Package 'aplpack'

February 19, 2015

Type Package

Title Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls, and some slider functions

Version 1.3.0

Date 2014-09-26

Author Hans Peter Wolf, Uni Bielefeld

Maintainer Hans Peter Wolf <pwolf@wiwi.uni-bielefeld.de>

Depends R (>= 2.8.0), tcltk

Suggests tkrplot

Description set of functions for drawing some special plots:
stem.leaf plots a stem and leaf plot,
stem.leaf.backback plots back-to-back versions of stem and leafs,
bagplot plots a bagplot,
skyline.hist plots several histgramm in one plot of a one dimensional data set,
plotsummary plots a graphical summary of a data set with one or more variables,
plothulls plots sequentially hulls of a bivariate data set,
faces plots chernoff faces,
spin3R for an inspection of a 3-dim point cloud,
slider functions for interactive graphics.

License GPL (>= 2)

URL http://www.wiwi.uni-bielefeld.de/com/wolf/software/aplpack.html

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-26 18:50:21

R topics documented:

`bagplot` *bagplot, a bivariate boxplot*

Description

`compute.bagplot()` computes an object describing a bagplot of a bivariate data set. `plot.bagplot()` plots a bagplot object. `bagplot()` computes and plots a bagplot.

Usage

```
bagplot(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
       show.outlier = TRUE, show.whiskers = TRUE,
       show.looppoints = TRUE, show.bagpoints = TRUE,
       show.loophull = TRUE, show.baghull = TRUE,
       create.plot = TRUE, add = FALSE, pch = 16, cex = 0.4,
       dkmethod = 2, precision = 1, verbose = FALSE,
       debug.plots = "no",   col.loophull="#aaccff",
       col.looppoints="#3355ff", col.baghull="#7799ff",
       col.bagpoints="#000088", transparency=FALSE, ...
)
compute.bagplot(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
       dkmethod=2,precision=1,verbose=FALSE,debug.plots="no")
## S3 method for class 'bagplot'
plot(x,
       show.outlier = TRUE, show.whiskers = TRUE,
       show.looppoints = TRUE, show.bagpoints = TRUE,
       show.loophull = TRUE, show.baghull = TRUE,
       add = FALSE, pch = 16, cex = 0.4, verbose = FALSE,
       col.loophull="#aaccff", col.looppoints="#3355ff",
       col.baghull="#7799ff", col.bagpoints="#000088",
       transparency=FALSE,...)
```

Arguments

x	x values of a data set; in bagplot: an object of class bagplot computed by compute.bagplot
y	y values of the data set
factor	factor defining the loop
na.rm	if TRUE 'NA' values are removed otherwise exchanged by median
approx.limit	if the number of data points exceeds approx.limit a sample is used to compute some of the quantities; default: 300
show.outlier	if TRUE outlier are shown
show.whiskers	if TRUE whiskers are shown
show.looppoints	if TRUE loop points are plottet
show.bagpoints	if TRUE bag points are plottet
show.loophull	if TRUE the loop is plotted
show.baghull	if TRUE the bag is plotted
create.plot	if FALSE no plot is created
add	if TRUE the bagplot is added to an existing plot
pch	sets the plotting character
cex	sets characters size
dkmethod	1 or 2, there are two method of approximating the bag, method 1 is very rough (only based on observations
precision	precision of approximation, default: 1
verbose	automatic commenting of calculations
debug.plots	if TRUE additional plots describing intermediate results are constructed
col.loophull	color of loop hull
col.looppoints	color of the points of the loop
col.baghull	color of bag hull
col.bagpoints	color of the points of the bag
transparency	see section details
...	additional graphical parameters

Details

A bagplot is a bivariate generalization of the well known boxplot. It has been proposed by Rousseeuw, Ruts, and Tukey. In the bivariate case the box of the boxplot changes to a convex polygon, the bag of bagplot. In the bag are 50 percent of all points. The fence separates points within the fence from points outside. It is computed by increasing the the bag. The loop is defined as the convex hull containing all points inside the fence. If all points are on a straight line you get a classical boxplot. bagplot() plots bagplots that are very similar to the one described in Rousseeuw et al. Remarks: The two dimensional median is approximated. For large data sets the error will be very small. On the other hand it is not very wise to make a (graphical) summary of e.g. 10 bivariate data points. In

case you want to plot multiple (overlapping) bagplots, you may want plots that are semi-transparent. For this you can use the `transparency` flag. If `transparency==TRUE` the alpha layer is set to '99' (hex). This causes the bagplots to appear semi-transparent, but ONLY if the output device is PDF and opened using: `pdf(file="filename.pdf", version="1.4")`. For this reason, the default is `transparency==FALSE`. This feature as well as the arguments to specify different colors has been proposed by Wouter Meuleman.

Value

`compute.bagplot` returns an object of class `bagplot` that could be plotted by `plot.bagplot()`. An object of the bagplot class is a list with the following elements: `center` is a two dimensional vector with the coordinates of the center. `hull.center` is a two column matrix, the rows are the coordinates of the corners of the center region. `hull.bag` and `hull.loop` contain the coordinates of the hull of the bag and the hull of the loop. `pxy.bag` shows you the coordinates of the points of the bag. `pxy.outer` is the two column matrix of the points that are within the fence. `pxy.outlier` represent the outliers. The vector `hdepths` shows the depths of data points. `is.one.dim` is `TRUE` if the data set is (nearly) one dimensional. The dimensionality is decided by analysing the result of `prcomp` which is stored in the element `prdata`. `xy` shows you the data that are used for the bagplot. In the case of very large data sets subsets of the data are used for constructing the bagplot. A data set is very large if there are more data points than `approx.limit`. `xydata` are the input data structured in a two column matrix.

