same units as the y variable. Computed from xsize or inches if not specified. Can be a single value or a vector.
add	if 'add' is 'TRUE' then the symbols are added to the existing plot, otherwise a new plot is created.
vadj,hadj	Numbers between 0 and 1 indicating how 'x' and 'y' specify the location of the symbol. The defaults center the symbol at x,y; 0 means put the bottom/left at x,y; and 1 means put the top/right of the symbol at x,y.
symb.plots	If symb is a function that does its own plotting, set this to TRUE, otherwise it should be FALSE.
xlab, ylab, main, xlim, ylim	If 'add' is 'FALSE' these are passed to the plot function when setting up the plot.
linesfun	The function to draw the lines if the function does not do its own drawing. The default is lines but could be replaced with polygon to draw filled polygons
...	Additional arguments will be replicated to the same length as x then passed to symb (if symb is a function) and/or the lines function (one value per symbol drawn).
MoreArgs	A list with any additional arguments to be passed to the symb function (as is, without being replicated/split).

Details

The symb argument can be a 2 column matrix or a list with components 'x' and 'y' that defines points on the interval [-1,1] that will be connected with lines to draw the symbol. If you want a closed polygon then be sure to replicate the 1st point as the last point.

If any point contains an NA then the line will not be drawn to or from that point. This can be used to create a symbol with disjoint parts that should not be connected.

If symb is a function then it should include a '...' argument along with any arguments to define the symbol. Any unmatched arguments that end up in the '...' argument will be replicated to the same length as 'x' (using the rep function) then the values will be passed one at a time to the symb function. If MoreArgs is specified, the elements of it will also be passed to symb without modification. The symb function can either return a matrix or list with the points that will then be passed to the lines function (see above). Or the function can call the plotting functions itself (set symb.plots to TRUE). High level plotting can be done (plot, hist, and other functions), or low level plotting functions (lines, points, etc) can be used; in this case they should add things to a plot with 'x' and 'y' limits of -1 to 1.

The size of the symbols can be specified by using inches in which case the symbol will be set inside of squares whose sizes are inches size based on the plotting device. The size can also be set using xsize and/or ysize which use the same units as the x and/or y variables. If only one is specified then the box will be square. If both are specified and they do not match the aspect ratio of the plot then the bounding box will not be square and the symbol will be distorted.

Value

This function is run for its side effect of plotting, it returns an invisible NULL.

Note

Since the '...' argument is passed to both lines and symb, the symb function should have a '...' argument so that it will ignore any additional arguments.

Arguments such as 'type' can be passed through the '...' argument if you want the symbol made of something other than lines.

Plotting coordinates and sizes are based on the size of the device at the time the function is called. If you resize the device after plotting, all bets are off.

Currently missing values in x or y are not handled well. It is best if remove all missing values first.

Author(s)

Greg Snow <538280@gmail.com>

See Also

symbols, subplot, mapply, ms.polygram, lines

Examples

```
# symb is matrix

my.symbols( 1:10, runif(10), ms.male, add=FALSE, xlab='x',
  ylab='y', inches=0.3, col=c('blue','green'), xlim=c(0,11), ylim=c(-0.1,1.1))
my.symbols( (1:10)+0.5, runif(10), ms.female, add=TRUE, inches=0.3,
  col=c('red','green') )
```

```
# symb is function returning matrix

plot(1:10, 1:10)
my.symbols( 1:10, 1:10, ms.polygram, n=1:10, inches=0.3 )


# symb is plotting function
# create a variation on monthplot

fit <- lm( log(co2) ~ time(co2) )
fit.r <- resid(fit)

x <- 1:12
y <- tapply(fit.r, cycle(co2), mean)

tmp.r <- split( fit.r, cycle(co2) )
tmp.r <- lapply( tmp.r, function(x) x-mean(x) )

yl <- do.call('range',tmp.r)

tmpfun <- function(w,data,ylim,...){
  tmp <- data[[w]]
  plot(seq(along=tmp),tmp, type='l', xlab='',ylab='',
       axes=FALSE, ylim=ylim)
  lines(par('usr')[1:2], c(0,0), col='grey')
}

my.symbols(x,y, symb=tmpfun, inches=0.4, add=FALSE, symb.plots=TRUE,
  xlab='Month',ylab='Adjusted CO2', xlim=c(0.5,12.5),
  ylim=c(-0.012,0.012),
  w=1:12, MoreArgs=list(data=tmp.r,ylim=yl) )

# using xsize and ysize
plot( 1:10, (1:10)*100, type='n', xlab='', ylab='' )
my.symbols( 5, 500, ms.polygon, n=250, inches=1.5 )
my.symbols( 5, 500, ms.polygon, n=250, xsize=2, col='blue' )
my.symbols( 5, 500, ms.polygon, n=250, ysize=200, col='green' )
my.symbols( 5, 500, ms.polygon, n=250, xsize=2, ysize=200, col='red' )
abline( v=c(4,6), col='grey' )
abline( h=c(400, 600), col='grey' )


# hand crafted hexagonal grid

x1 <- seq(0, by=2*sqrt(3), length.out=10)
y1 <- seq(0, by=3, length.out=10)

mypoints <- expand.grid(x=x1, y=y1)
mypoints[,1] <- mypoints[,1] + rep( c(0,sqrt(3)), each=10, length.out=100 )

plot(mypoints, asp=1, xlim=c(-2,35))
my.symbols(mypoints, symb=ms.filled.polygon, n=6,
  inches=par('pin')[1]/(diff(par('usr')[1:2]))*4,
```

```
bg=paste('gray',1:100,sep=''), fg='green' )
```

---

outliers *Outliers data*

Description

This dataset is approximately bell shaped, but with some outliers. It is meant to be used for demonstration purposes. If students are tempted to throw out all outliers, then have them work with this data (or use a scaled/centered/shuffled version as errors in a regression problem) and see how many throw away 3/4 of the data before rethinking their strategy.

Usage

```
data(outliers)
```

Format

The format is: num [1:100] -1.548 0.172 -0.638 0.233 -0.228 ...

Details

This is simulated data meant to demonstrate "outliers".

Source

Simulated, see the examples section.

Examples

```
data(outliers)
qqnorm(outliers)
qqline(outliers)
hist(outliers)

o.chuck <- function(x) {  # function to throw away outliers
qq <- quantile(x, c(1,3)/4, names=FALSE)
r <- diff(qq) * 1.5
tst <- x < qq[1] - r | x > qq[2] + r
if(any(tst)) {
cat('Removing ', paste(x[tst], collapse=', '), '\n')
x <- x[!tst]
out <- Recall(x)
} else {
out <- x
}
out
}
```

```
x <- o.chuck( outliers )
length(x)

if(require(MASS)) {
  char2seed('robust')
  x <- 1:100
  y <- 3 + 2*x + sample(scale(outliers))*10

  plot(x,y)
  fit <- lm(y~x)
  abline(fit, col='red')

  fit.r <- rlm(y~x)
  abline(fit.r, col='blue', lty='dashed')

  rbind(coef(fit), coef(fit.r))
  length(o.chuck(resid(fit)))
}



### The data was generated using code similar to:

char2seed('outlier')

outliers <- rnorm(25)

dir <- 1

while( length(outliers) < 100 ){
qq <- quantile(c(outliers, dir*Inf), c(1,3)/4)
outliers <- c(outliers,
qq[ 1.5 + dir/2 ] + dir*1.55*diff(qq) + dir*abs(rnorm(1)) )
dir <- -dir
}
```

---

pairs2 *Create part of a scatterplot matrix*

Description

This function is similar to the pairs function, but instead of doing all pairwise plots, it takes 2 matricies or data frames and does all combinations of the first on the x-axis with the 2nd on the y-axis. Used with pairs and subsets can spread a scatterplot matrix accross several pages.

Usage

```
pairs2(x, y, xlabels, ylabels, panel = points, ..., row1attop = TRUE, gap = 1)
```

Arguments

x	Matrix or data frame of variables to be used as the x-axes.
y	Matrix or data frame of variables to be used as the y-axes.
xlabels	Labels for x variables (defaults to colnames of x).
ylabels	Labels for y variables (defaults to colnames of y).
panel	Function to do the plotting (see pairs).
...	additional arguments passed to graphics functions
row1attop	Logical, should the 1st row be the top.
gap	Distance between plots.

Details

This functios is similar to the pairs function, but by giving it 2 sets of data it only does the combinations between them. Think of it as giving the upper right or lower left set of plots from pairs. If a regular scatterplot matrix is too small on the page/device then use pairs on subsets of the data to get the diagonal blocks of a scatterplot matrix and this function to get the off diagonal blocks.

Value

This function is run for the side effect of the plot. It does not return anything useful.

Note

Large amounts of the code for this function were blatently borrowed/stolen from the pairs function, the credit for the useful parts should go to the original authors, blame for any problems should go to me. This function is also released under GPL since much of it comes from GPL code.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

pairs, splom in the lattice package

Examples

```
pairs2(iris[,1:2], iris[,3:4], col=c('red','green','blue')[iris$Species])

# compare the following plot:
pairs(state.x77, panel=panel.smooth)

# to the following 4 plots

pairs(state.x77[,1:4], panel=panel.smooth)
pairs(state.x77[,5:8], panel=panel.smooth)
pairs2( state.x77[,1:4], state.x77[,5:8], panel=panel.smooth)
pairs2( state.x77[,5:8], state.x77[,1:4], panel=panel.smooth)
```

panel.my.symbols *Draw Symbols (User Defined) on a Lattice Plot*

Description

This function draws symbols on a lattice plot. It is similar to the builtin `symbols` function with the difference that it plots symbols defined by the user rather than a prespecified set of symbols.

Usage

```
panel.my.symbols(x, y, symb, inches=1, polygon=FALSE,
                       ..., symb.plots=FALSE, subscripts, MoreArgs)
```

Arguments

`x, y`	The `x` and `y` coordinates for the position of the symbols to be plotted. These can be specified in any way which is accepted by `xy.coords`.
`symb`	Either a matrix, list, or function defining the symbol to be plotted. If it is a matrix or list it needs to be formatted that it can be passed directly to the `llines` function. It then defines the shape of the symbol on on a range/domain of -1 to 1. If this is a function it can either return a matrix or list as above (points on the range/domain of -1 to 1).
`inches`	The size of the square containing the symbol in inches (note: unlike `symbols` this cannot be `FALSE`).
`polygon`	If TRUE, use `lpolygon` function to plot rather than the `llines` function.
`symb.plots`	Currently not implemented.
`...`	Additional arguments will be replicated to the same length as `x` then passed to `symb` (if `symb` is a function) and/or the `lines` function (one value per symbol drawn).
`subscripts`	subscripts for the current panel
`MoreArgs`	A list with any additional arguments to be passed to the `symb` function (as is, without being replicated/split).

Details

The `symb` argument can be a 2 column matrix or a list with components 'x' and 'y' that defines points on the interval [-1,1] that will be connected with lines to draw the symbol. If you want a closed polygon then be sure to replicate the 1st point as the last point or use the `polygon` option.

If any point contains an NA then the line will not be drawn to or from that point. This can be used to create a symbol with disjoint parts that should not be connected.

If `symb` is a function then any unmatched arguments that end up in the '...' argument will be replicated to the same length as 'x' (using the `rep` function) then the values will be passed one at a time to the `symb` function. If `MoreArgs` is specified, the elements of it will also be passed to `symb` without modification. The `symb` function can either return a matrix or list with the points that will then be passed to the `llines` function (see above).

Value

This function is run for its side effect of plotting, it returns an invisible NULL.

Note

Plotting coordinates and sizes are based on the size of the device at the time the function is called. If you resize the device after plotting, all bets are off.

Author(s)

Greg Snow <538280@gmail.com>

See Also

symbols, my.symbols, subplot, mapply, ms.polygram, lines

Examples

```
if(require(lattice)) {
  tmpdf <- data.frame( x=1:10, y=1:10, g=factor(rep( c("A","B"), each=5 )),
z=c(1:5,5:1) )

  xyplot( y ~ x, tmpdf, panel=panel.my.symbols, symb=ms.female, inches=0.3 )

  xyplot( y ~ x | g, tmpdf, panel=panel.my.symbols, symb=ms.male, inches=0.3)

  xyplot( y ~ x, tmpdf, panel=panel.superpose, groups=g,
panel.groups= function(group.number, ...) {
if(group.number==1) {
panel.my.symbols(..., symb=ms.male)
} else {
panel.my.symbols(..., symb=ms.female)
} },
inches=0.3
)

  xyplot( y ~ x, tmpdf, panel=panel.my.symbols, symb=ms.polygram, n=tmpdf$z,
inches=0.3)

  xyplot( y ~ x | g, tmpdf, panel=panel.my.symbols, symb=ms.polygram,
n=tmpdf$z, inches=0.3)

  xyplot( y ~ x, tmpdf, panel=panel.superpose, groups=g,
panel.groups = panel.my.symbols,
inches=0.3, symb=ms.polygon, n=tmpdf$z, polygon=TRUE,
adj=rep(c(0,pi/4),5)
)
}
```

petals *Play the Petals Around the Rose game*

Description

This plays the lateral thinking game Petals Around the Rose. This is a game where 5 regular dice are rolled and the players then try to figure out how many petals are around the rose.

Usage

```
petals(plot = TRUE, txt = TRUE)
```

Arguments

plot	Should the dice be plotted to the current/default graphics device.
txt	Should the dice be shown in the console window using text.

Details

At least one of the arguments plot and txt needs to be true, otherwise you will be guessing blind (or testing your psychic abilities).

The game is usually played with 5 physical dice, one person who knows the rules (the potentate of the rose, here the computer), and one or more players trying to learn the puzzle. The potentate can only give the players the following 3 rules:

1. The name of the game is "Petals Around the Rose" and the name is significant.
2. The answer is always 0 or an even number.
3. The potentate can tell the answer for any roll after any guesses are made.

The potentate (or other player) then rolls the 5 dice and any players are then allowed to guess. The potentate either confirms a correct guess or tells the correct answer, then the game continues with another roll. Players are not to discuss their reasoning so that each can solve it themselves. When a player thinks they have worked out the reasoning they demonstrate it by getting correct guesses, but not by discussing it with anyone. Generally 6 correct guesses in a row is considered evidence that they have figured out the rules and they are then considered a potentate of the rose.

For this implementation the computer will simulate the role of 5 dice and display the results and ask for a guess of how many petals are around the rose. The player then enters their guess and the computer then either confirms that it is correct or gives the correct answer.

Pressing enter without making a guess ends the game.

Value

This function only returns NULL, it is run for its side effects.

Note

Casual viewing of the function source code is unlikely to reveal the secret (and therefore this could be used as an example of one way to disguise portions of code from casual examination). More on disguising source code is at https://stat.ethz.ch/pipermail/r-devel/2011-October/062236.html.

Some basic debugging can reveal the secret, but that would be cheating and an admission that such a simple game has defeated you, so don't do it, just keep playing until you figure it out.

Author(s)

Greg Snow, <538280@gmail.com>

References

http://www.borrett.id.au/computing/petals-bg.htm

See Also

dice

Examples

```
if(interactive()){
  petals()
}
```

---

plot2script *Create a script from the current plot*

Description

This function attempts to create a script that will recreate the current plot (in the graphics window). You can then edit any parts of the script that you want changed and rerun to get the modified plot.

Usage

```
plot2script(file='clipboard')
```

Arguments

file	The filename (the clipboard by default) for the script to create or append to.

Details

This function works with the graphics window and mainly traditional graphics (it may work with lattice or other graphics, but has not really been tested with those).

This function creates a script file (or puts it on the clipboard so that you can past into a script window or text editor) that will recreate the current graph in the current graph window. The script consists of very low level functions (calls to `plot.window` and `axis` rather than letting `plot` handle all this).

If you want the higher level functions that were actually used, then use the `history` or `savehistory` commands (this will probably be the better method for most cases).

Some of the low level plotting functions use different arguments to the internal version than the user callable version (`box` for example), the arguments to these functions may need to be editted before the full script will run correctly.

The lengths of command lines between the creation of the script and what can be run in R do not always match, you may need to manually wrap long lines in the script before it will run properly.