Note

Version of bagplot: 10/2012

Author(s)

Peter Wolf

References

P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999): The bagplot: a bivariate boxplot, The American Statistician, vol. 53, no. 4, 382–387

See Also

boxplot

Examples

```
# example: 100 random points and one outlier
dat<-cbind(rnorm(100)+100,rnorm(100)+300)
dat<-rbind(dat,c(105,295))
bagplot(dat,factor=2.5,create.plot=TRUE,approx.limit=300,
   show.outlier=TRUE,show.looppoints=TRUE,
   show.bagpoints=TRUE,dkmethod=2,
   show.whiskers=TRUE,show.loophull=TRUE,
   show.baghull=TRUE,verbose=FALSE)
# example of Rousseeuw et al., see R-package rpart
cardata <- structure(as.integer( c(2560,2345,1845,2260,2440,
```

```
 2285, 2275, 2350, 2295, 1900, 2390, 2075, 2330, 3320, 2885,
 3310, 2695, 2170, 2710, 2775, 2840, 2485, 2670, 2640, 2655,
 3065, 2750, 2920, 2780, 2745, 3110, 2920, 2645, 2575, 2935,
 2920, 2985, 3265, 2880, 2975, 3450, 3145, 3190, 3610, 2885,
 3480, 3200, 2765, 3220, 3480, 3325, 3855, 3850, 3195, 3735,
 3665, 3735, 3415, 3185, 3690, 97, 114, 81, 91, 113, 97, 97,
 98, 109, 73, 97, 89, 109, 305, 153, 302, 133, 97, 125, 146,
 107, 109, 121, 151, 133, 181, 141, 132, 133, 122, 181, 146,
 151, 116, 135, 122, 141, 163, 151, 153, 202, 180, 182, 232,
 143, 180, 180, 151, 189, 180, 231, 305, 302, 151, 202, 182,
 181, 143, 146, 146)), .Dim = as.integer(c(60, 2)),
 .Dimnames = list(NULL, c("Weight", "Disp.")))
bagplot(cardata,factor=3,show.baghull=TRUE,
  show.loophull=TRUE,precision=1,dkmethod=2)
title("car data Chambers/Hastie 1992")
# points of y=x*x
bagplot(x=1:30,y=(1:30)^2,verbose=FALSE,dkmethod=2)
# one dimensional subspace
bagplot(x=1:100,y=1:100)
```

`bagplot.pairs` `pairs` *plot with bagplots*

Description

`bagplot.pairs` calls `pairs` and use bagplot() as panel function. It can be used for the inspection of data matrices.

Usage

```
bagplot.pairs(dm, trim = 0.0, main, numeric.only = TRUE,
              factor = 3, approx.limit = 300, pch = 16,
              cex = 0.8, precision = 1, col.loophull = "#aaccff",
              col.looppoints = "#3355ff", col.baghull = "#7799ff",
              col.bagpoints = "#000088", ...)
```

Arguments

`dm`	datamatrix, columns contain values of the variables
`trim`	fraction or vector of fractions of data points that should be removed from the variables before computing
`main`	title of the plot
`numeric.only`	if TRUE only numerical variables will be used. Otherwise an transformation to numeric will be performed.
`factor`	see help of bagplot
`approx.limit`	see help of bagplot
`pch`	see help of bagplot

cex	see help of bagplot
precision	see help of bagplot
col.loophull	see help of bagplot
col.looppoints	see help of bagplot
col.baghull	see help of bagplot
col.bagpoints	see help of bagplot
...	further arguments to be passed to pairs

Details

bagplot.pairs is a cover function which calls pairs and uses bagplot to display the data.

Value

The data which has been used for the plot.

Note

Feel free to have a look inside of bagplot.pairs and to improve it according to your ideas.

Author(s)

Peter Wolf

See Also

bagplot, pairs

Examples

```
# bagplot.pairs(freeny)
 # bagplot.pairs(trees,col.baghull="green", col.loophull="lightgreen")
```

---

boxplot2D *Boxplot of projection of two dimensional data*

Description

boxplot2D computes summary statistics of a one dimensional projection of a two dimensional data set and plots a sloped boxplot of the statistics into the scatterplot of the two dimensional data set.

Usage

```
boxplot2D(xy, add.to.plot = TRUE, box.size = 10, box.shift = 0,
angle = 0, angle.type = "0", tukey.style = TRUE, coef.out = 1.5,
coef.h.out = 3, design = "sl", na.rm=FALSE, ...)
```

Arguments

xy	(nx2)-matrix, two dimensional data set
add.to.plot	if TRUE the boxplot is added to the actual plot of the graphics device
box.size	height of the box (of the boxplot)
box.shift	shift of boxplot perpendicular to the projection direction
angle	direction of projection in units defined by angle.type
angle.type	"0": angle in (0,2*pi), "1": clock-like: angle.typ.0==2*pi*angle.typ.1/12, "2": degrees: angle.typ.0==2*pi*angle.typ.2/360, "3": by fraction: delta.y/delta.x
tukey.style	if TRUE outliers are defined as described in Tukey (1977)
coef.out	outliers are values that are more than coef.out*boxwidth away from the box, default: coef.out=1.5
coef.h.out	heavy outliers are values that are more than coef.h.out*boxwidth away from the box, default: coef.h.out=3
design	if sl then parallelogram else box
na.rm	if TRUE 'NA' values are removed otherwise exchanged by mean
...	additional graphical parameters

Note

version 08/2003

Author(s)

Peter Wolf

References

Tukey, J. *Exploratory Data Analysis.* Addison-Wesley, 1977.

See Also

boxplot

Examples

```
xy<-cbind(1:100, (1:100)+rnorm(100,,5))
par(pty="s")
plot(xy,xlim=c(-50,150),ylim=c(-50,150))
boxplot2D(xy,box.shift=-30,angle=3,angle.typ=1)
boxplot2D(xy,box.shift=20,angle=1,angle.typ=1)
boxplot2D(xy,box.shift=50,angle=5,angle.typ=1)
par(pty="m")
```

faces *Chernoff Faces*

Description

faces represent the rows of a data matrix by faces. plot.faces plots faces into a scatterplot.

Usage

```
faces(xy, which.row, fill = FALSE, face.type = 1, nrow.plot, ncol.plot,
    scale = TRUE, byrow = FALSE, main, labels, print.info = TRUE,
    na.rm = FALSE, ncolors = 20, col.nose = rainbow(ncolors),
    col.eyes = rainbow(ncolors, start = 0.6, end = 0.85),
    col.hair = terrain.colors(ncolors), col.face = heat.colors(ncolors),
    col.lips = rainbow(ncolors, start = 0, end = 0.2),
    col.ears = rainbow(ncolors, start = 0, end = 0.2), plot.faces = TRUE)
## S3 method for class 'faces'
plot(x, x.pos, y.pos, face.type = 1, width = 1, height = 1, labels,
        ncolors = 20, col.nose = rainbow(ncolors), col.eyes = rainbow(ncolors,
        start = 0.6, end = 0.85), col.hair = terrain.colors(ncolors),
        col.face = heat.colors(ncolors), col.lips = rainbow(ncolors,
        start = 0, end = 0.2), col.ears = rainbow(ncolors, start = 0,
        end = 0.2), ...)
```

Arguments

xy	xy data matrix, rows represent individuals and columns variables
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
face.type	an integer between 0 and 2 with the meanings: 0 = line drawing faces, 1 = the elements of the faces are painted, 2 = Santa Claus faces are drawn
nrow.plot	number of columns of faces on graphics device
ncol.plot	number of rows of faces
scale	if(scale==TRUE), variables will be normalized
byrow	if(byrow==TRUE), xy will be transposed
main	title
labels	character strings to use as names for the faces
print.info	if TRUE information about usage of variables for face elements are printed
na.rm	if TRUE 'NA' values are removed otherwise exchanged by mean of data
plot.faces	if FALSE no face is plotted
x	an object of class faces computed by faces
x.pos	x coordinates of positions of faces

y.pos	y coordinates of positions of faces
width	width of the faces
height	height of the faces
ncolors	number of colors in the palettes for painting the elements of the faces
col.nose	palette of colors for painting the nose
col.eyes	palette of colors for painting the eyes
col.hair	palette of colors for painting the hair
col.face	palette of colors for painting the face
col.lips	palette of colors for painting the lips
col.ears	palette of colors for painting the ears
...	additional graphical arguments

Details

Explanation of parameters: 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 painting elements of a face the colors of are found by averaging of sets of variables: (7,8)-eyes:iris, (1,2,3)-lips, (14,15)-ears, (12,13)-nose, (9,10,11)-hair, (1,2)-face.

Further details can be found in the literate program of `faces`.

Value

list of two elements: The first element `out$faces` is a list of standardized faces of `class faces`, this object could be plotted by plot.faces; a plot of faces is created on the graphics device if `plot.faces=TRUE`. The second list is short description of the effects of the variables.

Note

version 01/2009

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.wiwi.uni-bielefeld.de/mitarbeiter/wolf/ : S/R - functions : faces

See Also

—

Examples

```
faces()
faces(face.type=1)

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

data(longley)
faces(longley[1:9,],face.type=0)
faces(longley[1:9,],face.type=1)

plot(longley[1:16,2:3],bty="n")
a<-faces(longley[1:16,],plot=FALSE)
plot.faces(a,longley[1:16,2],longley[1:16,3],width=35,height=30)

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

a<-faces(rbind(1:3,5:3,3:5,5:7),plot.faces=FALSE)
plot(0:5,0:5,type="n")
plot(a,x.pos=1:4,y.pos=1:4,1.5,0.7)
# during Christmastime
faces(face.type=2)
```

---

hdepth *hdepth of points*

Description

`hdepth()` computes the h-depths of points.

Usage

```
hdepth(tp, data, number.of.directions=181)
```

Arguments

`tp`	two column matrix of the coordinates of points which h-depths are needed
`data`	two column matrix of the coordinates of the points of a data set
`number.of.directions`	number of directions to be checked

Details

The function `hdepth` computes the h-depths of the points `tp` relative to data set `data`. If `data` is missing `tp` will also be taken as data set.

Value

the h-depths of the test points

Note

Version of bagplot: 12/2012

Author(s)

Peter Wolf

See Also

bagplot

Examples

```
# computation of h-depths
data <- cbind(rnorm(40), rnorm(40))
xy <- cbind(runif(50,-2,2),runif(50,-2,2))
bagplot(data); text(xy, as.character(hdepth(xy,data)))
```

---

plothulls *plothulls for data peeling*

Description

plothulls plots convex hulls of a bivariate data set.