Value

This function is run for its side effects and does not return anything meaningful.

Note

For any serious projects it is best to put your code into a script to begin with and edit the original script rather than using this function.

This function depends on the `recordPlot` function which can change in any version. Therefore this function should not be considered stable.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`history`, `savehistory`, `recordPlot`, `source`

Examples

```
if(interactive()){

# create a plot
plot(runif(10),rnorm(10))
lines( seq(0,1,length=10), rnorm(10,1,3) )

# create the script
plot2script()

# now paste the script into a script window or text processor.
# edit the ranges in plot.window() and change some colors or
# other options.  Then run the script.
 }
```

power.examp *Graphically illustrate the concept of power.*

Description

Create graphs of a normal test statistic under the null and alternative hypotheses to graphically show the idea of power.

Usage

```
power.examp(n = 1, stdev = 1, diff = 1, alpha = 0.05, xmin = -2, xmax = 4)
run.power.examp(hscale=1.5, vscale=1.5, wait=FALSE)
run.power.examp.old()
```

Arguments

n	The sample size for the test statistic.
stdev	The standard deviation of the population.
diff	The true difference in means (alternate hypothesis).
alpha	The type I error rate to use for the test.
xmin	The minimum x value to show on the graph.
xmax	The maximum x value to show on the graph.
hscale	Controls width of plot, passed to tkrplot.
vscale	Controls height of plot, passed to tkrplot.
wait	Should R wait for the window to close.

Details

This function will draw 2 graphs representing an upper-tailed test of hypothesis.

The upper panel represents the test statistic under the null hypothesis that the true mean (or mean difference) is 0. It then also shows the upper tail area equal to alpha and the rejection region for the test statistic.

The lower panel shows the normal distribution for the test statistic under the alternative hypothesis where the true mean (or mean difference) is diff. Using the rejection region from the upper panel it shades the upper tail area that corresponds to the power of the test.

Both curves are affected by the specified stdev and sample size n.

The function run.power.examp will in addition create a Tk slider box that will allow you to interactively change the values of stdev, diff, alpha, and n to dynamically see the effects of the change on the graphs and on the power of the test.

This can be used to demonstrate the concept of power, show the effect of sample size on power, show the inverse relationship between the type I and type II error rates, and show how power is dependent on the true mean (or difference) and the population standard deviation.

Value

`power.examp` invisibly returns the power computed.

`run.power.examp` returns a list with the parameter settings and the power if `wait` is TRUE.

`run.power.examp.old` does not return anything meaningful.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`power.t.test`

Examples

```
power.examp()
power.examp(n=25)
power.examp(alpha=0.1)
```

---

put.points.demo *Demonstrate Correlation and Regression by placing and moving data points*

Description

Place data points on a graph to demonstrate concepts related to correlation and regression.

Usage

```
put.points.demo(x = NULL, y = NULL, lsline = TRUE)
```

Arguments

`x`	x-coordinates for initial points.
`y`	y-coordinates for initial points.
`lsline`	Logical, should the ls regresion line be included.

Details

The plot area is divided into 2 sections, the left section shows a scatterplot of your points, the right panel controls what happens when you click in the left panel.

The top of the right panel has an "end" button that you click on to end the demonstration.

The middle right panel toggles the least squares line and information.

The bottom right panel has radio buttons that determine what clicking in the left panel will do, the options are to add a point, delete a point, or move a point.

To move a point click on the point you want to move, it will become solid, then click in the place you want it to move to.

When deleting or moving points, the closest point to where you click will be deleted or moved, even if you click in an empty area.

Whenever you add, delete, or move a point the correlation, r^2, and regression line will be updated. You can start with a set of points then demonstrate what happens to the correlation and regression line when outliers are added or important points are moved or deleted.

Value

This function does not return anything.

Author(s)

Greg Snow <538280@gmail.com>

See Also

plot, cor

Examples

```
if(interactive()){
put.points.demo()

x <- rnorm(25, 5, 1)
y <- x + rnorm(25)
put.points.demo(x,y)
}
```

Pvalue.norm.sim *Simulate P-values*

Description

Simulate and plot p-values from a normal or binomial based test under various conditions. When all the assumptions are true, the p-values should follow an approximate uniform distribution. These functions show that along with how violating the assumptions changes the distribution of the p-values.

Usage

```
Pvalue.norm.sim(n = 50, mu = 0, mu0 = 0, sigma = 1, sigma0 = sigma,
     test= c("z", "t"), alternative = c("two.sided", "less", "greater", "<>",
           "!=", "<", ">"), alpha = 0.05, B = 10000)
Pvalue.binom.sim(n=100, p=0.5, p0=0.5, test=c('exact','approx'),
                           alternative=c('two.sided', 'less', 'greater',
                           '<>','!=','<','>'),
                           alpha=0.05, B=1000)
run.Pvalue.norm.sim()
run.Pvalue.binom.sim()
```

Arguments

n	Sample Size for each simulated dataset
mu	Simulation mean for samples
mu0	Hypothesized mean for tests
sigma	Simulation SD for samples
sigma0	Hypothesized SD for tests, if blank or missing, use the sample SD in the tests
p	Simulation proportion for samples
p0	Hypothesized proportion for tests
test	Which test to use, "z" or "t" tests for normal, "exact" (binomial) or "approx" (normal approximation) for binomial
alternative	Direction for alternative hypothesis
alpha	alpha level for test (optional)
B	Number of simulated datasets

Details

These functions generate B samples from either a normal or binomial distribution, then compute the P-values for the test of significance on each sample and plot the P-values.

The run.Pvalue.norm.sim and run.Pvalue.binom.sim functions are GUI wrappers for the other 2 functions allowing you to change the parameters and click on "refresh" to run a new set of simulations.

Using NA for sigma0 will result in the sample standard deviations being used (leave blank in the GUI).

When the simulation conditions and the hypothesized values match, the distributions of the p-values will be approximately uniform. Changing the parameter of interest will show the idea of power. Changing the other parameters can show the effects of assumptions not being met.

Value

The P-values are invisibly returned.

Note

Note: the 2-sided p-values for the binomial may not match the results from binom.test and prop.test. The method used here is an approximation for speed.

Author(s)

Greg Snow, <538280@gmail.com>

References

Murdock, D, Tsai, Y, and Adcock, J (2008) _P-Values are Random Variables_. The American Statistician. (62) 242-245.

See Also

t.test, z.test, binom.test, prop.test, tkexamp

Examples

```
 if(interactive()) {
  run.Pvalue.norm.sim()
  run.Pvalue.binom.sim()
}
```

---

rgl.coin *Animated die roll or coin flip*

Description

Open an rgl window, plot either a representation of a coin or a die then animate the flipping/rolling.

Usage

```
rgl.coin(x, col = "black", heads = x[[1]], tails = x[[2]], ...)

rgl.die(x=1:6, col.cube = "white", col.pip = "black", sides = x, ...)

flip.rgl.coin(side = sample(2, 1), steps = 150)

roll.rgl.die(side = sample(6, 1), steps = 250)
```

Arguments

x	for rgl.coin a list with information for drawing the faces of the coin, defaults to coin.faces. For rgl.die a vector with the number of pips to put on the sides of the die (alternative way of specifying sides).
col	Color of lines on the coin faces.

heads	Design to use as "heads" side of coin.
tails	Design to use as "tails" side of coin.
col.cube	Color of the cube for the die.
col.pip	Color of the pips (spots) on the die
sides	Vector of length 6 indicating which numbers to show on the die.
side	Which side of the coin (1 or 2) or die (1 through 6) should end up face up.
steps	The number of steps in each part of the animation, higher values will be smoother and slower, lower values will be faster but more jumpy.
...	Currently any additional options are silently ignored.

Details

You must use the plot function first to create the coin or die, then use the flip or roll function to start the animation. You can animate multiple times for a single use of the plotting function.

You can manually rotate the image as well, see the rgl package for details.

The defaults plot a regular coin and die, but arguments are available to create special casses (2 headed coin, die with 2 6's and no 1, ...).

The data list coin.faces contains information on designs for the faces of the coins in case you want to choose a different design.

The default rolling and flipping options ranomly choose which side will be face up following a uniform distribution. You can specify the side yourself, or use the sample function to do a biased random flip/roll.

Value

Which side ended up face up (1 or 2 for coin, 1 through 6 for die). This is the internal numbering and does not match a change in the sides argument.

Note

The current algorithm for animating the die roll shows all the sides, but I am not satisfied with it. Please suggest improvements.

Author(s)

Greg Snow <538280@gmail.com>

See Also

dice, plot.dice, coin.faces, sample

Examples

```
if(interactive()){
rgl.coin()
flip.rgl.coin()
flip.rgl.coin(1)
flip.rgl.coin(2)

rgl.clear()

# two-headed coin
rgl.coin(tails=coin.faces$qh)

rgl.clear()

# letters instead of pictures
rgl.coin(heads=coin.faces$H, tails=coin.faces$T)

# biased flip
flip.rgl.coin( sample(2,1, prob=c(0.65, 0.35) ) )

rgl.clear()

rgl.die()
roll.rgl.die()
roll.rgl.die(6)

# biased roll
roll.rgl.die( sample(6,1, prob=c(1,2,3,3,2,1) ) )
}
```

`rgl.Map` *Plot a map in an rgl window*

Description

Plots a map (from a Map object from package spData) on a unit sphere in an rgl window that can then be interactively rotated.

Usage

```
rgl.Map(Map, which, ...)
```

Arguments

`Map`	An `sfc_MULTIPOLYGON` object
`which`	Vector indicating the subset of polygons to plot.
`...`	Additional arguments passed on to `rgl.lines`.

Details

This assumes that the map is cordinates in degrees and plots the map on a unit sphere in an rgl window making a globe. You can then rotate the globe by clicking and dragging in the window.

Value

There is no return value, this function is run for its side effect.

Note

This function is still beta level software (some extra lines show up).

Author(s)

Greg Snow <538280@gmail.com>

See Also

`rgl` in package rgl, `plot` method in package sp

Examples

```
if(interactive()){

if(require("spData")) {
  data(world)
  rgl.Map(world$geom)
  spheres3d(0,0,0,.999, col='lightblue')
}
}
```

---

roc.demo *Demonstrate ROC curves by interactively building one*

Description

This demonstration allows you to interactively build a Receiver Operator Curve to better understand what goes into creating them.

Usage

```
roc.demo(x = rnorm(25, 10, 1), y = rnorm(25, 11, 1.5))
```

Arguments

x	Data values for group 1 (controls).
y	Data values for group 2 (cases).

Details

Density plots for the 2 groups will be created in the lower panel of the plot (colored red (group 1) and blue (group 2)) along with rug plots of the actual datapoints. There is also a green vertical line that represents a decision rule cutoff, any points higher than the cutoff are predicted to be in group 2 and points less than the cuttoff are predicted to be in group 1. The sensitivity and specificity for the current cuttoff value are printed below the plot.

A Tk slider box is also created that allows you to move the cuttoff value and update the plots. As the cutoff value changes, the different combinations of sensitivity and specificity are added to the ROC curve in the top panel (the point corresponding to the current cuttoff value is highlighted in red). A line is also drawn from the point representing sensitivity and specificity both equal to 1 to the point closest to it.

Value

No meaninful value is returned, this function is run solely for the side effects.

Author(s)

Greg Snow <538280@gmail.com>

See Also

slider, ROC function in package Epi, auROC in package limma, package ROC

Examples

```
if(interactive()){
roc.demo()
with(CO2,
  roc.demo(uptake[Type=='Mississippi'],
           uptake[Type=='Quebec']       )
)
}
```

`rotate.cloud` *Interactively rotate 3D plots*

Description

Interactively rotate common 3d plots: cloud, persp, and wireframe.

Usage

```
rotate.cloud(x, ...)
rotate.persp(x, y, z)
rotate.wireframe(x, ...)
```

Arguments

x	x, see persp, or formula/matrix to pass to cloud or wireframe
y	y, see persp
z	z, see persp
...	additional arguments passed on to cloud or persp

Details

Use these functions just like cloud, persp, and wireframe. In addition to the default plot a Tk slider window will be created that will allow you to rotate the plot.

The rotations parameters are passed the screen argument of cloud and wireframe and the theta, phi, r, d, ltheta, lphi, and shade arguments of persp.

For cloud and wireframe plots the order of the x, y, and z argumets can be rearanged, just type the appropriate letters in the boxes on the left, then press the "refresh" button (changing the order changes the plot for these 2 plots).

Value

These functions are run for the side effects of the plots and Tk windows, nothing meaninful is returned.

Author(s)

Greg Snow <538280@gmail.com>

See Also

cloud it the lattice package, persp, wireframe in the lattice package

Examples

```
if(interactive()){
rotate.cloud(Sepal.Length ~ Petal.Length*Petal.Width, data=iris)

rotate.wireframe(volcano)

z <- 2 * volcano        # Exaggerate the relief
x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
rotate.persp(x,y,z)
}
```

`run.cor.examp` *Interactively demonstrate correlations*

Description

Make a scatterplot and a Tk slider window that allows you to interactively set the correlation and/or R^2.

Usage

```
run.cor.examp(n=100, seed, vscale=1.5, hscale=1.5, wait=FALSE)
run.cor2.examp(n=100, seed, vscale=1.5, hscale=1.5, wait=FALSE)
run.old.cor.examp(n = 100, seed)
run.old.cor2.examp(n = 100, seed)
```

Arguments

`n`	Number of points to plot.
`seed`	What seed to use.
`vscale`	Vertical scale passed to tkrplot.
`hscale`	Horizontal scale passed to tkrplot.
`wait`	Should R wait for the tk window to close.

Details

The function `run.cor.examp` draws a scatterplot and allows you to set the correlation using a Tk slider window.

The function `run.cor2.examp` does the same, but has a slider for R^2 as well as the correlation, when either slider is moved the other one will update to match.

The 2 "old" versions use the default graphics device with a seperate window with the sliders, the versions without "old" in the name include the plot and sliders together in a single tk window.

The size of the plot can be changed by changing the values in the hscale and vscale boxes and clicking on the "Refresh" button.

Value

If `wait` is TRUE, then the return value is a list with the x and y values of the final plot.

If `wait` is FALSE (and in the "old" versions) an invisible NULL is returned.

Note

If `wait` is TRUE then R will wait until you click on the "Exit" button before you can use your R session again. If `wait` is FALSE then the tk window will appear, but R will regain control so that you can continue to use R as well as interact with the demonstration window.

Author(s)

Greg Snow <538280@gmail.com>

See Also

cor, tkexamp

Examples

```
if(interactive()) {
  run.cor2.examp()
}
```

`run.hist.demo` *Create a histogram and interactively change the number of bars.*

Description

Create a histogram then use a Tk slider window to change the number of bars, the minimum, and the maximum.

Usage

```
run.hist.demo(x)
```

Arguments

x	Data to plot.

Details

Draws a histogram and creates a Tk slider window that allows you to explore how changing the parameters affects the appearance of the plot.

Value

No meaninful value is returned.

Author(s)

Greg Snow <538280@gmail.com>

See Also

hist, slider

Examples

```
if(interactive()){
run.hist.demo( rnorm(250, 100, 5) )
}
```

SensSpec.demo *Demonstrate Sensitivity, Specificity, PPV, and NPV*

Description

This function demonstrates how to get PPV and NPV from Sensitivity, Specificity, and Prevalence by using a virtual population rather than a direct application of Bayes Rule. This approach is more intuitive to mathphobes.

Usage

```
SensSpec.demo(sens, spec, prev, n = 100000, step = 11)
```

Arguments

sens	Sensitivity (between 0 and 1)
spec	Specificity (between 0 and 1)
prev	Prevalence (between 0 and 1)
n	Size of the virtual population (large round number)
step	which step of the process to display

Details

The common way to compute Positive Predictive Value (probability of disease given a positive test (PPV)) and Negative Predictive Value (probability of no disease given negative test (NPV)) is to use Bayes' rule with the Sensitivity, Specificity, and Prevalence.

This approach can be overwhelming to non-math types, so this demonstration goes through the steps of assuming a virtual population, then filling in a 2x2 table based on the population and given values of Sensitivity, Specificity, and Prevalence. PPV and NPV are then computed from this table. This approach is more intuitive to many people.