Usage

```
plothulls(x, y, fraction, n.hull = 1, main, add = FALSE, col.hull,
    lty.hull, lwd.hull, density = 0, ...)
```

Arguments

x	two column matrix of the coordinates of points of x-values of a data set
y	if x is one dimensional then y contains the y-values of the data set
fraction	... of points that lies inside the hull to be plotted
n.hull	number of directions sequential hulls to be plotted
main	title for the graphics
add	if TRUE no new plot is initialized
col.hull	color(s) of the hull(s)
lty.hull	line type(s) of the hull(s)
lwd.hull	line width(s) of the hull(s)
density	density argument of polygon() that draws the hulls
...	further arguments used in the call of plot() or points()

Details

The function `plothulls` computes hulls of a bivariate data set using the function `chull`. After finding a hull the hull maybe plotted. Then the data points of the hull will be removed and the hull of the remaining points is computed. The style of plotting a hull depends on the setting of `col.hull`, `lty.hull`, `lwd.hull` and `density`. `density=NA` has the effect that the regions of the hulls are filled by a color. Using `fraction` you can plot a single hull. `n.hull` defines the number of hull that should be drawn one after the other.

Value

The hull(s) are stored as a list of matrices with two columns, the innermost first and so on.

Note

Version of plothulls: 10/2013

Author(s)

Peter Wolf

References

Green, P.J. (1981): Peeling bivariate data. In: Interpreting Multivariate Data, V. Barnett (ed.), pp 3-19, Wiley. Porzio, Giovanni C., Ragozini, Giancarlo (2000): Peeling multvariate data sets: a new approach. Quanderni di Statistica, Vol. 2.

See Also

`bagplot`

Examples

```
# 10 hulls computed from the faithful data and plotted
plothulls(faithful, n.hull=10, lty.hull=1)
# plotting additionally a hull with 90 percent of points within the hull
plot(faithful)
plothulls(faithful, fraction=.90, add=TRUE, col.hull="red", lwd.hull=3)
# hull with 10 percent of points within the hull
plothulls(faithful, fraction=.10, col.hull="red", lwd.hull=3)
# first 3 hulls of the cars data set
n <- 3
plothulls(cars, n.hull=n, col.hull=1:n, lty.hull=1:n)
# 5 hulls represented by colored regions
n <- 5
cols <- heat.colors(9)[3:(3+n-1)]
plothulls(cars, n.hull=n, col.hull=cols, lty.hull=1:n, density=NA, col=0)
points(cars, pch=17, cex=1)
# 6 hulls: regions colored and boundaries shown
n <- 6
cols <- rainbow(n)
plothulls(cars, n.hull=n, col.hull=cols, lty.hull=1:n, density=NA, col=0)
```

```
plothulls(cars, n.hull=n, add=TRUE, col.hull=1, lwd.hull=2, lty=1, col=0)
```

`plotsummary` *graphical summaries of variables of a data set*

Description

`plotsummary` shows some important characteristics of the variables of a data set. For each variable a plot is computed consisting of a barplot, an ecdf, a density trace and a boxplot.

Usage

```
plotsummary(data, trim = 0, types = c("stripes", "ecdf", "density", "boxplot"),
            y.sizes = 4:1, design = "chessboard", main, mycols = "RB")
```

Arguments

`data`	Data set for computing a graphical summary.
`trim`	`trim` defines the fraction of observation for trimming on both ends of the data.
`types`	vector of types of representation of the data set. The elements of the vector will induce small plots which are stacked in vertical order. The first letter of the types is sufficient for defining a type.
`y.sizes`	defines the relative sizes of the small plots. The values are divided by their sum to get percentages.
`design`	if `design` is `chessboard` the graphics device is fragmented into rows and cols. Otherwise the images of a variable build vertical stripes.
`main`	defines a title for the graphics.
`mycols`	allows to define some colors for the showing the regions separated by the quartils.

Details

`plotsummary` can be use for a quick and dirty inspection of a data matrix or a list of variables. Without further specification some representation of each of the variables is built and stacked into a plot. The sizes of the types of representation can be set as well as the layout design of the graphics device. It is helpful to trim the data before processing because outliers will often hide the interesting characteristics.

Author(s)

Peter Wolf, pwolf@wiwi.uni-bielefeld.de

See Also

`pairs`, `summary`, `str`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--\tor do  help(data=index)  for the standard data sets.
plotsummary(cars)
plotsummary(cars, types=c("ecdf", "density", "boxplot"),
            y.sizes = c(1,1,1), design ="stripes")
plotsummary(c(list(rivers=rivers, co2=co2), cars), y.sizes=c(10,3,3,1), mycols=3)
plotsummary(cars, design="chessboard")
# find all matrices in your R
ds.of.R <- function(type="vector"){
  dat <- ls(pos=grep("datasets",search()))
  dat.type <- unlist(lapply(dat,function(x) {
     num <- mode(x<-eval(parse(text=x)))
     num <- ifelse(is.array(x),"array",num)
     num <- ifelse(is.list(x),"list",num)
     num <- ifelse(is.matrix(x),"matrix",num)
     num <- ifelse(is.data.frame(x),"matrix",num)
     num <- ifelse(num=="numeric","vector",num)
     num }))
  return(dat[dat.type==type])
}
namelist <- ds.of.R("matrix")
# inspect the matrices one after the other
for(i in seq(along=namelist)){
  print(i); print(namelist[i])
  xy <- get(namelist[i])
  # plotsummary(xy,y.sizes=4:1,trim=.05,main=namelist[i])
  # Sys.sleep(1)
}
```

---

skyline.hist skyline.hist *computes a skyline plot which is special histogram.*

Description

The function skyline.hist draws several histograms in one plot. The resulting image may look like a skyline.

Usage

```
skyline.hist(x, n.class, n.hist = 1, main, ylab="density",
             night = FALSE, col.bars = NA, col.border = 4, lwd.border = 2.5,
            n.shading = 6, lwd.shading = 2, col.shading = NA, lty.shading = 3,
            pcol.data = "green", cex.data = 0.3, pch.data = 16, col.data = 1,
             lwd.data = .2, permutation = FALSE,
             xlab, xlim, ylim, new.plot=TRUE, bty="n", ...)
```

Arguments

x	one dimensional data set.
n.class	number of classes that should be used to find the width of the bars of the histogram(s).
n.hist	number of histograms that should be plotted.
main	used for call of title.
ylab	text for y axis.
night	If TRUE the background will be colored blue. If FALSE there will be no colored background. Otherwise night is used as background color.
col.bars	defines the color of the bars. If is.na(col.bars) and night==TRUE the bars will be colored gray.
col.border	color of the borders of the bars.
lwd.border	line width of the borders of the bars.
n.shading	number of vertical lines for filling the bars of the histograms.
lwd.shading	line width of the vertical lines for shading the bars.
col.shading	color for the vertical lines for shading. If NA heat colors are used.
lty.shading	line type for the vertical lines for shading.
pcol.data	color of data points.
cex.data	character size of plotting character.
pch.data	plotting character of data points.
lwd.data	line width for segments between data points.
col.data	color for segments between data points.
permutation	if not FALSE a permutation of the data set is erformed.
xlab	text for y axis.
xlim	range of x.
ylim	range of y.
new.plot	logical. If TRUE a new plot is constructed.
bty	box type, used by plot.
...	further graphical parameters passed to plot.

Details

skyline.hist computes several histograms and plots them one upon the other. The histograms differ in the positions of the first cells, but all cells have the same width. The parameters n.class and n.hist have the greatest effect on the design of the result. col.border allows to color the border of the rectangular boxes of the histogram bars. col.bars defines the fill color of the bars. n.shading defines the number of vertical lines of type lty.shading and width lwd.shading that are drawn within the boxes. Another feature of skyline.hist is to represent the data points. The data points of a cell are plotted according their x-values and their ranks (within the points of the cell). The resulting points are connected by line segments and you will see a time series running from bottom to top in each cell. The points and lines can be specified by pcol.data, cex.data,

pch.data, lwd.data, col.data. To get rid of the original order of the data you can permutated them (permutation=1). The "skyline" of the plot may be similar to the skyline of a town and the vertical lines may look like small windows of buildings. In Young et. al. you find "shaded histograms". These histograms have triggered the idea of skyline.hist and the representation of a one dimensional data set by laying histograms on top of otheroverlied histograms.

Value

The result of a call of hist is returned.

Author(s)

Peter Wolf, pwolf@wiwi.uni-bielefeld.de

References

F.W. Young, R.M. Valero-Mora, M. Friendly (2006): Visual Statistics. Wiley, p207–208.

See Also

hist, density

Examples

```
par(mfrow=c(3,3))
for(n.c in c(2,4,8)){  # some values for n.class
  for(n.h in c(2,4,3)){  # some values for number of n.hist
      n.s <- 9         # value for number of vertical lines
      skyline.hist(co2, n.shading=n.s, n.hist=n.h ,n.class=n.c,
                  night=n.h==3, col.border=n.h!=4)
  }
}
skyline.hist(x=rivers, n.class=4, n.hist=2, n.shading=0, main="rivers",
           cex.data=.5, lwd.data = .2, col.data = "green", pcol.data = "red",
           col.border=NA, night=FALSE, ylab="density")
skyline.hist(x=rivers, n.class=4, n.hist=5, n.shading=0, main="rivers",
           cex.data=.5, lwd.data = 1, col.data = "green", pcol.data = "red",
           col.border=NA, night="blue" , ylab="density", col.bars =NA)
skyline.hist(x=rivers, n.class=10, n.hist=2, n.shading=0, main="rivers",
           cex.data=.5, lwd.data = 1, col.data = "green", pcol.data = "red",
           col.border=NA, night=FALSE , ylab="density", col.bars = "lightblue")
skyline.hist(x=rivers, n.class=10, n.hist=1, n.shading=0, main="rivers",
           cex.data=1, lwd.data = 0, col.data = "green", pcol.data = "red",
           col.border=NA, night=FALSE , ylab="density", col.bars = "lightblue" )
skyline.hist(x=rivers, n.class=6, n.hist=1, n.shading=0, main="rivers",
           cex.data=0.1, lwd.data = 2, col.data = "red", pcol.data = "green",
           night="orange" , ylab="density", col.bars = "white", col.border=1 )
skyline.hist(x=rivers, n.class=6, n.hist=1, n.shading=0, main="rivers",
           cex.data=0.1, lwd.data = 2, col.data = "red", pcol.data = "green",
           col.border=NA, night=FALSE , ylab="density", col.bars = "lightblue")
skyline.hist(x=rivers, n.class=6, n.hist=1, n.shading=5, col.shading = "blue",
           main="rivers",
```