The function can be run multiple times with different values of `step` to show the steps in building the table, then rerun with different values to show how changes in the inputs affect the results.

Value

An invisible matrix with the 2x2 table

Author(s)

Greg Snow, <538280@gmail.com>

See Also

roc.demo, fagan.plot, the various Epi packages, tkexamp

Examples

```
for(i in seq(1,11,2)) {
  SensSpec.demo(sens=0.95, spec=0.99, prev=0.01, step=i)
  if( interactive() ) {
    readline("Press Enter to continue")
  }
}
```

---

shadowtext *Add text to a plot with a contrasting background.*

Description

This is similar to the text function, but it also puts a background shadow (outline) behind the text to make it stand out from the background better.

Usage

```
shadowtext(x, y = NULL, labels, col = "white", bg = "black",
  theta = seq(pi/32, 2 * pi, length.out = 64), r = 0.1,
  cex = 1, ...)
```

Arguments

x	x-coordinates for the text
y	y-coordinates for the text
labels	The text labels to plot
col	Color of the text
bg	Color of the background shadow
theta	Angles for plotting the background
r	Thickness of the shadow relative to plotting size
cex	Character expansion passed through to text and used in computing text size.
...	Additional arguments passed on to text

Details

When adding text to a plot it is possible that the color of the text may make it difficult to see relative to its background. If the text spans different backgrounds then it may not be possible to find a single color to give proper contrast.

This function creates a contrasting shadow for the text by first plotting several copies of the text at angles theta and distance r in the background color, then plotting the text on top.

This gives a shadowing or outlining effect to the text making it easier to read on any background.

Value

This function is run for its side effects, returns NULL.

Author(s)

Greg Snow, <538280@gmail.com>, with improvements by Thomas Danhorn

See Also

text

Examples

```
plot(1:10, 1:10, bg='aliceblue')
rect(3,3,5,8, col='navy')
text(5,6, 'Test 1', col='lightsteelblue')
shadowtext(5,4, 'Test 2', col='lightsteelblue')
```

---

sigma.test *One sample Chi-square test for a population variance*

Description

Compute the test of hypothesis and compute a confidence interval on the variance of a population.

Usage

```
sigma.test(x, sigma = 1, sigmasq = sigma^2,
  alternative = c("two.sided", "less", "greater"), conf.level = 0.95, ...)
```

Arguments

x	Vector of data values.
sigma	Hypothesized standard deviation of the population.
sigmasq	Hypothesized variance of the population.
alternative	Direction of the alternative hypothesis.
conf.level	Confidence level for the interval computation.
...	Additional arguments are silently ignored.

Details

Many introductory statistical texts discuss inference on a single population variance and introduce the chi-square test for a population variance as another example of a hypothesis test that can be easily derived. Most statistical packages do not include the chi-square test, perhaps because it is not used in practice very often, or because the test is known to be highly sensitive to nonnormal data. For the two-sample problem, see var.test.

Value

An object of class `htest` containing the results

Note

This test is highly sensitive to nonnormality.

Author(s)

G. Jay Kerns <gkerns@ysu.edu>

See Also

`var.test`, `print.htest`

Examples

```
x <- rnorm(20, mean = 15, sd = 7)
sigma.test(x, sigma = 6)
```

`simfun` *Create a function to simulate data*

Description

This function is used to create a new function that will simulate data. This could be used by a teacher to create homework or test conditions that the students would then simulate data from (each student could have their own unique data set) or this function could be used in simulations for power or other values of interest.

Usage

```
simfun(expr, drop, ...)
```

Arguments

`expr`	This is an expression, usually just one or more statements, that will generate the simulated data.
`drop`	A character vector of names of objects/columns that will be dropped from the return value. These are usually intermediate objects or parameter values that you don't want carried into the final returned object.
`...`	Additional named items that will be in the environment when `expr` is evaluated.

Details

This function creates another function to simulate data. You supply the general ideas of the simulation to this function and the resulting function can then be used to create simulated datasets. The resulting function can then be given to students for them to simulate datasets, or used localy as part of larger simulations.

The environment where the expression is evaluated will have all the columns or elements of the `data` argument available as well as the `data` argument itself. Any variables/parameters passed through `...` in the original function will also be available. You then supply the code based on those variables to create the simulated data. The names of any columns or parameters submitted as part of `data` will need to match the code exactly (provide specific directions to the users on what columns need to be named). Rember that indexing using factors indexes based on the underlying integers not the character representation. See the examples for details.

The resulting function can be saved and loaded/attached in different R sessions (it is important to use `save` rather than something like `dput` so that the environment of the function is preserved).

The function includes an optional seed that will be used with the `char2seed` function (if the seed is a character) so that each student could use a unique but identifiable seed (such as their name or something based on their name) so that each student will use a different dataset, but the instructor will be able to generate the exact same dataset to check answers.

The "True" parameters are hidden in the environment of the function so the student will not see the "true" values by simply printing the function. However an intermediate level R programmer/user would be able to extract the simulation parameters (but the correct homework or test answer will not be the simulation parameters).

Value

The return value is a function that will generate simulated datasets. The function will have 2 arguments, `data` and `seed`. The `data` argument can be either a data frame of the predictor variables (study design) or a list of simulation parameters. The `seed` argument will be passed on to `set.seed` if it is numeric and `char2seed` if it is a character.

The return value of this function is a dataframe with the simulated data and any explanitory variables passed to the function.

See the examples for how to use the result function.

Note

This function was not designed for speed, if you are doing long simulations then hand crafting the simulation function will probably run quicker than one created using this function.

Like the prediction functions the data frame passed in as the data argument will need to have exact names of the columns to match with the code (including capitolization).

This function is different from the `simulate` functions in that it allows for different sample sizes, user specified parameters, and different predictor variables.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

set.seed, char2seed, within, simulate, save, load, attach

Examples

```
# Create a function to simulate heights for a given dataset

simheight <- simfun( {h <- c(64,69); height<-h[sex]+ rnorm(10,0,3)}, drop='h' )

my.df <- data.frame(sex=factor(rep(c('Male','Female'),each=5)))
simdat <- simheight(my.df)
t.test(height~sex, data=simdat)

# a more general version, and have the expression predefined
# (note that this assumes that the levels are Female, Male in that order)

myexpr <- quote({
  n <- length(sex)
  h <- c(64,69)
  height <- h[sex] + rnorm(n,0,3)
})

simheight <- simfun(eval(myexpr), drop=c('n','h'))
my.df <- data.frame(sex=factor(sample(rep(c('Male','Female'),c(5,10)))))
(simdat <- simheight(my.df))

# similar to above, but use named parameter vector and index by names

myexpr <- quote({
  n <- length(sex)
  height <- h[ as.character(sex)] + rnorm(n,0,sig)
})

simheight <- simfun(eval(myexpr), drop=c('n','h','sig'),
  h=c(Male=69,Female=64), sig=3)
my.df <- data.frame(sex=factor(sample(c('Male','Female'),100, replace=TRUE)))
(simdat <- simheight(my.df, seed='example'))

# Create a function to simulate Sex and Height for a given sample size
# (actually it will generate n males and n females for a total of 2*n samples)
# then use it in a set of simulations
simheight <- simfun( {sex <- factor(rep(c('Male','Female'),each=n))
                      height <- h[sex] + rnorm(2*n,0,s)
                      }, drop=c('h','n'), h=c(64,69), s=3)
(simdat <- simheight(list(n=10)))

out5  <- replicate(1000, t.test(height~sex, data=simheight(list(n= 5)))$p.value)
out15 <- replicate(1000, t.test(height~sex, data=simheight(list(n=15)))$p.value)

mean(out5  <= 0.05)
mean(out15 <= 0.05)
```

```
# use a fixed population

simstate <- simfun({
  tmp <- state.df[as.character(State),]
  Population <- tmp[['Population']]
  Income <- tmp[['Income']]
  Illiteracy <- tmp[['Illiteracy']]
}, state.df=as.data.frame(state.x77), drop=c('tmp','state.df'))
simstate(data.frame(State=sample(state.name,10)))

# Use simulation, but override setting the seed

simheight <- simfun({
  set.seed(1234)
  h <- c(64,69)
  sex <- factor(rep(c('Female','Male'),each=50))
  height <- round(rnorm(100, rep(h,each=50),3),1)
  sex <- sex[ID]
  height <- height[ID]
}, drop='h')
(newdat <- simheight(list(ID=c(1:5,51:55))))
(newdat2<- simheight(list(ID=1:10)))

# Using a fitted object

fit <- lm(Fertility ~ . , data=swiss)
simfert <- simfun({
  Fertility <- predict(fit, newdata=data)
  Fertility <- Fertility + rnorm(length(Fertility),0,summary(fit)$sigma)
}, drop=c('fit'), fit=fit)

tmpdat <- as.data.frame(lapply(swiss[,-1],
            function(x) round(runif(100, min(x), max(x)))))
names(tmpdat) <- names(swiss)[-1]
fertdat <- simfert(tmpdat)
head(fertdat)
rbind(coef(fit), coef(lm(Fertility~., data=fertdat)))

# simulate a nested mixed effects model
simheight <- simfun({
  n.city <- length(unique(city))
  n.state <- length(unique(state))
  n <- length(city)
  height <- h[sex] + rnorm(n.state,0,sig.state)[state] +
    rnorm(n.city,0,sig.city)[city] + rnorm(n,0,sig.e)
}, sig.state=1, sig.city=0.5, sig.e=3, h=c(64,69),
  drop=c('sig.state','sig.city','sig.e','h','n.city','n.state','n'))

tmpdat <- data.frame(state=gl(5,20), city=gl(10,10),
  sex=gl(2,5,length=100, labels=c('F','M')))
heightdat <- simheight(tmpdat)

# similar to above, but include cost information, this assumes that
```

```
# each new state costs $100, each new city is $10, and each subject is $1
# this shows 2 possible methods

simheight <- simfun({
  n.city <- length(unique(city))
  n.state <- length(unique(state))
  n <- length(city)
  height <- h[sex] + rnorm(n.state,0,sig.state)[state] +
    rnorm(n.city,0,sig.city)[city] + rnorm(n,0,sig.e)
  cost <- 100 * (!duplicated(state)) + 10*(!duplicated(city)) + 1
  cat('The total cost for this design is $', 100*n.state+10*n.city+1*n,
  '\n', sep='')
}, sig.state=1, sig.city=0.5, sig.e=3, h=c(64,69),
  drop=c('sig.state','sig.city','sig.e','h','n.city','n.state','n'))

tmpdat <- data.frame(state=gl(5,20), city=gl(10,10),
  sex=gl(2,5,length=100, labels=c('F','M')))
heightdat <- simheight(tmpdat)
sum(heightdat$cost)


# another mixed model method

simheight <- simfun({
  state <- gl(n.state, n/n.state)
  city <- gl(n.city*n.state, n/n.city/n.state)
  sex <- gl(2, n.city, length=n, labels=c('F','M') )
  height <- h[sex] + rnorm(n.state,0,sig.state)[state] +
    rnorm(n.city*n.state,0,sig.city)[city] + rnorm(n,0,sig.e)
}, drop=c('n.state','n.city','n','sig.city','sig.state','sig.e','h'))

heightdat <- simheight( list(
  n.state=5, n.city=2, n=100, sig.state=10, sig.city=3, sig.e=1, h=c(64,69)
))
```

`slider` *slider / button control widgets*

Description

`slider` constructs a Tcl/Tk-widget with sliders and buttons automated calculation and plotting. For example slider allows complete all axes rotation of objects in a plot.

Usage

```
slider(sl.functions, sl.names, sl.mins, sl.maxs, sl.deltas, sl.defaults,
but.functions, but.names, no, set.no.value, obj.name, obj.value,
reset.function, title)
```

Arguments

sl.functions	set of functions or function connected to the slider(s)
sl.names	labels of the sliders
sl.mins	minimum values of the sliders' ranges
sl.maxs	maximum values of the sliders' ranges
sl.deltas	change of step per click
sl.defaults	default values for the sliders
but.functions	function or list of functions that are assigned to the button(s)
but.names	labels of the buttons
no	slider(no=i) requests slider i
set.no.value	slider(set.no.value=c(i,val)) sets slider i to value val
obj.name	slider(obj.name=name) requests the value of variable name from environment slider.env
obj.value	slider(obj.name=name,obj.value=value) assigns value to variable name in environment slider.env
reset.function	function that comprises the commands of the reset.button
title	title of the control window

Details

With slider you can: a. define (multiple) sliders and buttons, b. request or set slider values, and c. request or set variables in the environment slider.env. Slider function management takes place in the environment slider.env. If slider.env is not found it is generated.

Definition: ... of sliders: First of all you have to define sliders, buttons and the attributes of them. Sliders are established by six arguments: sl.functions, sl.names, sl.minima, sl.maxima,sl.deltas, and sl.defaults. The first argument, sl.functions, is either a list of functions or a single function that entails the commands for the sliders. If there are three sliders and slider 2 is moved with the mouse the function stored in sl.functions[[2]] (or in case of one function for all sliders the function sl.functions) is called.

Definition: ... of buttons: Buttons are defined by a vector of labels but.names and a list of functions: but.functions. If button i is pressed the function stored in but.functions[[i]] is called.

Requesting: ... a slider: slider(no=1) returns the actual value of slider 1, slider(no=2) returns the value of slider 2, etc. You are allowed to include expressions of the type slider(no=i) in functions describing the effect of sliders or buttons.

Setting: ... a slider: slider(set.no.value=c(2,333)) sets slider 2 to value 333. slider(set.no.value=c(i,value)) can be included in the functions defining the effects of moving sliders or pushing buttons.

Variables: ... of the environment slider.env: Sometimes information has to be trransferred back and forth between functions defining the effects of sliders and buttons. Imagine for example two sliders: one to control p and another one to control q, but they should satisfy: p+q=1. Consequently, you have to correct the value of the first slider after the second one was moved. To prevent the creation of global variables store them in the environment slider.env. Use slider(obj.name="p.save",obj.value=1-slide to assign value 1-slider(no=2) to the variable p.save . slider(obj.name=p.save) returns the value of variable p.save.

Value

Using `slider` in definition mode `slider` returns the value of new created the top level widget. `slider(no=i)` returns the actual value of slider `i`. `slider(obj.name=name)` returns the value of variable `name` in environment `slider.env`.

Note

You can move the slider in 3 different ways: You can left click and drag the slider itself, you can left click in the trough to either side of the slider and the slider will move 1 unit in the direction you clicked, or you can right click in the trough and the slider will jump to the location you clicked at.

This function may not stay in this package (consider it semi-depricated), the original of the slider function is in the relax package and can be used from there. In TeachingDemos the `tkexamp` function is taking the place of `slider` and gives a possibly more general approach.

Author(s)

Hans Peter Wolf

See Also

`tkexamp`, `sliderv`

Examples

```
# example 1, sliders only
## Not run:
## This example cannot be run by examples() but should work in an interactive R session
plot.sample.norm<-function(){
 refresh.code<-function(...){
   mu<-slider(no=1); sd<-slider(no=1); n<-slider(no=3)
   x<-rnorm(n,mu,sd)
   plot(x)
 }
 slider(refresh.code,sl.names=c("value of mu","value of sd","n number of observations"),
     sl.mins=c(-10,.01,5),sl.maxs=c(+10,50,100),sl.deltas=c(.01,.01,1),sl.defaults=c(0,1,20))
}
plot.sample.norm()

## End(Not run)

# example 2, sliders and buttons
## Not run:
## This example cannot be run by examples() but should work in an interactive R session
plot.sample.norm.2<-function(){
 refresh.code<-function(...){
   mu<-slider(no=1); sd<-slider(no=2); n<-slider(no=3)
   type=  slider(obj.name="type")
   x<-rnorm(n,mu,sd)
   plot(seq(x),x,ylim=c(-20,20),type=type)
 }
```

```
 slider(refresh.code,sl.names=c("value of mu","value of sd","n number of observations"),
     sl.mins=c(-10,.01,5),sl.maxs=c(10,10,100),sl.deltas=c(.01,.01,1),sl.defaults=c(0,1,20),
       but.functions=list(
              function(...){slider(obj.name="type",obj.value="l");refresh.code()},
              function(...){slider(obj.name="type",obj.value="p");refresh.code()},
              function(...){slider(obj.name="type",obj.value="b");refresh.code()}
       ),
       but.names=c("lines","points","both"))
  slider(obj.name="type",obj.value="l")
}
plot.sample.norm.2()

## End(Not run)

# example 3, dependent sliders
## Not run:
## This example cannot be run by examples() but should work in an interactive R session
print.of.p.and.q<-function(){
 refresh.code<-function(...){
   p.old<-slider(obj.name="p.old")
   p<-slider(no=1); if(abs(p-p.old)>0.001) {slider(set.no.value=c(2,1-p))}
   q<-slider(no=2); if(abs(q-(1-p))>0.001) {slider(set.no.value=c(1,1-q))}
   slider(obj.name="p.old",obj.value=p)
   cat("p=",p,"q=",1-p,"\n")
 }
 slider(refresh.code,sl.names=c("value of p","value of q"),
       sl.mins=c(0,0),sl.maxs=c(1,1),sl.deltas=c(.01,.01),sl.defaults=c(.2,.8))
 slider(obj.name="p.old",obj.value=slider(no=1))
}
print.of.p.and.q()

## End(Not run)

# example 4, rotating a surface
## Not run:
## This example cannot be run by examples() but should work in an interactive R session
R.veil.in.the.wind<-function(){
  # Mark Hempelmann / Peter Wolf
  par(bg="blue4", col="white", col.main="white",
      col.sub="white", font.sub=2, fg="white") # set colors and fonts
  samp  <- function(N,D) N*(1/4+D)/(1/4+D*N)
  z<-outer(seq(1, 800, by=10), seq(.0025, 0.2, .0025)^2/1.96^2, samp) # create 3d matrix
  h<-100
  z[10:70,20:25]<-z[10:70,20:25]+h; z[65:70,26:45]<-z[65:70,26:45]+h
  z[64:45,43:48]<-z[64:45,43:48]+h; z[44:39,26:45]<-z[44:39,26:45]+h
  x<-26:59; y<-11:38; zz<-outer(x,y,"+"); zz<-zz*(65<zz)*(zz<73)
  cz<-10+col(zz)[zz>0];rz<-25+row(zz)[zz>0]; z[cbind(cz,rz)]<-z[cbind(cz,rz)]+h
  refresh.code<-function(...){
    theta<-slider(no=1); phi<-slider(no=2)
    persp(x=seq(1,800,by=10),y=seq(.0025,0.2,.0025),z=z,theta=theta,phi=phi,
          scale=T, shade=.9, box=F, ltheta = 45,
          lphi = 45, col="aquamarine", border="NA",ticktype="detailed")
  }
```

```
  slider(refresh.code, c("theta", "phi"), c(0, 0),c(360, 360),c(.2, .2),c(85, 270)  )
}
R.veil.in.the.wind()

## End(Not run)
```

`sliderv` *Create a Tk slider window*

Description

Create a Tk slider window with the sliders positioned vertically instead of horizontally.

Usage

```
sliderv(refresh.code, names, minima, maxima, resolutions, starts,
  title = "control", no = 0, set.no.value = 0)
```

Arguments

`refresh.code`	Function to be called when sliders are moved.
`names`	Labels for the sliders.
`minima`	Vector of minimum values for the sliders.
`maxima`	Vector of maximum values for the sliders.
`resolutions`	Vector of resolutions for the sliders.
`starts`	Vector of starting values for the sliders.
`title`	Title to put at the top of the Tk box.
`no`	The number of the slider whose value you want.
`set.no.value`	Vector of length 2 with the number of slider to set and the new value.

Details

This is a variation on the `slider` function with vertical sliders arranged in a row rather than horizontal sliders arranged in a column.

This is based on an early version of `slider` and therefore does not have as many bells and whistles (but sometimes fits the screen better).

Value

Returns the value of a given slider when used as: `slider(no=i)`.

Note

You can move the slider in 3 different ways: You can left click and drag the slider itself, you can left click in the trough to either side of the slider and the slider will move 1 unit in the direction you clicked, or you can right click in the trough and the slider will jump to the location you clicked at.

This function may not stay in this package (consider it semi-depricated). See the tkexamp function for another approach to do the same thing.

Author(s)

Greg Snow <538280@gmail.com>

See Also

tkexamp, slider

Examples

```
if(interactive()){
face.refresh <- function(...){
vals <- sapply(1:15, function(x) slider(no=x))
faces( rbind(0, vals, 1), scale=F)
}

sliderv( face.refresh, as.character(1:15), rep(0,15), rep(1,15),
rep(0.05, 15), rep(0.5,15), title='Face Demo')
}
```

SnowsCorrectlySizedButOtherwiseUselessTestOfAnything

Snow's Correctly Sized But Otherwise Useless Test of Anything

---

Description

This is a hypothesis test designed to be correctly sized in that the probability of rejecting the null when it is true will be equal to your alpha level. Other than that it is a pretty useless test mainly intended for when people say something like "I just need a p-value".

Usage

```
SnowsCorrectlySizedButOtherwiseUselessTestOfAnything(x,
  data.name = deparse(substitute(x)),
  alternative = "You Are Lucky", ..., seed)
```

Arguments

x	The data, or nothing, or something equally irrelevant
data.name	The name of the data for the output
alternative	The phrase for the alternate hypothesis in the output
...	Additional arguments that will be silently ignored (like x), future versions may mockingly ignore these instead
seed	A seed (numeric or character) used to seed the random number generator. Use this or manually set the seed if you want reproducible (but still meaningless) results

Details

Some of the advantages/disadvantages of this test include:

- The probability of a Type I error is alpha
- Power can be easily computed (it is alpha)
- Power is independent of the sample size
- Power is independent of the hypotheses
- This test is not affected by missing data (present data either)
- This test does not depend on any distributional or independence assumptions

Value

An object of class htest with the following elements:

p.value	The p-value
statistic	The test statistic (identical to the p-value)
data.name	The name of the data (if any)
method	The name of the test
alternative	a phrase representing the alternative hypothesis
seed	optionally the seed that was used

Note

If someone has suggested that you consider this test, they most likely do not intend for you to actually use the test, rather to reconsider your question or the assumptions that you are making or trying to avoid. This test should only be used to illustrate a point and decisions (other than maybe who should pay for lunch) should never be made based on the results of this test.

Author(s)

Greg Snow <538280@gmail.com>

References

The author is unlikely to be willing to publish in any "journal" that would be willing to publish this test.

fortune(264)

See Also

runif

Examples

```
SnowsCorrectlySizedButOtherwiseUselessTestOfAnything(log(rnorm(100)))
```

SnowsPenultimateNormalityTest

Test the uninteresting question of whether the data represents an exact normal distribution.

Description

This function tests the null hypothesis that the data comes from an exact normal population. This is a much less interesting/useful null than what people usually want, which is to know if the data come from a distribution that is similar enough to the normal to use normal theory inference.

Usage

```
SnowsPenultimateNormalityTest(x)
```

Arguments

x	The data

Details

The theory for this test is based on the probability of getting a rational number from a truly continuous distribution defined on the reals.

The main goal of this test is to quickly give a p-value for those that feel it necessary to test the uninteresting and uninformative null hypothesis that the data represents an exact normal, and allows the user to then move on to much more important questions, like "is the data close enough to the normal to use normal theory inference?".

After running this test (or better instead of running this and any other test of normality) you should ask yourself what it means to test for normality and why you would want to do so. Then plot the data and explore the interesting/useful questions.

Value

An object of class "htest" with components:

p.value	The p-value
alternative	a string representing the alternative hypothesis
method	a string describing the method
data.name	a string describing the name of the data

Note

Note: if you just use this function and report the p-value then the function has failed in its purpose. If this function helps you to think about your analysis and what question(s) you are really interested in, create meaningful plots, and focus on the more meaningful parts of research, then it has succeeded. See also Cochrane's Aphorism.

Author(s)

Greg Snow <538280@gmail.com>

References

```
fortune(234)
```

See Also

qqnorm, vis.test

Examples

```
SnowsPenultimateNormalityTest(rt(100,25))
```

---

spread.labs *Spread out close points for labeling in plots*

Description

This function takes as set of coordinates and spreads out the close values so that they can be used in labeling plots without overlapping.

Usage

```
spread.labs(x, mindiff, maxiter = 1000, stepsize = 1/10, min = -Inf, max = Inf)
```

Arguments

x	The coordinate values (x or y, not both) to spread out.
mindiff	The minimum distance between return values
maxiter	The maximum number of iterations
stepsize	How far to move values in each iteration
min	Minimum bound for returned values
max	Maximum bound for returned values

Details

Sometimes the desired locations for labels in plots results in the labels overlapping. This function takes the coordinate values (x or y, not both) and finds those points that are less than mindiff (usually a function of strheight or strwidth) apart and increases the space between them (by stepsize * mindiff). This may or may not be enough and moving some points away from their nearest neighbor may move them too close to another neighbor, so the process is iterated until either maxiter steps have been tried, or all the values are at least mindiff apart.

The min and max arguments prevent the values from going outside that range (they should be specified such that the original values are all inside the range).

The values do not need to be presorted.

Value

A vector of coordinates (order corresponding to the original x) that can be used as a replacement for x in placing labels.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

text, the spread.labels function in the plotrix package.

Examples

```
# overlapping labels
plot(as.integer(state.region), state.x77[,1], ylab='Population',
xlab='Region',xlim=c(1,4.75), xaxt='n')
axis(1, at=1:4, lab=levels(state.region) )

text( as.integer(state.region)+.5, state.x77[,1], state.abb )
segments( as.integer(state.region)+0.025, state.x77[,1],
as.integer(state.region)+.375, state.x77[,1] )

# now lets redo the plot without overlap

tmp.y <- state.x77[,1]
for(i in levels(state.region) ) {
```

```
  tmp <- state.region == i
  tmp.y[ tmp ] <- spread.labs( tmp.y[ tmp ], 1.2*strheight('A'),
maxiter=1000, min=0 )
}

plot(as.integer(state.region), state.x77[,1], ylab='Population',
xlab='Region', xlim=c(1,4.75), xaxt='n')
axis(1, at=1:4, lab=levels(state.region) )

text( as.integer(state.region)+0.5, tmp.y, state.abb )
segments( as.integer(state.region)+0.025, state.x77[,1],
as.integer(state.region)+0.375, tmp.y )
```

`squishplot` *Squish the plotting area to a specified aspect ratio*

Description

Adjusts the plotting area to a specific aspect ratio. This is different from using the `asp` argument in that it puts the extra space in the margins rather than inside the plotting region.

Usage

```
squishplot(xlim, ylim, asp = 1, newplot=TRUE)
```

Arguments

`xlim`	The x limits of the plot, or the entire x vector.
`ylim`	The y limits of the plot, or the entire y vector.
`asp`	The y/x aspect ratio.
`newplot`	Should plot.new() be called before making the calculations.

Details

This function sets the plot area of the current graph device so that the following plot command will plot with the specified aspect ratio.

This is different from using the `asp` argument to `plot.default` in where the created white space goes (see the example). Using `plot.default` will place the whitespace within the plotting region and can result in the axes and annotations being quite far from the actual data. This command sets up the plotting region so that the extra whitespace is in the margin areas and moves the axes and annotations close to the data.

Any other desired parameter settings or resizing of the graphics device should be set before calling `squishplot`, especially settings dealing with multiple figures or margin areas.

After plotting, the parameters need to be reset or later plots may come out wrong.

Value

Invisible list containing the 'plt' values from par that were in place before the call to squishplot that can be used to reset the graphical parameters after plotting is finished.

Note

Remember to set other graphical parameters, then call squishplot, then call the plotting function(s), then reset the parameters.

Author(s)

Greg Snow <538280@gmail.com>

See Also

plot.default, plot.window, par

Examples

```
x <- rnorm(25, 10, 2 )
y <- 5 + 1.5*x + rnorm(25,0,2)

par(mfrow=c(1,3))
plot(x,y)

op <- squishplot(x,y,1)
plot(x,y)
par(op)

plot(x,y, asp=1)
```

---

steps *Steps data*

Description

This is the export from a pedometer worn for nearly about 11 months by the package author.

Usage

```
data(steps)
```

Format

A data frame with 331 observations on the following 79 variables.

`Date` The Date for the given data

`Total.Steps` Total Steps recorded for the day

`Aerobic.Steps` Total Aerobic Steps recorded for the day (see below)

`Aerobic.Walking.Time` Time spent in aerobic walking for the day

`Calories` Estimated calories burned for the day

`Distance` Estimated distance walked for the day in miles

`Fat.Burned` Estimated grams of fat burned by walking

`Steps.12AM` Steps recorded between Midnight and 1 am

`Steps.1AM` Steps recorded between 1 am and 2 am

`Steps.2AM` Steps recorded between 2 am and 3 am

`Steps.3AM` Steps recorded between 3 am and 4 am

`Steps.4AM` Steps recorded between 4 am and 5 am

`Steps.5AM` Steps recorded between 5 am and 6 am

`Steps.6AM` Steps recorded between 6 am and 7 am

`Steps.7AM` Steps recorded between 7 am and 8 am

`Steps.8AM` Steps recorded between 8 am and 9 am

`Steps.9AM` Steps recorded between 9 am and 10 am

`Steps.10AM` Steps recorded between 10 am and 11 am

`Steps.11AM` Steps recorded between 11 am and Noon

`Steps.12PM` Steps recorded between Noon and 1 pm

`Steps.1PM` Steps recorded between 1 pm and 2 pm

`Steps.2PM` Steps recorded between 2 pm and 3 pm

`Steps.3PM` Steps recorded between 3 pm and 4 pm

`Steps.4PM` Steps recorded between 4 pm and 5 pm

`Steps.5PM` Steps recorded between 5 pm and 6 pm

`Steps.6PM` Steps recorded between 6 pm and 7 pm

`Steps.7PM` Steps recorded between 7 pm and 8 pm

`Steps.8PM` Steps recorded between 8 pm and 9 pm

`Steps.9PM` Steps recorded between 9 pm and 10 pm

`Steps.10PM` Steps recorded between 10 pm and 11 pm

`Steps.11PM` Steps recorded between 11 pm and Midnight

`Aerobic.Steps.12AM` Aerobic steps recorded between Midnight and 1 am

`Aerobic.Steps.1AM` Aerobic steps recorded between 1 am and 2 am

`Aerobic.Steps.2AM` Aerobic steps recorded between 2 am and 3 am

`Aerobic.Steps.3AM` Aerobic steps recorded between 3 am and 4 am

`Aerobic.Steps.4AM` Aerobic steps recorded between 4 am and 5 am

`Aerobic.Steps.5AM` Aerobic steps recorded between 5 am and 6 am

`Aerobic.Steps.6AM` Aerobic steps recorded between 6 am and 7 am

`Aerobic.Steps.7AM` Aerobic steps recorded between 7 am and 8 am

`Aerobic.Steps.8AM` Aerobic steps recorded between 8 am and 9 am

`Aerobic.Steps.9AM` Aerobic steps recorded between 9 am and 10 am

`Aerobic.Steps.10AM` Aerobic steps recorded between 10 am and 11 am

`Aerobic.Steps.11AM` Aerobic steps recorded between 11 am and Noon

`Aerobic.Steps.12PM` Aerobic steps recorded between Noon and 1 pm

`Aerobic.Steps.1PM` Aerobic steps recorded between 1 pm and 2 pm

`Aerobic.Steps.2PM` Aerobic steps recorded between 2 pm and 3 pm

`Aerobic.Steps.3PM` Aerobic steps recorded between 3 pm and 4 pm

`Aerobic.Steps.4PM` Aerobic steps recorded between 4 pm and 5 pm

`Aerobic.Steps.5PM` Aerobic steps recorded between 5 pm and 6 pm

`Aerobic.Steps.6PM` Aerobic steps recorded between 6 pm and 7 pm

`Aerobic.Steps.7PM` Aerobic steps recorded between 7 pm and 8 pm

`Aerobic.Steps.8PM` Aerobic steps recorded between 8 pm and 9 pm

`Aerobic.Steps.9PM` Aerobic steps recorded between 9 pm and 10 pm

`Aerobic.Steps.10PM` Aerobic steps recorded between 10 pm and 11 pm

`Aerobic.Steps.11PM` Aerobic steps recorded between 11 pm and Midnight

`Used.12AM` Any movement detected between Midnight and 1 am

`Used.1AM` Any movement detected between 1 am and 2 am

`Used.2AM` Any movement detected between 2 am and 3 am

`Used.3AM` Any movement detected between 3 am and 4 am

`Used.4AM` Any movement detected between 4 am and 5 am

`Used.5AM` Any movement detected between 5 am and 6 am

`Used.6AM` Any movement detected between 6 am and 7 am

`Used.7AM` Any movement detected between 7 am and 8 am

`Used.8AM` Any movement detected between 8 am and 9 am

`Used.9AM` Any movement detected between 9 am and 10 am

`Used.10AM` Any movement detected between 10 am and 11 am

`Used.11AM` Any movement detected between 11 am and Noon

`Used.12PM` Any movement detected between Noon and 1 pm

`Used.1PM` Any movement detected between 1 pm and 2 pm

`Used.2PM` Any movement detected between 2 pm and 3 pm

`Used.3PM` Any movement detected between 3 pm and 4 pm

`Used.4PM` Any movement detected between 4 pm and 5 pm

`Used.5PM` Any movement detected between 5 pm and 6 pm

`Used.6PM` Any movement detected between 6 pm and 7 pm

`Used.7PM` Any movement detected between 7 pm and 8 pm

`Used.8PM` Any movement detected between 8 pm and 9 pm

`Used.9PM` Any movement detected between 9 pm and 10 pm

`Used.10PM` Any movement detected between 10 pm and 11 pm

`Used.11PM` Any movement detected between 11 pm and Midnight

Examples

```
data(steps)
## maybe str(steps) ; plot(steps) ...
```

`stork` *Neyman's Stork data*

Description

Data invented by Neyman to look at spurious correlations and adjusting for lurking variables by looking at the relationship between storks and biths.