```
            cex.data=0.1, lwd.data = 1, col.data = "black", pcol.data = "green",
            col.border=NA, night=FALSE , ylab="density", col.bars = "green")
  skyline.hist(x=rivers, n.class=6, n.hist=3, n.shading=5, col.shading = "blue",
            main="rivers", col.bars = "green",
            cex.data=0.1, lwd.data = 1, col.data = "black", pcol.data = "green",
            col.border="white", night="magenta" , ylab="density")
  skyline.hist(x=rivers, n.class=6, n.hist=4, n.shading=5, col.shading = "blue",
            main="rivers",
            cex.data=0.8, lwd.data = 1, col.data = "blue", pcol.data = "red",
            col.border=NA, night=FALSE , ylab="density", col.bars = "green")
```

`slider` *slider / button control widgets*

Description

`slider` and `gslider` construct a Tcl/Tk-widget with sliders and buttons to demonstrate the effects of variation of parameters on calculations and plots.

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, prompt=FALSE,
  sliders.frame.vertical=TRUE)

gslider(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, prompt=FALSE,
  sliders.frame.vertical=TRUE, hscale=1, vscale=1,
  pos.of.panel = c("bottom","top","left","right")[1])
```

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 induce a reset.button and contains the commands of it.
title	title of the control window
prompt	if TRUE slider functions are called by moving a slider, if FALSE slider functions are called after releasing the mouse button
sliders.frame.vertical	if TRUE the sliders are stacked one above the other; otherwise they are positioned side by side
hscale	horizontal scale factor for image size; compare tkrplot in package tkrplot
vscale	vertical scale factor for image size; compare tkrplot in package tkrplot
pos.of.panel	position of the panel field for sliders and buttons. Value of pos.of.panel: bottom, top, left or right.

Details

slider constructs a separated panel for controlling the parameters whereas gslider integrates a graphical device and buttons and sliders within one window.

The following actions can be done: a) definition of (multiple) sliders and buttons, b) request or specification of slider values, and c) request or specification of variables in the environment slider.env. The 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 contains 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 to assign value 1-slider(no=2) to the variable p.save . slider(obj.name=p.save) returns the value of variable p.save.

Dependencies The function gslider depends on package tkrplot.

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`. `gslider` return in definition mode the result of `tkrplot` which was called to construct the widget.

Author(s)

Hans Peter Wolf

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=2); 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(obj.name="type",obj.value="l")
 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"))
}
plot.sample.norm.2()

## End(Not run)
```

```
# example 2a, sliders and buttons and graphics in one window
## 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(obj.name="type",obj.value="l")
 gslider(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"))
}
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
  refresh.code<-function(...){
    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
     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)
```

`slider.bootstrap.lm.plot`

interactive bootstapping for lm

Description

`slider.bootstrap.lm.plot` computes a scatterplot and adds regression curves of samples of the data points. The number of samples and the degree of the model are controlled by sliders.

Usage

```
slider.bootstrap.lm.plot(x, y, ...)
```

Arguments

`x`	two column matrix or vector of x values if y is used
`y`	y values if x is not a matrix
`...`	additional graphics parameters

Details

`slider.bootstrap.lm.plot` draws a scatterplot of the data points and fits a linear model to the data set. Regression curves of samples of the data are then added to the plot. Within a Tcl/Tk control widget the degree of the model, the repetitions and the start of the random seed are set. After modification of a parameter the plot is updated.

Value

a message about the usage

Author(s)

Hans Peter Wolf

References

~~

See Also

plot

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  daten<-iris[,2:3]
  slider.bootstrap.lm.plot(daten)

## End(Not run)
```

---

slider.brush *interactive brushing functions*

Description

These functions compute a pairs plot or a simple xy-plot and open a slider control widget for brushing.

`slider.brush.pairs` computes a pairs plot; the user defines an interval for one of the variables and in effect all data points in this interval will be recolored.

`slider.brush.plot.xy` computes an xy-plot; the user defines a interval for a third variable `z` and all points `(x,y)` will be recolored red if the `z` value is in the interval.

Usage

```
slider.brush.pairs(x, ...)
slider.brush.plot.xy(x, y, z, ...)
```

Arguments

`...`	new settings for global graphics parameters
`x`	matrix or data frame or vector
`y`	vector of y values if `x` is not a matrix
`z`	vector of z values if `x` is not a matrix

Details

slider.brush.pairs draws a pairs plot of the data set x. The first slider defines the lower limit of the interval and the second its width. By the third slider a variable is selected. All data points for which the selected variable is in the interval are recolored red.

slider.brush.plot.xy draws an xy-plot of the data set x. The first slider defines the lower limit of the interval of z values and the second one its width. All data points for which the variable z is in the interval are recolored red.

Value

a message about the usage

Author(s)

Hans Peter Wolf

References

W. S. Cleveland, R. A. Becker, and G. Weil. The Use of Brushing and Rotation for Data Analysis. InW. S. Cleveland and M. E. McGill, editors, Dynamic Graphics for Statistics. Wadsworth and Brooks/Cole, Pacific Grove, CA, 1988.