Usage

```
data(stork)
```

Format

A data frame with 54 observations on the following 6 variables.

`County` ID of county

`Women` Number of Women (*10,000)

`No.storks` Number of Storks sighted

`No.babies` Number of Babies Born

`Stork.rate` Storks per 10,000 women (=No.storks/Women)

`Birth.rate` Babies per 10,000 women (=No.babies/Women)

Details

This is an entertaining example to show a relationship that is due to a third possibly lurking variable. The source paper shows how completely different relationships can be found by mis-analyzing the data.

Source

Kronmal, Richard A. (1993) Spurious Cerrolation and the Fallacy of the Ratio Standard Revisited. Journal of the Royal Statistical Society. Series A, Vol. 156, No. 3, 379-392.

References

Neyman, J. (1952) Lectures and Conferences on Mathematical Statistics and Probability, 2nd edn, pp. 143-154. Washington DC: US Department of Agriculture.

Examples

```
data(stork)
pairs(stork[,-1], panel=panel.smooth)
## maybe str(stork) ; plot(stork) ...
```

---

subplot *Embed a new plot within an existing plot*

Description

Subplot will embed a new plot within an existing plot at the coordinates specified (in user units of the existing plot).

Usage

```
subplot(fun, x, y, size=c(1,1), vadj=0.5, hadj=0.5,
 inset=c(0,0), type=c('plt','fig'), pars=NULL)
```

Arguments

fun	an expression defining the new plot to be embedded.
x	x-coordinate(s) of the new plot (in user coordinates of the existing plot), or a character string.
y	y-coordinate(s) of the new plot, x and y can be specified in any of the ways understood by xy.coords.
size	The size of the embedded plot in inches if x and y have length 1.
vadj	vertical adjustment of the plot when y is a scalar, the default is to center vertically, 0 means place the bottom of the plot at y, 1 places the top of the plot at y.
hadj	horizontal adjustment of the plot when x is a scalar, the default is to center horizontally, 0 means place the left edge of the plot at x, and 1 means place the right edge of the plot at x.
inset	1 or 2 numbers representing the proportion of the plot to inset the subplot from edges when x is a character string. The first element is the horizontal inset, the second is the vertical inset.
type	Character string, if 'plt' then the plotting region is defined by x, y, and size with axes, etc. outside that box; if 'fig' then all annotations are also inside the box.
pars	a list of parameters to be passed to par before running fun.

Details

The coordinates x and y can be scalars or vectors of length 2. If vectors of length 2 then they determine the opposite corners of the rectangle for the embedded plot (and the parameters size, vadj, and hadj are all ignored).

If x and y are given as scalars then the plot position relative to the point and the size of the plot will be determined by the arguments size, vadj, and hadj. The default is to center a 1 inch by 1 inch plot at x,y. Setting vadj and hadj to (0,0) will position the lower left corner of the plot at (x,y).

If x is a character string, then it will be parsed for the strings "left", "right", "top", and "bottom" and x and y will be set appropriately (anything not specified will be set at the center in that dimension) using also the inset argument. This allows the position of the subplot to be specified as 'topleft' or 'bottom', etc. The inset argument is in proportion of the plot units, so 0.1 means inset 10% of the width/height of the plotting distance. If hadj/vadj are not specified, they will be set appropriately.

The rectangle defined by x, y, size, vadj, and hadj will be used as the plotting area of the new plot. Any tick marks, axis labels, main and sub titles will be outside of this rectangle if type is 'plt'. If type is 'fig' then the annotations will be inside the box.

Any graphical parameter settings that you would like to be in place before fun is evaluated can be specified in the pars argument (warning: specifying layout parameters here (plt, mfrow, etc.) may cause unexpected results).

After the function completes the graphical parameters will have been reset to what they were before calling the function (so you can continue to augment the original plot).

Value

An invisible list with the graphical parameters that were in effect when the subplot was created. Passing this list to par will enable you to augment the embedded plot.

Author(s)

Greg Snow <538280@gmail.com>

See Also

grconvertX, par, symbols, my.symbols, ms.image

Examples

```
# make an original plot
plot( 11:20, sample(51:60) )

# add some histograms

subplot( hist(rnorm(100)), 15, 55)
subplot( hist(runif(100),main='',xlab='',ylab=''), 11, 51, hadj=0, vadj=0)
subplot( hist(rexp(100, 1/3)), 20, 60, hadj=1, vadj=1, size=c(0.5,2) )
subplot( hist(rt(100,3)), c(12,16), c(57,59), pars=list(lwd=3,ask=FALSE) )

### some of the following examples work fine in an interactive session,
### but loading the packages required does not work well in testing.
```

```
# augment a map
if( interactive() && require(spData) ){
plot(state.vbm,fg=NULL)
tmp <- cbind( state.vbm$center_x, state.vbm$center_y )
for( i in 1:50 ){
tmp2 <- as.matrix(USArrests[i,c(1,4)])
tmp3 <- max(USArrests[,c(1,4)])
subplot( barplot(tmp2, ylim=c(0,tmp3),names=c('',''),yaxt='n'),
x=tmp[i,1], y=tmp[i,2], size=c(.1,.1))
}
}

tmp <- rnorm(25)
qqnorm(tmp)
qqline(tmp)
tmp2 <- subplot( hist(tmp,xlab='',ylab='',main=''),
grconvertX(0.1,from='npc'), grconvertY(0.9,from='npc'),
vadj=1, hadj=0 )
abline(v=0, col='red') # wrong way to add a reference line to histogram

# right way to add a reference line to histogram
op <- par(no.readonly=TRUE)
par(tmp2)
abline(v=0, col='green')
par(op)


# scatter-plot using images
if(interactive() && require(png)) {

  image.png <- function(x,...) {
    cols <- rgb( x[,,1], x[,,2], x[,,3], x[,,4] )
    z <- 1:length(cols)
    dim(z) <- dim(x[,,1])
    z <- t(z)
    z <- z[ ,rev(seq_len(ncol(z))) ]
    image(z, col=cols, axes=FALSE, ...)
  }

  logo <- readPNG(system.file("img", "Rlogo.png", package="png"))

  x <- runif(10)
  y <- runif(10)
  plot(x,y, type='n')
  for(i in 1:10) {
    subplot(image.png(logo), x[i], y[i], size=c(0.3,0.3))
  }
}
```

TkApprox *Plot a set of data in a Tk window and interactively move lines to see predicted y-values corresponding to selected x-values.*

Description

This function plots a dataset in a Tk window then places 3 lines on the plot which show a predicted y value for the given x value. The lines can be clicked on and dragged to new x-values with the predicted y-values automatically updating. A table at the bottom of the graph shows the differences between the pairs of x-values and y-values.

Usage

```
TkApprox(x, y, type = "b", snap.to.x = FALSE, digits = 4,
  cols = c("red", "#009900", "blue"), xlab = deparse(substitute(x)),
  ylab = deparse(substitute(y)), hscale = 1.5, vscale = 1.5,
  wait = TRUE, ...)
```

Arguments

x	The x-values of the data, should be sorted
y	The corresponding y-values of the data
type	Type of plot (lines, points, both) passed to plot
snap.to.x	If True then the lines will snap to x-values (can be changed with a checkbox in the Tk window)
digits	Number of significant digits to display (passed to format)
cols	Vector of 3 colors, used for the reference lines
xlab	Label for x-axis
ylab	Label for y-axis
hscale	Horizontal Scale of the plot, passed to tkrplot
vscale	Vertical Scale of the plot, passed to tkrplot
wait	Should R wait for the window to be closed
...	Additional parameters passed to plot

Details

This provides an interactive way to explore predictions from a set of x and y values. Internally the function approxfun is used to make the predictions.

The x-value of the 3 reference lines can be changed by clicking and dragging the line to a new position. The x and y values are shown in the margins of the graph. Below the graph is a table with the differences (absolute value) between the pairs of points.

This can be used to find peaks/valleys in trends and to see how they differ from starting points, other peaks/valleys, etc..

Value

If wait is FALSE then an invisible NULL is returned, if wait is TRUE then an invisible list with the x and y values of the 3 reference lines is returned.

Author(s)

Greg Snow <538280@gmail.com>

See Also

approxfun, TkSpline

Examples

```
if(interactive()) {
  with(ccc, TkApprox(Time2,Elevation))
}
```

---

tkBrush *Change the Color and Styles of points interactively*

Description

Creates a Tk window with a scatterplot matrix, then allows you to "brush" the points to change their color and/or style.

Usage

```
tkBrush(mat,hscale=1.75,vscale=1.75,wait=TRUE,...)
```

Arguments

mat	A matrix of the data to plot, columns are variables, rows are observations, same as pairs
hscale	Passed to tkrplot
vscale	Passed to tkrplot
wait	Should the function wait for you to finish, see below
...	Additional arguments passed to the panel functions

Details

This function creates a Tk window with a pairs plot of `mat`, then allows you to interactively move a rectangle (the brush) over the points to change their color and plotting character.

The arrow keys can be used to change the size and shape of the brush. The left arrow makes the rectangle wider, the right makes it narrower. The up arrow key makes it taller, the right makes it shorter.

When the mouse button is not pressed the points inside the brush will change while in the brush, but return to their previous state when the brush moves off them. If the mouse button is pressed then the points inside the brush will be changed and the change will remain until a different set of conditions is brushed on them.

The style of the brushed points is determined by the values of the 2 entry boxes on the right side of the plot. You can specify the plotting character in the `pch` box, this can be anything that you would regularly pass to the `pch` argument of `points`, e.g. an integer or single character. You can specify the color of the brushed points using the `color` entry box, specify the name of any color recognized by R (see `colors`), if this box does not contain a legal color name then black will be used.

If `wait` is FALSE then the Tk window will exist independently of R and you can continue to do other things in the R window, in this case the function returns NULL. If `wait` is TRUE then R waits for you to close the Tk window (using the quit button) then returns a list with the colors and plotting characters resulting from your brushing, this information can be used to recreate the plot using `pairs` on a new graphics device (for printing or saving).

Value

Either NULL (if Wait=FALSE) or a list with components `col` and `pch` corresponding to the state of the points.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`pairs`,`colors`,`points`, the `iplots` package

Examples

```
if(interactive()){

# Iris dataset

out1 <- tkBrush(iris)

#  Now brush the points

pairs(iris, col=out1$col, pch=out1$pch)

# or

colhist <- function(x,...){
```

```
    tmp <- hist(x,plot=F)
    br <- tmp$breaks
    w <- as.numeric(cut(x,br,include.lowest=TRUE))
    sy <- unlist(lapply(tmp$counts,function(x)seq(length=x)))
    my <- max(sy)
    sy <- sy/my
    my <- 1/my
    sy <- sy[order(order(x))]
    tmp.usr <- par('usr'); on.exit(par(usr=tmp.usr))
    par(usr=c(tmp.usr[1:2],0,1.5))
    rect(br[w], sy-my, br[w+1], sy,
       col=out1$col, # note out1$col is hardcoded here.
       border=NA)
    rect(br[-length(br)], 0, br[-1], tmp$counts*my)
}
pairs(iris, col=out1$col, pch=out1$pch, diag.panel=colhist)


# some spheres

s1 <- matrix(nrow=0,ncol=3)

while( nrow(s1) < 1000 ){
tmp <- rnorm(3)
if( sum(tmp^2) <= 1 ){
s1 <- rbind(s1,tmp)
}
}

s2 <- matrix(rnorm(3000), ncol=3)
s2 <- s2/apply(s2,1,function(x) sqrt(sum(x^2)))

tkBrush(s1, wait=FALSE)
tkBrush(s2, wait=FALSE)

# now paint values where var 2 is close to 0 in both plots
# and compare the var 1 and var 3 relationship

}
```

TkBuildDist *Interactively create a probability distribution.*

Description

Build a probability distribution (one option for creating a prior distribution) by clicking or dragging a plot.

Usage

```
TkBuildDist(x = seq(min + (max - min)/nbin/2, max - (max - min)/nbin/2,
  length.out = nbin), min = 0, max = 10, nbin = 10, logspline = TRUE,
  intervals = FALSE)

TkBuildDist2( min=0, max=1, nbin=10, logspline=TRUE)
```

Arguments

x	A starting set of data points, will default to a sequence of uniform values.
min	The minimum value for the histogram
max	The maximum value for the histogram
nbin	The number of bins for the histogram
logspline	Logical, whether to include a logspline curve on the plot and in the output.
intervals	Logical, should the logspline fit be based on the interval counts rather than the clicked data points, also should the interval summary be returned.

Details

Bothe of these functions will open a Tk window to interact with. The window will show a histogram (the defaults will show a uniform distribution), optionally a logspline fit line will be included as well. Including the logspline will slow things down a bit, so you may want to skip it on slow computers.

If you use the TkBuildDist function then a left click on the histogram will add an additional point to the histogram bar clicked on (the actual x-value where clicked will be saved, returned, and used in the optional logspline unless intervals is TRUE). Right clicking on the histogram will remove the point closest to where clicked (based only on x), which will usually have the effect of decreasing the clicked bar by 1, but could affect the neigboring bar if you click near the edge or click on a bar that is 0.

If you use the TkBuildDist2 function then the individual bars can be adjusted by clicking at the top of a bar and dragging up or down, or clicking at what you want the new height of the bar to be. As the current bar is adjusted the other bars will adjust in the oposite direction proportional to their current heights.

The logspline fit assumes the basis for the distribution is the real line, the min and max arguments only control the histogram and where values can be changed.

Value

Both functions return a list with the breaks that were used the logspline fit (if logspline is TRUE), the x-values clicked on (for TkBuildDist), and the proportion of the distribution within each interval (for TkBuildDist2 or if intervals is TRUE).

Author(s)

Greg Snow <538280@gmail.com>

See Also

The logspline package

Examples

```
if(interactive()){
tmp1 <- TkBuildDist()
tmp2 <- TkBuildDist2()
}
```

---

tkexamp *Create Tk dialog boxes with controls to show examples of changing parameters on a graph.*

Description

This utility will create a Tk window with a graph and controls to change the parameters of the plotting function interactively.

Usage

```
 tkexamp(FUN, param.list, vscale=1.5, hscale=1.5, wait=FALSE,
plotloc='top', an.play=TRUE, print=FALSE, ...)
```

Arguments

FUN	A function call to create the example plot
param.list	A list of lists with information on the parameters to control and the controls to use. See Details Below
vscale	Vertical size of plot, passed to tkrplot
hscale	Horizontal size of plot, passed to tkrplot
wait	Should R wait for the demo to end
plotloc	Character with "top", "left", or "right" indicating where the plot should be placed relative to the controls
an.play	Should the scheduling in tcltk2 package be used for animations
print	Automatically print the result (useful for ggplot2/lattice)
...	Extra arguments, currently ignored

Details

This is a helper function to create interactive demonstrations of the effect of various function arguments on the resulting graph.

The FUN argument should be a function call to create the basic plot (if run stand alone this should create the starting plot). The arguments to be changed should not be included.

The param.list is a nested list of lists that defines which controls to use for which function arguments. Additional levels of nested lists creates groups of controls (see examples below) and if the list is named in the enclosing list, that name will be used to label the group.

The lowest level of lists control a single function argument with the control to be used. The name of the list in the enclosing list is the name of the function argument to be used, e.g. "pch=list(...)" will create a control for the pch parameter.

The first element of the innermost list is a character string specifying which control to use (from the list below), the rest of the elements must be named and specify parameters of the controls. For details on all possible parameters see the tcltk documentation. Any parameter can be set using this list, for example most controls have a width parameter that can be set with code like width=5. Most controls also have an init argument that specifies the initial value that the control will be set to (most have a default in case you don't specify the value).

The following are the possible controls you can specify as the first element of the list along with the most common parameters to specify:

"numentry", an entry box where a number can be typed in, this will be passed to FUN wrapped in as.numeric().

"entry", an entry box where a character string can be typed in (this will be passed to FUN as a character string, not converted).

"slider", a slider (or scale) that can be dragged left and right to choose the different values. The common parameters to specify are "from" (the lowest value), "to" (the largest value), and "resolution" (the increment size when sliding).

"vslider", just like slider except that the slider is dragged up and down rather than left and right.

"spinbox", an entry widget for a number with small arrows on the right side that can be used to increment/decrement the value, or you can type in a value. The common parameters to set are "from" (smallest value), "to" (largest value), and "increment" (how much to change the value by when clicking on the arrows). You can also set "values" which is a vector of values that can be used. This will be passed to FUN as a number.

"checkbox", a box that can be checked, passed to FUN as a logical (TRUE if checked, FALSE if not checked). To set the intial value as TRUE (the default is FALSE) use init='T'.

"combobox", an entry widget with an arrow on the right side that will bring up a list of values to choose from. This value is passed to FUN as a character string. The important parameter to set is "values" which is a vector of character strings to choose between. This option will only work with tcl version 8.5 or later and will probably produce an error in earlier versions.

"radiobuttons", a set of choices with check boxes next to each, when one is selected the previous selection is cleared. The important parameter to set is "values" wich is a vector of character strings to choose between.

"animate", is a combination of a slider and a button. If the tcltk2 package is avaliable and an.play=TRUE then the button will say "Play" and pressing the button will automatically increment the slider (and

update the graph) until it reaches the maximum value. Otherwise the button will say "Inc" and you must click and hold on the button to run the animation (this might be prefered in that you can stop the animation). Either way you can set the delay option (all other options match with the slider option) and move the slider when the interaction is not happening. The animation starts at the current value on the slider and goes to the maximum value. You should only have at most one animation control (multiple will confuse each other), this includes not having multiple windows operating at the same time with animation controls.

Each nesting of lists will also change how the controls are placed (top to bottom vs. left to right).

The Tk window will also have a default set of controls at the bottom. These include entry widgets for `vscale` and `hscale` for changing the size of the graph (initially set by arguments to `tkexamp`). A "Refresh" button that will refresh the graph with the new parameter values (some controls like sliders will automatically refresh, but others like entries will not refresh on their own and you will need to click on this button to see the updates). A "Print Call" button that when clicked will print a text string to the R terminal that represents the function call with the current argument settings (copying and pasting this to the command line should recreate the current plot on the current plotting device). And an "Exit" button that will end the demo and close the window.

Value

If `wait` is FALSE then it returns an invisible NULL, if `wait` is TRUE then it returns a list with the argument values when the window was closed.

Note

You can move the sliders in 3 different ways: You can left click and drag the slider itself, you can left click in the trough to either side of the slider and the slider will move 1 unit in the direction you clicked, or you can right click in the trough and the slider will jump to the location you clicked at.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

`tkrplot`, the fgui package, the playwith package, and the rpanel package

Examples

```
if(interactive()) {

x <- sort( runif(25,1,10) )
y <- rnorm(25, x)

# some common plotting parameters

tke.test1 <- list(Parameters=list(
                 pch=list('spinbox',init=1,from=0,to=255,width=5),
                 cex=list('slider',init=1.5,from=0.1,to=5,resolution=0.1),
                 type=list('combobox',init='b',
                   values=c('p','l','b','o','c','h','s','S','n'),
```

```
                      width=5),
                lwd=list('spinbox',init=1,from=0,to=5,increment=1,width=5),
                lty=list('spinbox',init=1,from=0,to=6,increment=1,width=5)
                ))

tkexamp( plot(x,y), tke.test1, plotloc='top' )

# different controls for the parameters

tke.test2 <- list(Parameters=list(
                pch=list('spinbox',init=1,values=c(0:25,32:255),width=5),
                cex=list('slider',init=1.5,from=0.1,to=5,resolution=0.1),
                type=list('radiobuttons',init='b',
                  values=c('p','l','b','o','c','h','s','S','n'),
                       width=5),
                lwd=list('spinbox',init=1,from=0,to=5,increment=1,width=5),
                lty=list('spinbox',init=1,from=0,to=6,increment=1,width=5),
                xpd=list('checkbox')
                ))

tkexamp( plot(x,y), tke.test2, plotloc='left')

tmp <- tkexamp( plot(x,y), list(tke.test2), plotloc='right', wait=TRUE )

# now recreate the plot
tmp$x <- x
tmp$xlab <- 'x'
tmp$y <- y
tmp$ylab <- 'y'
do.call('plot', tmp)

# a non plotting example

tke.test3 <- list(
        sens=list('slider', init=0.95, from=0.9, to=1, resolution=0.005),
        spec=list('slider', init=0.9, from=0.8, to=1, resolution=0.005),
        prev=list('slider', init=0.01, from=0.0001, to=0.1, resolution=0.0001),
        step=list('spinbox', init=1, from=1, to=11, width=5),
        n=list('numentry',init=100000, width=7)
)

options(scipen=1)
tkexamp( SensSpec.demo(), tke.test3 )
# now increment step and watch the console

# Above example but converting it to plot

tempfun <- function(sens,spec,prev,step,n) {
if(missing(sens) || missing(n)) return(invisible(NULL))
tmp <- capture.output( SensSpec.demo(sens=sens,spec=spec,
prev=prev, n=n, step=step) )
par(cex=2.25)
plot.new()
```

```
tmp2 <- strheight(tmp)
text(0, 1-cumsum(tmp2*1.5), tmp, family='mono', adj=0)
title('Sensitivity and Specificity Example')
}

tkexamp( tempfun(), tke.test3, hscale=4, vscale=2 )

# an example using trellis graphics

tke.test4 <- list(
   alpha=list('slider', from=0,to=1,init=1,
               resolution=0.05),
   cex=list('spinbox',init=.8,from=.1,to=3,increment=.1,width=5),
   col=list('entry',init='#0080ff'),
   pch=list('spinbox',init=1, from=0,to=255,
               increment=1,width=5),
   fill=list('entry',init='transparent')
)

tempfun <- function(x,y,alpha,cex,col,pch,fill) {
if(missing(alpha) || missing(cex)) {return()}
trellis.par.set(plot.symbol=list(alpha=alpha, cex=cex, col=col,
font=1,pch=pch,fill=fill))
print(xyplot( y~x ))
}

require(lattice)

tkexamp( tempfun(x,y), list(tke.test4), plotloc='left')


# Two example using ggplot2

if( require(ggplot2) ) {

##  1
  tkexamp( qplot(cty,data=mpg, geom='histogram'),
      list(binwidth=list('slider',from=1,to=25)),
      print=TRUE)

##  2
  tmpfun <- function(bw=2){
print(ggplot(mpg, aes(cty)) +
   geom_histogram(binwidth = bw))
  }

  tkexamp( tmpfun, list(bw=list('slider',from=1, to=5)))


}

}
```

TkListView *Interactivly view structures of list and list like objects.*

Description

This is somewhat like the `str` function, except that it creates a new Tk window and a tree object representing the list or object. You can then click on the '+' signs to expand branches of the list to see what they contain.

Usage

```
TkListView(list)
```

Arguments

`list`	The list or object to be viewed.

Details

This function opens a Tk window with a tree view of the list in the leftmost pane. Next to the tree is the result from the `str` function for each element of the list. Clicking on the '+' symbol next to list elements will expand the tree branch to show what that list/sublist contains. On the right is an output pane with 3 buttons below it. These can be used by first selecting (clicking on) a list element in the left pane (this can be a whole list or single element), then clicking on one of the buttons. The output from the button appears in the right pane (replacing anything that may have been there before). The 'print' button just prints the element/sublist selected. The 'str' button calls the `str` function on the selected element/list/sublist. The 'Eval:' button will evaluate the code in the entry box next to it with the selected element of the list being the 'x' variable. For example you could click on an element in the list that is a numeric vector, type 'hist(x)' in the entry box, and click on the 'Eval:' button to produce a histogram (current/default R graphics device) of the data in that element.

any lists/objects with attributes will show the attributes as an additional branch in the tree with a label of "«attributes»".

This function works on S3 objects that are stored as lists. Since currently S4 objects are saved as attributes, wrapping them in a list will work with this function to view their structure, see the example below.

Value

This function is ran for its side effects, it does not return anything of use.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

`str`

Examples

```
if(interactive()) {
  tmp <- list( a=letters, b=list(1:10, 10:1), c=list( x=rnorm(100),
        z=data.frame(x=rnorm(10),y=rnorm(10))))
  TkListView(tmp)

  fit <- lm(Petal.Width ~ ., data=iris)
  TkListView(fit)


  if(require(stats4)){
   # this example is copied almost verbatim from ?mle
         x <- 0:10
     y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
     ll <- function(ymax=15, xhalf=6)
         -sum(stats::dpois(y, lambda=ymax/(1+x/xhalf), log=TRUE))
     (fit <- mle(ll))
     TkListView(list(fit))
  }
}
```

`TkPredict` *Plot predicted values from a model against one of the predictors for a given value of the othe predictors*

Description

These functions create a plot of predicted values vs. one of the predictors for given values of the other predictors. TkPredict further creates a Tk gui to allow you to change the values of the other predictors.

Usage

```
Predict.Plot(model, pred.var, ..., type='response', add=FALSE,
plot.args=list(), n.points=100, ref.val, ref.col='green', ref.lty=1,
data)
TkPredict(model, data, pred.var, ...)
```

Arguments

`model`	A model of class 'lm' or 'glm' (or possibly others) from which to plot predictions.
`pred.var`	A character string indicating which predictor variable to put on the x-axis of the plot.
`...`	for `Predict.Plot` The predictor variables and their values for the predictions. See below for detail.
`type`	The type value passed on to the predict function.

add	Whether to add a line to the existing plot or start a new plot.
plot.args	A list of additional options passed on to the plotting function.
n.points	The number of points to use in the approximation of the curve.
ref.val	A reference value for the pred.var, a reference line will be drawn at this value to the corresponding predicted value.
ref.col, ref.lty	The color and line type of the reference line if plotted.
data	The data frame or environment where the variables that the model was fit to are found. If missing, the model will be examined for an attempt find the needed data.

Details

These functions plot the predicted values from a regression model (lm or glm) against one of the predictor variables for given values of the other predictors. The values of the other predictors are passed as the ... argument to Predict.Plot or are set using gui controls in TkPredict (initial values are the medians).

If the variable for the x axis (name put in pred.var) is not included with the ... variables, then the range will be computed from the data argument or the data component of the model argument.

If the variable passed as pred.var is also included in the ... arguments and contains a single value, then this value will be used as the ref.val argument.

If it contains 2 or more values, then the range of these values will be used as the x-limits for the predictions.

When running TkPredict you can click on the "Print Call" button to print out the call of Predict.Plot that will recreate the same plot. Doing this for different combinations of predictor values and editing the plot.args and add arguments will give you a script that will create a static version of the predictions.

Value

These functions are run for their side effects of creating plots and do not return anything.

Note

The GUI currently allows you to select a factor as the x-variable. If you do this it will generate some errors and you will not see the plot, just choose a different variable as the x-variable and the plot will return.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

tkrplot, tkexamp, predict

Examples

```
library(splines)

fit.lm1 <- lm( Sepal.Width ~ ns(Petal.Width,3)*ns(Petal.Length,3)+Species,
data=iris)

Predict.Plot(fit.lm1, pred.var = "Petal.Width", Petal.Width = 1.22,
      Petal.Length = 4.3, Species = "versicolor",
  plot.args = list(ylim=range(iris$Sepal.Width), col='blue'),
      type = "response")
Predict.Plot(fit.lm1, pred.var = "Petal.Width", Petal.Width = 1.22,
      Petal.Length = 4.3, Species = "virginica",
plot.args = list(col='red'),
      type = "response", add=TRUE)
Predict.Plot(fit.lm1, pred.var = "Petal.Width", Petal.Width = 1.22,
      Petal.Length = 4.4, Species = "virginica",
plot.args = list(col='purple'),
      type = "response", add=TRUE)

fit.glm1 <- glm( Species=='virginica' ~ Sepal.Width+Sepal.Length,
data=iris, family=binomial)

Predict.Plot(fit.glm1, pred.var = "Sepal.Length", Sepal.Width = 1.99,
      Sepal.Length = 6.34, plot.args = list(ylim=c(0,1), col='blue'),
  type = "response")
Predict.Plot(fit.glm1, pred.var = "Sepal.Length", Sepal.Width = 4.39,
      Sepal.Length = 6.34, plot.args = list(col='red'),
type = "response", add=TRUE)




if(interactive()){
TkPredict(fit.lm1)

TkPredict(fit.glm1)
}
```

`TkSpline` *Plot a set of data in a Tk window and interactively move a line to see predicted y-values from a spline fit corresponding to selected x-values.*

Description

This function plots a dataset in a Tk window then draws the spline fit through the points. It places a line to show the predicted y from the given x value. The line can be clicked on and dragged to new x-values with the predicted y-values automatically updating. A table at the bottem of the graph shows the values and the 3 derivatives.

Usage

```
TkSpline(x, y, method='natural', snap.to.x=FALSE, digits=4,
                     col=c('blue','#009900','red','black'),
                     xlab=deparse(substitute(x)), ylab=deparse(substitute(y)),
                     hscale=1.5, vscale=1.5, wait=TRUE,
                     ...)
```

Arguments

x	The x-values of the data, should be sorted
y	The corresponding y-values of the data
method	Spline Method, passed to splinefun
snap.to.x	Logical, if TRUE then the line will only take on the values of x
digits	Number of digits to print, passed to format
col	Colors of the prediction and other lines
xlab	Label for the x-axis, passed to plot
ylab	Label for the y-axis, passed to plot
hscale	Horizontal scaling, passed to tkrplot
vscale	Vertical scaling, passed to tkrplot
wait	Should R wait for the window to close
...	Additional parameters passed to plot

Details

This provides an interactive way to explore predictions from a set of x and y values. Internally the function splinefun is used to make the predictions.

The x-value of the reference line can be changed by clicking and dragging the line to a new position. The x and y values are shown in the margins of the graph. Below the graph is a table with the y-value and derivatives.

Value

If wait is FALSE then an invisible NULL is returned, if wait is TRUE then an invisible list with the x and y values and derivatives is returned.