See Also

pairs, plot

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.brush.pairs(iris)

## End(Not run)
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.brush.plot.xy(iris[,1:3])

## End(Not run)
```

---

slider.hist *interactive histogram and density traces*

Description

The functions slider.hist and slider.density compute histograms and density traces whereas some parameter are controlled by sliders.

slider.hist computes a histogram; the number of classes is defined by a slider.

slider.density computes a density trace; width and type of the kernel are defined by sliders.

Usage

```
slider.hist(x, panel, ...)
slider.density(x, panel, ...)
```

Arguments

x	data set to be used for plotting
panel	function constructing additional graphical elements to the plot
...	additional (graphics) parameters which are passed to the invoked high level plotting function

Details

`slider.hist` draws a histogram of the data set `x` by calling `hist` and opens a Tcl/Tk widget with one slider. The slider defines the number of classes of the histogram. Changing the slider results in redrawing of the plot. For further details see the help page of `hist`. `rug` is used as the default panel function.

`slider.density` draws a density trace of the data set `x` by `plot(density(...))` and opens a Tcl/Tk-widget with two sliders. The first slider defines the width of the density trace and the second one the kernel function: `"1-gaussian", "2-epanechnikov", "3-rectangular", "4-triangular","5-biweigh`
Changing one of the sliders results in a redrawing of the plot. For further details see the help page of `density`. `rug` is used as the default panel function.

Value

a message about the usage

Author(s)

Hans Peter Wolf

References

~~

See Also

`hist`, `slider`

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.hist(log(islands))

## End(Not run)
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
slider.density(rivers,xlab="rivers",col="red")
```

```
## End(Not run)
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
slider.density(log(rivers),xlab="rivers",col="red",
  panel=function(x){
    xx<-seq(min(x),max(x),length=100)
    yy<-dnorm(xx,mean(x),sd(x))
    lines(xx,yy)
    rug(x)
    print(summary(yy))
  }
)

## End(Not run)
```

`slider.lowess.plot` *interactive lowess smoothing*

Description

`slider.lowess.plot` computes an xy-plot of the data and adds LOWESS lines. The smoother span and the number of iterations are selected by sliders.

Usage

```
slider.lowess.plot(x, y, ...)
```

Arguments

`x`	data set to be used for plotting or vector of x values
`y`	vector of y values in case `x` is not a matrix
`...`	additional (graphics) parameter settings

Details

`slider.lowess.plot` computes a scatterplot of the data. Then a LOWESS smoother line is added to the plot. For more details about the lowess parameters `f` and `iter` take a look at the help page of `lowess`. The parameters are set by moving sliders of the control widget. The first slider defines the smoother span `f` and the second one the number of iterations.

Value

a message about the usage

Author(s)

Hans Peter Wolf

References

for references see help file of `lowess`

See Also

`lowess`, `slider`

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.lowess.plot(cars)

## End(Not run)
```

`slider.smooth.plot.ts` *interactive Tukey smoothing*

Description

`slider.smooth.plot.ts` computes smooth curves of a time series plot by Tukey's smoothers. The kind of smoothing is controlled by a Tcl/Tk widget.

Usage

```
slider.smooth.plot.ts(x, ...)
```

Arguments

`x`	time series
`...`	additional graphical parameters

Details

`slider.smooth.plot.ts` draws the time series `x`. The user selects a filter of the set `c("3RS3R", "3RSS", "3RSR", "3R", "` step by step and the resulting curve is added to the plot. The selection is performed by pressing a button of the control widget of `slider.smooth.plot.ts`. The button `reset` restarts the smoothing process.

Value

a message about the usage

Author(s)

Hans Peter Wolf

References

Tukey, J. W. (1977). Exploratory Data Analysis, Reading Massachusetts: Addison-Wesley.

See Also

plot, smooth

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.smooth.plot.ts(rnorm(100))

## End(Not run)
```

---

slider.split.plot.ts *interactive splitting of time series*

Description

slider.split.plot.ts plots linear fitted lines or summary statistics in sections of a time series. The sections are controlled by sliders.

Usage

```
slider.split.plot.ts(x, type="l", ...)
```

Arguments

x	time series or vector
type	plotting type: type will be forwarded to function plot
...	additional graphics parameters

Details

slider.split.plot.ts draws a time series plot and let you define sections of the series by fixing a limit on the time scale as well as a window width. The whole range of the series is partitioned in pieces of the same length in a way that the fixed limit will be one of the section limits. Then linear models are fitted and plotted in the sections. Alternatively – by pressing the button fivenum summary – summary statistics are drawn instead of the model lines.

The first slider fixes the width of the sections and the second one the limit between two of them.

By clicking on button linear model or fivenum summary the user switches between drawing model curves and five number summary.

Value

a message about the usage

Author(s)

Hans Peter Wolf

See Also

plot

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.split.plot.ts(as.vector(sunspots)[1:100])

## End(Not run)
```

---

slider.stem.leaf *construction of stem and leaf display interactively*

Description

'slider.stem.leaf' computes a stem and leaf display within a graphics device. The parameters are controlled by a control widget.

Usage

```
slider.stem.leaf(x, main = main)
```

Arguments

x	data set for plotting
main	main title of the plot

Details

The function 'slider.stem.leaf' allows the user to construct a stem and leaf display within a graphics device. The main parameters will be set by a Tcl/Tk control widget. The line rule is selected by pressing one of the buttons 'Dixon', 'Sturges', 'Velleman'. A slider controls the separation of the stem. Additionally the character size device could be set.

Value

a short message is returned

Note

The function is a function of the package aplpack

Author(s)

Peter Wolf, Nov 2009

See Also

stem

Examples

```
 ## Not run:
   slider.stem.leaf(islands)

## End(Not run)
```

---

slider.zoom.plot.ts *interactive zooming of time series*

Description

This function shows one or two sections of a time series. The window(s) is (are) controlled by sliders.

Usage

```
slider.zoom.plot.ts(x, n.windows, ...)
```

Arguments

x	time series
n.windows	if(n.windows>1 two sections are defined
...	additional graphical parameters

Details

slider.zoom.plot.ts plots the original time series and it lets you select one or two sections of the series by fixing the width(s) and the starting point(s) of the region(s). Then the section(s) of the series is (are) plotted separately one below the other.

The first slider defines the width of the section(s). The second (third) one sets the start of the first (second) section.

Value

a message about the usage

Author(s)

Hans Peter Wolf

See Also

plot

Examples