Author(s)

Greg Snow <538280@gmail.com>

See Also

splinefun, TkApprox

Examples

```
if(interactive()) {
  x <- 1:10
  y <- sin(x)
  TkSpline(x,y, xlim=c(0,11))
}
```

---

towork *Sample data downloaded and converted from a GPS unit*

Description

These are GPS information from 3 trips.

Format

Data frames with the following variables.

`Index` Measurement number

`Time` a POSIXt, Time of measurement

`Elevation` a numeric vector, Elevation in Feet

`Leg.Dist` a character/numeric vector, The distance traveled in that leg (in feet for `ccc`)

`Leg.Time` a difftime, the time of that leg

`Speed` a numeric vector, Speed in mph

`Direction` a numeric vector, Direction in Degrees, 0 is North, 90 is East, 180 is South, 270 is West

`LatLon` a character vector, Latitude and Longitude as characters

`Leg.Dist.f` a numeric vector, Length of that leg in feet

`Leg.Dist.m` a numeric vector, Length of that leg in miles

`Lat` a numeric vector, Numeric latitude

`Lon` a numeric vector, Numeric longitude (negative for west)

`Distance` a numeric vector, Distance from start in feet

`Distance.f` a numeric vector, Distance from start in feet

`Distance.m` a numeric vector, Distance from start in miles

`Time2` a difftime, Time from start

`Time3` a difftime, cumsum of `Leg.Time`

Details

The data frame `ccc` came from when I was walking back to my office from a meeting and decided to take the scenic route and started the GPS.

The data frame `h2h` is a trip from my office to another for a meeting. The first part is traveling by car, the last part by foot from the parking lot to the building. Speed is a mixture of distributions.

The data frame `towork` came from driving to work one morning (the first point is where the GPS got it's first lock, not my house). The overall trip was mostly NorthWest but with enough North and NorthEast that a simple average of direction shows SouthEast, good example for circular stats.

Source

My GPS device

Examples

```
if( interactive() ){
  with(ccc, TkApprox(Distance, Elevation))
}
```

`tree.demo` *Interactively demonstrate regression trees*

Description

Interactively recursively partition a dataset to demonstrate regression trees.

Usage

```
tree.demo(x, y)
```

Arguments

x	The predictor variable.
y	The response variable.

Details

This function first creates a scatterplot of x and y and shows the residual sum of squares from fitting a horizontal line to the y-values.

Clicking anywhere on the graph will show an updated graph where the data is partitioned into 2 groups based on the x-value where you clicked with a horizontal line fit to each group (including showing the updated residual sum of squares). Clicking again will move the partitioning value based on the new click.

When you have found a good partitioning (reduces the RSS), right click and choose 'stop' and that partition will become fixed. Now you can click to do a second set of partions (breaking the data into 3 groups).

To finish the demo, right click and choose 'stop', then right click again and choose 'stop' again.

Value

A vector with the x-values of the cut points that you selected (sorted).

Author(s)

Greg Snow <538280@gmail.com>

See Also

The rpart and tree packages

Examples

```
if(interactive()){
data('ethanol', package='lattice')
print(with(ethanol, tree.demo(E,NOx)))
}
```

`triplot` *Create or add to a Trilinear Plot*

Description

Create (or add to) a trilinear plot of 3 proportions that sum to 1.

Usage

```
triplot(x, y = NULL, z = NULL, labels = dimnames(x)[[2]],
  txt = dimnames(x)[[1]], legend = NULL, legend.split = NULL,
  inner = TRUE, inner.col = c("lightblue", "pink"), inner.lty = c(2, 3),
  add = FALSE, main = "", ...)
```

Arguments

`x`	Vector or matrix of up to 3 columns.
`y`	Vector (if `x` is a vector).
`z`	Vector (if `x` is a vector).
`labels`	Labels for the 3 components (printed at corners).
`txt`	Vector of text strings to be plotted instead of points.
`legend`	Labels for the data points
`legend.split`	What proportion of the labels will go on the left.
`inner`	Logical, should the inner reference lines be plotted.
`inner.col`	Colors for the 2 inner triangles.
`inner.lty`	Line types for the 2 inner triangles.
`add`	Add points to existing plot (TRUE), or create a new plot (FALSE).
`main`	Main title for the plot.
`...`	Additional arguments passed on to `points` or `text`.

Details

Trilinear plots are useful for visualizing membership in 3 groups by plotting sets of 3 proportions that sum to 1 within each set.

The data can be passed to the function as a matrix with either 2 or 3 columns, or as seperate vectors to `x`, `y`, and optionaly `z`. If 2 columns are passed in, then they must be between 0 and 1 and the 3rd column will be created by subtracting both from 1. If 3 columns of data are given to the function then each will be divided by the sum of the 3 columns (they don't need to sum to 1 before being passed in).

Value

An invisible matrix with 2 columns and the same number of rows as `x` corresponding to the points plotted (after transforming to 2 dimensions).

The return matrix can be passed to `identify` for labeling of individual points.

Using `type='n'` and `add=FALSE` will return the transformed points without doing any plotting.

Author(s)

Greg Snow <538280@gmail.com>

References

Allen, Terry. Using and Interpreting the Trilinear Plot. Chance. 15 (Summer 2002).

See Also

`triangle.plot` in package ade4, `ternaryplot` in package vcd, `tri` in package cwhtool, `soil.texture` and `triax.plot` in package plotrix.

Examples

```
triplot(USArrests[c(1,4,2)])
tmp <- triplot(USArrests[c(1,4,2)],txt=NULL)
if(interactive()){ identify(tmp, lab=rownames(USArrests) ) }

tmp <- rbind( HairEyeColor[,,'Male'], HairEyeColor[,,'Female'])
tmp[,3] <- tmp[,3] + tmp[,4]
tmp <- tmp[,1:3]
triplot(tmp, legend=rep(c('Male','Femal'),each=4),
 col=rep(c('black','brown','red','yellow'),2))
```

txtStart *Save a transcript of commands and/or output to a text file.*

Description

These functions save a transcript of your commands and their output to a script file, possibly for later processing with the "enscript" or "pandoc" program.

They work as a combinations of `sink` and `history` with a couple extra bells and whistles.

Usage

```
txtStart(file, commands=TRUE, results=TRUE, append=FALSE, cmdfile,
          visible.only=TRUE)

txtStop()

txtComment(txt,cmdtxt)

txtSkip(expr)

etxtStart(dir = tempfile("etxt"), file = "transcript.txt",
           commands = TRUE, results = TRUE, append = FALSE,
           cmdbg = "white", cmdcol = "red", resbg = "white",
           rescol = "navy", combg = "cyan", comcol = "black",
           cmdfile, visible.only = TRUE)

etxtStop()

etxtComment(txt, cmdtxt)

etxtSkip(expr)

etxtPlot(file=paste(tempfile('plot',R2txt.vars$dir),'.eps',sep=''),
         width=4, height=4)

wdtxtStart(commands=TRUE, results=TRUE, fontsize=9, cmdfile,
          visible.only=TRUE)

wdtxtStop()

wdtxtComment(txt,cmdtxt)

wdtxtSkip(expr)

wdtxtPlot(height=5, width=5, pointsize=10)
```

```
mdtxtStart(dir=tempfile('mdtxt'), file='transcript.md',
           commands=TRUE, results=TRUE, append=FALSE,
           cmdfile, visible.only=TRUE)

mdtxtStop()

mdtxtComment(txt,cmdtxt)

mdtxtSkip(expr)

mdtxtPlot(file=tempfile('plot',R2txt.vars$dir,'.png'),
                    width=4, height=4)
```

Arguments

dir	Directory to store transcript file and any graphics file in
file	Text file to save transcript in
commands	Logical, should the commands be echoed to the transcript file
results	Logical, should the results be saved in the transcript file
append	Logical, should we append to file or replace it
cmdbg	Background color for command lines in file
cmdcol	Color of text for command lines in file
resbg	Background color for results sections in file
rescol	Text color of results sections in file
combg	Background color for comments in file
comcol	Text color of comments in file
cmdfile	A filename to store commands such that it can be sourced or copied and pasted from
visible.only	Should non-printed output be included, not currently implemented.
txt	Text of a comment to be inserted into file
cmdtxt	Text of a comment to be inserted into cmdfile
expr	An expression to be executed without being included in file or cmdfile
width	Width of plot, passed to dev.copy2eps, wdPlot, or dev.copy
height	Height of plot, passed to dev.copy2eps, wdPlot, or dev.copy
fontsize	Size of font to use in MSWord
pointsize	passed to wdPlot

Details

These functions are used to create transcript/command files of your R session. There are 4 sets of functions, those starting with "txt",those starting with "etxt", and those starting with "wdtxt" and those starting with "mdtxt". The "txt" functions create a plain text transcript while the "etxt"

functions create a text file with extra escapes and commands so that it can be post processed with enscript (an external program) to create a postscript file and can include graphics as well. The postscript file can be converted to pdf or other format file.

The "wdtxt" functions will insert the commands and results into a Microsoft Word document.

The "mdtxt" functions create a text file but with MarkDown escapes so that it can be post processed with "pandoc" (an external program) to create other formats such as html, pdf, MS Word documents, etc. If the command starts with the string "pander" or "pandoc" (after optional whitespace) then the results will be inserted directly, without escapes, into the transcript file. This assumes that you are using code from the "pander" package which generates markdown formatted output. This will create nicer looking tables and other output.

If `results` is TRUE and `commands` is FALSE then the result is similar to the results of `sink`. If `commands` is true as well then the transcript file will show both the commands and results similar to the output on the screen. If both `commands` and `results` are FALSE then pretty much the only thing these functions will accomplish is to waste some computing time.

If `cmdfile` is specified then an additional file is created with the commands used (similar to the `history` command), this file can be used with `source` or copied and pasted to the terminal.

The Start functions specify the file/directory to create and start the transcript, `wdtxtStart` will open Word if it is not already open or create a connection to an open word window. The prompts are changed to remind you that the commands/results are being copied to the transcript. The Stop functions stop the recording and reset the prompts.

The R parser strips comments and does some reformatting so the transcript file may not match exactly with the terminal output. Use the `txtComment`, `etxtComment`, `wdtxtComment`, or `mdtxtComment` functions to add a comment. This will show up as a line offset by whitespace in the transcript file, highlighted in the etxt version, and the default font in Word. If `cmdtxt` is specified then that line will be inserted into `cmdfile` preceded by a # so it will be skipped if sourced or copied.

The `txtSkip`, `etxtSkip`, `wdtxtSkip`, and `mdtxtSkip` functions will run the code in `expr` but will not include the commands or results in the transcript file (this can be used for side computations, or requests for help, etc.).

The `etxtPlot` function calls `dev.copy2eps` to create a copy of the current plot and inserts the proper command into the transcript file so that the eps file will be included in the final postscript file after processing.

The `wdtxtPlot` function calls `wdPlot` to send a copy of the current graph to MS Word.

The `mdtxtPlot` function calls `dev.copy` to create a copy of the current plot as a .png file and inserts the proper command into the transcript file so that the .png file will be included when processing with pandoc.

Value

Most of these commands do not return anything of use. The exceptions are:

`etxtStop` returns the name of the transcript file (including the directory path).

`txtSkip`, `etxtSkip`, `wdtxtSkip`, and `mdtxtSkip` return the value of `expr`.

Note

These commands do not do any fancy formatting of output, just what you see in the regular terminal window. If you want more formatted output then you should look into `Sweave`, `knitr`, or the R2HTML package.

The MS word functions will insert into the current word document at the location of the cursor. This means that if you look at the document and move the current location to somewhere in the middle (or have another word document open with the location in the middle), when you go back to R, the new transcript will be inserted into the middle of the document. So be careful to position the cursor at the end of the correct document before going back to R. Note that the "wdtxt" functions depend on the "R2wd" package which in turn depends on tools that are not free.

Do not use these functions in combination with R2HTML or `sink`. Only one of these sets of functions will work at a time.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

`sink`, `history`, `Sweave`, the odfWeave package, the R2HTML package, the R2wd package, the pander package

Examples

```
## Not run:
etxtStart()
etxtComment('This is todays transcript')
date()
x <- rnorm(25)
summary(x)
stem(x)
etxtSkip(?hist)
hist(x)
etxtPlot()
Sys.Date()
Sys.time()
my.file <- etxtStop()

# assumes enscript and ps2pdf are on your path
system(paste('enscript -e -B -p transcript.ps ', my.file) )
system('ps2pdf transcript.ps')

# if the above commands used mdtxt instead of etxt and the pandoc
# program is installed and on your path (and dependent programs) then use:
system(paste('pandoc -o transcript.docx ', my.file))

## End(Not run)
```

updateusr *Updates the 'usr' coordinates in the current plot.*

Description

For a traditional graphics plot this function will update the 'usr' coordinates by transforming a pair of points from the current usr coordinates to those specified.

Usage

```
updateusr(x1, y1 = NULL, x2, y2 = NULL)
```

Arguments

x1	The x-coords of 2 points in the current 'usr' coordianates, or anything that can be passed to xy.coords.
y1	The y-coords of 2 points in the current 'usr' coordinates, or an object representing the points in the new 'usr' coordinates.
x2	The x-coords for the 2 points in the new coordinates.
y2	The y-coords for the 2 points in the new coordinates.

Details

Sometimes graphs (in the traditional graphing scheme) end up with usr coordinates different from expected for adding to the plot (for example barplot does not center the bars at integers). This function will take 2 points in the current 'usr' coordinates and the desired 'usr' coordinates of the 2 points and transform the user coordinates to make this happen. The updating only shifts and scales the coordinates, it does not do any rotation or warping transforms.

If x1 and y1 are lists or matricies and x2 and y2 are not specified, then x1 is taken to be the coordinates in the current system and y1 is the coordinates in the new system.

Currently you need to give the function exactly 2 points in each system. The 2 points cannot have the same x values or y values in either system.

Value

An invisible list with the previous 'usr' coordinates from par.

Note

Currently you need to give coordinates for exactly 2 points without missing values. Future versions of the function will allow missing values or multiple points.

Author(s)

Greg Snow, <538280@gmail.com>

See Also

par

Examples

```
tmp <- barplot(1:4)
updateusr(tmp[1:2], 0:1, 1:2, 0:1)
lines(1:4, c(1,3,2,2), lwd=3, type='b',col='red')

# update the y-axis to put a reference distribution line in the bottom
# quarter

tmp <- rnorm(100)
hist(tmp)
tmp2 <- par('usr')
xx <- seq(min(tmp), max(tmp), length.out=250)
yy <- dnorm(xx, mean(tmp), sd(tmp))
updateusr( tmp2[1:2], tmp2[3:4], tmp2[1:2], c(0, max(yy)*4) )
lines(xx,yy)
```

USCrimes *US Crime Statistics*

Description

This is a 3 dimensional Array of the US crime statistics downloaded from the "Uniform Crime Reporting Statistics" of the US government. It comprises the years 1960 through 2010 for all 50 states, Washington DC, and a total for the country.

Usage

```
data(USCrimes)
```

Format

The format is: num [1:52, 1:51, 1:19] 3266740 226167 1302161 1786272 15717204 ... - attr(*, "dimnames")=List of 3 ..$ State: chr [1:52] "Alabama" "Alaska" "Arizona" "Arkansas" ... ..$: chr [1:51] "1960" "1961" "1962" "1963" ... ..$: chr [1:19] "Population" "ViolentCrimeRate" "MurderRate" "RapeRate" ...

Details

The first dimension is the state, the dimnames match the variable `state.name` with the exception of including "District of Columbia" in the 9th position (alphabetically) and "United States-Total" in position 45 (alphabetical).

The second dimension is the year, ranging from 1960 to 2010. If indexing by year, remember to put the year in quotes.