```
## Not run:
## This example cannot be run by examples() but should be work in an interactive R session
  slider.zoom.plot.ts(co2,2)

## End(Not run)
```

---

spin3R *spin3R*

Description

Simple spin function to rotate and to inspect a 3-dimensional cloud of points

Usage

```
spin3R(x, alpha = 1, delay = 0.015, na.rm=FALSE)
```

Arguments

x	(nx3)-matrix of points
alpha	angle between successive projections
delay	delay in seconds between two plots
na.rm	if TRUE 'NA' values are removed otherwise exchanged by mean

Details

spin3R computes two-dimensional projections of (nx3)-matrix x and plots them on the graphics devise. The cloud of points is rotated step by step. The rotation is defined by a tcl/tk control widget. spin3R requires tcl/tk package of R.

Note

version 01/2003

Author(s)

Peter Wolf

References

Cleveland, W. S. / McGill, M. E. (1988): Dynamic Graphics for Statistics. Wadsworth & Brooks/Cole, Belmont, California.

See Also

`spin` of S-Plus

Examples

```
xyz<-matrix(rnorm(300),100,3)
# now start:     spin3R(xyz)
```

---

stem.leaf *stem and leaf display and back to back stem and leaf display*

Description

Creates a classical ("Tukey-style") stem and leaf display / back-to-back stem and leaf display.

Usage

```
stem.leaf(data, unit, m, Min, Max, rule.line = c("Dixon", "Velleman", "Sturges"),
    style = c("Tukey", "bare"), trim.outliers = TRUE, depths = TRUE,
    reverse.negative.leaves = TRUE, na.rm = FALSE, printresult = TRUE)
stem.leaf.backback(x,y, unit, m, Min, Max, rule.line = c("Dixon", "Velleman",
    "Sturges"), style = c("Tukey", "bare"), trim.outliers = TRUE,
    depths = TRUE, reverse.negative.leaves = TRUE, na.rm = FALSE,
    printresult=TRUE, show.no.depths = FALSE, add.more.blanks = 0,
    back.to.back = TRUE)
```

Arguments

`data`	a numeric vector of data
`x`	first dataset for `stem.leaf.backback`
`y`	first dataset for `stem.leaf.backback`
`unit`	leaf unit, as a power of 10 (e.g., `100`, `.01`); if `unit` is missing `unit` is choosen by `stem.leaf`.
`m`	number of parts (1, 2, or 5) into which each stem will be separated; if `m` is missing the number of parts/stem (`m`) is choosen by `stem.leaf`.
`Min`	smallest non-outlying value; omit for automatic choice.
`Max`	largest non-outlying value; omit for automatic choice.
`rule.line`	the rule to use for choosing the desired number of lines in the display; `"Dixon"` = 10*log10(n); `"Velleman"` = 2*sqrt(n); `"Sturges"` = 1 + log2(n); the default is `"Dixon"`.

`style`	`"Tukey"` (the default) for "Tukey-style" divided stems; `"bare"` for divided stems that simply repeat the stem digits.
`trim.outliers`	if `TRUE` (the default), outliers are placed on `LO` and `HI` stems.
`depths`	if `TRUE` (the default), print a column of "depths" to the left of the stems; the depth of the stem containing the median is the stem-count enclosed in parentheses.
`reverse.negative.leaves`	if `TRUE` (the default), reverse direction the leaves on negative stems (so, e.g., the leaf 9 comes before the leaf 8, etc.).
`na.rm`	if TRUE "NA" values are removed otherwise the number of NAs are counted.
`printresult`	if TRUE output of the stem and leaf display by `cat`.
`show.no.depths`	if TRUE no depths are printed.
`add.more.blanks`	number of blanks that are added besides the leaves.
`back.to.back`	if FALSE two parallel stem and leaf displays are constructed.

Details

Unlike the `stem` function in the `base` package, `stem.leaf` produces classic stem-and-leaf displays, as described in Tukey's *Exploratory Data Analysis*. The function `stem.leaf.backback` creates back-to-back stem and leaf displays.

Value

The computed stem and leaf display is printed out. Invisibly `stem.leaf` returns the stem and leaf display as a list containing the elements `info` (legend), `display` (stem and leaf display as character vecter), `lower` (very small values), `upper` (very large values), `depths` (vector of depths), `stem` (stem information as a vector), and `leaves` (vector of leaves).

Author(s)

Peter Wolf, the code has been slightly modified by John Fox <jfox@mcmaster.ca> with the original author's permission, help page written by John Fox, the help page has been slightly modified by Peter Wolf.

References

Tukey, J. *Exploratory Data Analysis.* Addison-Wesley, 1977.

See Also

`stem`

Examples

```
stem.leaf(co2)
stem.leaf.backback(co2[1:120],co2[121:240])
stem.leaf.backback(co2[1:120],co2[121:240], back.to.back = FALSE)
stem.leaf.backback(co2[1:120],co2[121:240], back.to.back = FALSE,
```

```
                    add.more.blanks = 3, show.no.depths = TRUE)
stem.leaf.backback(rivers[-(1:30)],rivers[1:30], back.to.back = FALSE, unit=10, m=5,
                    Min=200, Max=900, add.more.blanks = 20, show.no.depths = TRUE)
```

Index