The third dimension is the variable:

Population: Total number of residents

ViolentCrimeRate: The total of the violent crimes (Murder, Rape, Robbery, Assault) per 100,000 population

MurderRate: The number of Murders and Nonnegligent Manslaughters per 100,000 population

RapeRate: Forcible Rapes per 100,000 population

RobberyRate: Robberies per 100,000 population

AssaultRate: Aggravated Assults per 100,000

PropertyCrimeRate: The total of the property crimes (Burglary, Theft, Vehicle Theft) per 100,000 population

BurglaryRate: Burglaries per 100,000 population

TheftRate: Larceny-Thefts per 100,000 population

VehicleTheftRate: Motor Vehicle Thefts per 100,000 population

ViolentCrimeTotal: The total of the violent crimes (Murder, Rape, Robbery, Assault

Murder: The number of Murders and Nonnegligent Manslaughters

Rape: Forcible Rapes

Robbery: Robberies

Assault: Aggravated Assults

PropertyCrimeTotal: The total of the property crimes (Burglary, Theft, Vehicle Theft)

Burglary: Burglaries

Theft: Larceny-Thefts

VehicleTheft: Motor Vehicle Thefts

Source

Originally: "https://ucrdatatool.gov/"", but that site does not work any more. Likely source for similar data: "https://www.fbi.gov/how-we-can-help-you/more-fbi-services-and-information/ucr/publications"

Examples

```
data(USCrimes)
## maybe str(USCrimes)

# plot time series/sparkline for each state
if(require(spData) && interactive()) {
data(state.vbm)
plot(state.vbm)

tmp.x <- state.vbm$center_x
tmp.x <- c( tmp.x[1:8], 147, tmp.x[9:43], 83, tmp.x[44:50] )
tmp.y <- state.vbm$center_y
tmp.y <- c( tmp.y[1:8], 45, tmp.y[9:43], -18, tmp.y[44:50] )
tmp.r <- range( USCrimes[,'ViolentCrimeRate'], na.rm=TRUE)
for(i in 1:52) {
subplot( plot(1960:2010, USCrimes[i,,'ViolentCrimeRate'],
ann=FALSE, bty='n', type='l', axes=FALSE),
```

```
tmp.x[i], tmp.y[i], size=c(0.2,0.2) )
}
}


## Gapminder style animation over time
if( interactive() ) {
x.r <- range( USCrimes[-c(9,45),,'Population'], na.rm=TRUE )
y.r <- range( USCrimes[-c(9,45),,'PropertyCrimeRate'], na.rm=TRUE )

tmpfun <- function(Year=1960, ... ) {
y <- as.character(Year)
plot( USCrimes[-c(9,45),y,'Population'],
 USCrimes[-c(9,45),y,'PropertyCrimeRate'],
type='n', xlab='log Population',
ylab='Property Crime Rate',
main=y, xlim=x.r, ylim=y.r, log='x' )
text( USCrimes[-c(9,45),y,'Population'],
 USCrimes[-c(9,45),y,'PropertyCrimeRate'],
state.abb, ... )
}

tmp.list <- list( Year=list('animate', from=1960, to=2010, delay=250) )

tmpcol <- c('blue','darkgreen','red','purple')[state.region]
tkexamp( tmpfun(col=tmpcol), tmp.list )
}
```

`vis.binom` *Plot various distributions then interactivly adjust the parameters.*

Description

Plot a curve of a distribution, then using a Tk slider window adjust the parameters and see how the distribution changes. Optionally also plots reference distributions.

Usage

```
vis.binom()
vis.gamma()
vis.normal()
vis.t()
```

Details

These functions plot a distribution, then create a Tk slider box that allows you to adjust the parameters of the distribution to see how the curve changes.

Check boxes are available in some cases to also show reference distributions (normal and poisson for the binomial, exponential and chi-squared for gamma, and normal for t). The exponential and chi-squared distributions are those with the same mean as the plotted gamma.

If you change the plotting ranges then you need to click on the 'refresh' button to update the plot.

Value

These functions are run for their side effects and do not return anything meaningful.

Author(s)

Greg Snow <538280@gmail.com>

See Also

dnorm, dgamma, etc.

Examples

```
if(interactive()){
vis.binom()
vis.normal()
vis.gamma()
vis.t()
}
```

---

vis.boxcox *Interactively visualize Box-Cox transformations*

Description

Explore the Box-Cox family of distributions by plotting data transformed and untransformed and interactively choose values for lambda.

Usage

```
vis.boxcox(lambda = sample(c(-1,-0.5,0,1/3,1/2,1,2), 1),
           hscale=1.5, vscale=1.5, wait=FALSE)

vis.boxcoxu(lambda = sample( c(-1,-0.5,0,1/3,1/2,1,2), 1),
                        y, xlab=deparse(substitute(y)),
                        hscale=1.5, vscale=1.5, wait=FALSE)

vis.boxcox.old(lambda = sample(c(-1, -0.5, 0, 1/3, 1/2, 1, 2), 1))

vis.boxcoxu.old(lambda = sample(c(-1, -0.5, 0, 1/3, 1/2, 1, 2), 1))
```

Arguments

lambda	The true value of lambda to use.
y	Optional data to use in the transform.
xlab	Label for x-axis.
hscale	The horizontal scale, passed to tkrplot.
vscale	The vertical scale, passed to tkrplot.
wait	Should R wait for the demo window to close.

Details

These functions will generate a sample of data and plot the untrasformed data (left panels) and the transformed data (right panels). Initially the value of lambda is 1 and the 2 sets of plots will be identical.

You then adjust the transformation parameter lambda to see how the right panels change.

The function vis.boxcox shows the effect of transforming the y-variable in a simple linear regression.

The function vis.boxcoxu shows a single variable compared to the normal distribution.

Value

The old versions have no useful return value. If wait is FALSE then they will return an invisible NULL, if wait is TRUE then the return value will be a list with the final value of lamda, the original data, and the transformed y (at the final lamda value).

Author(s)

Greg Snow <538280@gmail.com>

References

GEP Box; DR Cox. An Analysis of Transformations. Journal of the Royal Statitical Society. Series B, Vol. 26, No. 2 (1964) 211-252

See Also

bct, boxcox in package MASS

Examples

```
if(interactive()) {
vis.boxcoxu()
vis.boxcox()
}
```

vis.test *Do a Visual test of a null hypothesis by choosing the graph that does not belong.*

Description

These functions help in creating a set of plots based on the real data and some modification that makes the null hypothesis true. The user then tries to choose which graph represents the real data.

Usage

```
vis.test(..., FUN, nrow=3, ncol=3, npage=3, data.name = "", alternative)
vt.qqnorm(x, orig=TRUE)
vt.normhist(x, ..., orig=TRUE)
vt.scatterpermute(x, y, ..., orig=TRUE)
vt.tspermute(x, type='l', ..., orig=TRUE)
vt.residpermute(model, ..., orig=TRUE)
vt.residsim(model, ..., orig=TRUE)
```

Arguments

`...`	data and arguments to be passed on to `FUN` or to plotting functions, see details below
`FUN`	The function to create the plots on the original or null hypothesis data
`nrow`	The number of rows of graphs per page
`ncol`	The number of columns of graphs per page
`npage`	The number of pages to use in the testing
`data.name`	Optional character string for the name of the data in the output
`alternative`	Optional character string for the alternative hypothesis in the output
`orig`	Logical, should the original data be plotted, or data based on the null hypothesis
`x`	data or x-coordinates of the data
`y`	y-coordinates of the data
`type`	type of plot, passed on to plot function (use 'p' for points)
`model`	An `lm` object, or any model object for which `fitted` and `resid` return vectors

Details

The `vis.test` function will create a `nrow` by `ncol` grid of plots, one of which is based on the real (original) data and the others which are based on a null hypothesis simulation (a statistical "lineup"). The real plot is placed at random within the set. The user then clicks on their best guess of which plot is the real one (the most different from the others). If the null hypothesis is true for the real data, then this will be a guess with a 1/(`nrow`*`ncol`) probability of success. This process is then repeated for a total of `npage` times. A p-value is then constructed based on the number of correct guesses and the null hypothesis that there is a 1/(`nrow`*`ncol`) chance of guessing correct each time

(this will work best if the person doing the choosing has not already seen plots/summaries of the data).

If the plotting function (FUN) is not passed as a named argument, then the first argument (in the ...) that is a function will be used. If no functions are passed then the function will stop with an error.

The plotting function (FUN) can be an existing function or a user supplied function. The function must have an argument named "orig" which indicates whether to plot the original data or the null hypothesis data. A new seed will be set before each call to FUN except when orig is TRUE. Inside the function if orig is TRUE then the function should plot the original data. When orig is FALSE then the function should do some form of simulation based on the data with the null hypothesis true and plot the simulated data (making sure to give no signs that it is different from the original plot).

The return object includes a list with the seeds set before each of the plots (NA for the original data plot) and a vector of the plots selected by the user. This information can be used to recreate the simulated plots by setting the seed then calling FUN.

The vt.qqnorm function tests the null hypothesis that a vector of data comes from a normal distribution (or at least pretty close) by creating a qqnorm plot of the original data, or the same plot of random data from a normal distribution with the same mean and standard deviation as the original data.

The vt.normhist function tests the null hypothesis that a vector of data comes from a normal distribution (or at least pretty close) by plotting a histogram with a reference line representing a normal distribution of either the original data or a set of random data from a normal distribution with the same mean and standard deviation as the original.

The vt.scatterpermute function tests the null hypothesis of "no relationship" between 2 vectors of data. When orig is TRUE the function creates a scatterplot of the 2 variables, when orig is FALSE the function first permutes the y variable randomly (making no relationship) then creates a scatter plot with the original x and permuted y variables.

The vt.tspermute function creates a time series type plot of a single vector against its index. When orig is false, the vector is permuted before plotting.

The vt.residpermute function takes a regression object (class lm, or any model type object for which fitted and resid return vectors) and does a residual plot of the fitted values on the x axis and residuals on the y axis. The loess smooth curve (scatter.smooth is the plotting function) and a reference line at 0 are included. When orig is FALSE the residuals are randomly permuted before being plotted.

The vt.residsim function takes a regression object (class lm, or any model type object for which fitted and resid return vectors) and does a residual plot of the fitted values on the x axis and residuals on the y axis. The loess smooth curve (scatter.smooth is the plotting function) and a reference line at 0 are included. When orig is FALSE the residuals are simulate from a normal distribution with mean 0 and standard deviation the same as the residuals.

Value

The vis.test function returns an object of class htest with the following components:

method	The string "Visual Test"
data.name	The name of the data passed to the function
statistic	The number of correct "guesses"

p.value	The p-value based on the number of correct "guesses"
nrow	The number of rows per page
ncol	The number of columns per page
npage	The number of pages
seeds	A list with 3 vectors containing the seeds set before calling FUN, the correct plot has an NA
selected	A vector of length npage indicating the number of the figure picked in each of the npage tries

The other functions are run for their side effects and do not return anything meaningful.

Warning

The p-value is based on the assumption that under the null hypothesis there is a 1/(nrow*ncol) chance of picking the correct plot and that the npage choices are independent of each other. This may not be true if the user is familiar with the data or remembers details of the plot between picks.

Author(s)

Greg Snow <538280@gmail.com>

References

Buja, A., Cook, D. Hofmann, H., Lawrence, M. Lee, E.-K., Swayne, D.F and Wickham, H. (2009) Statistical Inference for exploratory data analysis and model diagnostics Phil. Trans. R. Soc. A 2009 367, 4361-4383 doi: 10.1098/rsta.2009.0120

See Also

set.seed

Examples

```
if(interactive()) {
  x <- rexp(25, 1/3)
  vis.test(x, vt.qqnorm)

  x <- rnorm(100, 50, 3)
  vis.test(x, vt.normhist)
}
```

`z.test` *Z test for known population standard deviation*

Description

Compute the test of hypothesis and compute confidence interval on the mean of a population when the standard deviation of the population is known.

Usage

```
z.test(x, mu = 0, stdev, alternative = c("two.sided", "less", "greater"),
  sd = stdev, n=length(x), conf.level = 0.95, ...)
```

Arguments

`x`	Vector of data values or the mean of the data.
`mu`	Hypothesized mean of the population.
`stdev`	Known standard deviation of the population.
`alternative`	Direction of the alternative hypothesis.
`sd`	Alternative to `stdev`
`n`	The sample size if `x` is the sample mean.
`conf.level`	Confidence level for the interval computation.
`...`	Additional arguments are silently ignored.

Details

Many introductory statistical texts introduce inference by using the Z test and Z based confidence intervals based on knowing the population standard deviation. Most statistical packages do not include functions to do Z tests since the T test is usually more appropriate for real world situations. This function is meant to be used during that short period of learning when the student is learning about inference using Z procedures, but has not learned the T based procedures yet. Once the student has learned about the T distribution the `t.test` function should be used instead of this one (but the syntax is very similar, so this function should be an appropriate introductory step to learning `t.test`).

Value

An object of class `htest` containing the results

Note

This function should be used for learning only, real data should generally use `t.test`.

Author(s)

Greg Snow <538280@gmail.com>

See Also

t.test, print.htest

Examples

```
x <- rnorm(25, 100, 5)
z.test(x, 99, 5)
```

---

zoomplot *Zoom or unzoom an existing plot in the plot window*

Description

This function allows you to change the x and y ranges of the plot that is currently in the plot window. This has the effect of zooming into a section of the plot, or zooming out (unzooming) to show a larger region than is currently shown.

Usage

```
zoomplot(xlim, ylim=NULL )
oldzoomplot(xlim, ylim=NULL )
```

Arguments

xlim, ylim	The new x and y limits of the plot. These can be passed in in any form understood by xy.coords. The range of xlim and ylim are actually used so you can pass in more than 2 points.

Details

This function recreates the current plot in the graphics window but with different xlim, ylim arguments. This gives the effect of zooming or unzooming the plot.

This only works with traditional graphics (not lattice/trellis).

This function is a quick hack that should only be used for quick exploring of data. For any serious work you should create a script with the plotting commands and adjust the xlim and ylim parameters to give the plot that you want.

Only the x and y ranges are changed, the size of the plotting characters and text will stay the same.

The oldzoomplot function is the version that worked for 2.15 and earlier, zoomplot should be used for R 3.0.

Value

This function is run for its side effects and does not return anything meaningful.

Note

For any serious projects it is best to put your code into a script to begin with and edit the original script rather than using this function.

This function works with the standard `plot` function and some others, but may not work for more complicated plots.

This function depends on the `recordPlot` function which can change in any version. Therefore this function should not be considered stable.

Author(s)

Greg Snow <538280@gmail.com>

See Also

`plot.default`, `par`, `matplot`, `plot2script`, `source`

Examples

```
if(interactive()){

with(iris, plot(Sepal.Length, Petal.Width,
    col=c('red','green','blue')[Species]))
text( 6.5, 1.5, 'test' )
zoomplot( locator(2) ) # now click on 2 points in the plot to zoom in

plot( 1:10, rnorm(10) )
tmp <- rnorm(10,1,3)
lines( (1:10) + 0.5, tmp, col='red' )
zoomplot( c(0,11), range(tmp) )
}
```

`%<%` *Less than or Less than and equal operators that can be chained together.*

Description

Comparison operators that can be chained together into something like 0 %<% x %<% 1 instead of 0 < x && x < 1.

Usage

```
x %<% y
x %<=% y
```

Arguments

`x,y`	Values to compare

Details

These functions/operators allow chained inequalities. To specify that you want the values between two values (say 0 and 1) you can use `0 %<% x %<% 1` rather than `0 < x \&\& x < 1` .

Value

A logical vector is returned that can be used for subsetting like <, but the original values are included as attributes to be used in additional comparisons.

Note

This operator is not fully associative and has different precedence than < and <=, so be careful with parentheses. See the examples.

Author(s)

Greg Snow, <538280@gmail.com>

Examples

```
x <- -3:3

 -2 %<% x %<% 2
c( -2 %<% x %<% 2 )
x[ -2 %<% x %<% 2 ]
x[ -2 %<=% x %<=% 2 ]


x <- rnorm(100)
y <- rnorm(100)

x[ -1 %<% x %<% 1 ]
range( x[ -1 %<% x %<% 1 ] )


cbind(x,y)[ -1 %<% x %<% y %<% 1, ]
cbind(x,y)[ (-1 %<% x) %<% (y %<% 1), ]
cbind(x,y)[ ((-1 %<% x) %<% y) %<% 1, ]
cbind(x,y)[ -1 %<% (x %<% (y %<% 1)), ]
cbind(x,y)[ -1 %<% (x %<% y) %<% 1, ] # oops


3
3
```

Index