Package 'StratigrapheR'

June 19, 2019

Type Package

Title Integrated Stratigraphy

Version 0.0.6

Author Sebastien Wouters [aut, cre], Adam D. Smith [ctb]

Maintainer Sebastien Wouters <sebastien.wouters@doct.uliege.be>

Description Includes bases for litholog generation: graphical functions based on R base graphics, interval management functions and svg importation functions among others. Also include stereographic projection functions, and other functions made to deal with large datasets while keeping options to get into the details of the data.
When using for publication please cite Wouters, S., Da Silva, A.C. Crucifix, M., Sinnesael, M., Zivanovic, M., Boulvain, F., Devleeschouwer, X., 2019, Litholog generation with the StratigrapheR package and signal decomposition for cyclostratigraphic purposes. Geophysical Research Abstracts Vol. 21, EGU2019-5520, 2019, EGU General Assembly 2019. <http://hdl.handle.net/2268/234402>
The palaeomagnetism functions are based on:
Tauxe, L., 2010. Essentials of Paleomagnetism. University of California Press. <https://earthref.org/MagIC/books/Tauxe/Essentials/>;
Allmendinger, R. W., Cardozo, N. C., and Fisher, D., 2013, Structural Geology Algorithms: Vectors & Tensors: Cambridge, England, Cambridge University Press, 289 pp.;
Cardozo, N., and Allmendinger, R. W., 2013, Spherical projections with OSXStereonet: Computers & Geosciences, v. 51, no. 0, p. 193 - 205, <doi: 10.1016/j.cageo.2012.07.021>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.5.0)

Imports graphics, grDevices, XML, stats, utils, dplyr, stringr, ggplot2, GGally, shiny, diagram

Suggests astrochron, RFOC, plyr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-06-19 09:30:03 UTC

R topics documented:

`as.lim` *Creates / Checks / Manipulates lim objects*

Description

Creates and checks limits of intervals (what we define here as a 'lim' object), with control of specified properties. Basically we define an interval by its left and right boundaries, by an id and by a rule of boundary inclusion.

Usage

```
as.lim(lim = NULL, l = NULL, r = NULL, id = 1L, b = "[]")

is.lim(lim = NULL, l = NULL, r = NULL, id = 1L, b = "[]")

are.lim.nonunique(lim = NULL, l = NULL, r = NULL, check.lim = TRUE)

are.lim.nonadjacent(lim = NULL, l = NULL, r = NULL, b = "[]",
  check.lim = TRUE)

are.lim.distinct(lim = NULL, l = NULL, r = NULL, check.lim = TRUE)
```

```
are.lim.ordered(lim = NULL, l = NULL, r = NULL, id = 1L,
  decreasingly = FALSE, dependently = FALSE, check.lim = TRUE)

order.lim(lim = NULL, l = NULL, r = NULL, id = 1L, b = "[]",
  decreasingly = FALSE)
```

Arguments

`lim`	a list of n left (1st element) and n right (2ndt element) interval limits, of n interval IDs, and of n interval boundary rules (e.g. "[]")
`l`	the left interval limits (numerical vector of length n)
`r`	the right interval limits (numerical vector of length n)
`id`	the interval IDs (numerical or character vector of length n, the default is 1 for each interval). They can be similar for different intervals.
`b`	the interval boundaries rules: "[]" (or "closed") to include both boundaries points, "][" (or "()" and "open") to exclude both boundary points, "[[" (or "[)","right-open" and"left-closed") to include only the left boundary point, and "]]" (or "(]", "left-open", "right-closed") to include only the right boundary point. The notation is simplified to "[]", "[[", "]]" and "][" only.
`check.lim`	whether to check if the object is a lim object
`decreasingly`	whether the order to check for or set for is decreasing
`dependently`	whether the intervals themselves should be ordered relatively to the other

Details

`as.lim`: creates a lim object

`is.lim`: checks if arguments qualify as a lim object

`are.lim.nonunique`: checks if there are no intervals of identical l and r

`are.lim.nonadjacent`: checks if there are no pairs of intervals having at least one similar boundary

`are.lim.distinct`: checks if the intervals are not overlapping

`are.lim.ordered`: checks if the intervals are ordered (in l and r, and if dependently is TRUE, relative to the other intervals of same id)

`order.lim`: orders l and r parts of the intervals (use `simp.lim` for more advanced ordering)

See Also

To find which values are in which interval: `in.lim`

To simplify intervals by merging overlapping parts: `simp.lim`

To extract the part outside of intervals: `flip.lim`

To make intervals with boundaries in between given values: `mid.lim`

To discretise intervals: `tie.lim`

To simplify boundary rules into "[]", "[[", "]]" and "][": `rebound`

To plot interval data as rectangles: `infobar`

Examples

```
example <- as.lim(l = c(0,1,2), r = c(0.5,2.1,2.5), id = "I")

is.lim(lim = example)

are.lim.nonunique(l = c(0,1,2),r = c(0.5,1.5,2.5))

are.lim.nonunique(l = c(0,1,2),r = c(0.5,1.5,2))

are.lim.nonadjacent(l = c(0,1,2),r = c(0.5,1.5,2.5))

are.lim.nonadjacent(l = c(0,1,1.5),r = c(0.5,1.5,2))

are.lim.ordered(l = c(0,1,2),r = c(0.5,1.5,2.5))

are.lim.ordered(l = c(0,1,2.5),r = c(0.5,1.5,2))

are.lim.ordered(l = c(0,1,2),r = c(0.5,1.5,2.5),dependently = TRUE)

are.lim.ordered(l = c(0,2,1),r = c(0.5,2.5,1.5),dependently = TRUE)

are.lim.distinct(l = c(0,1,2),r = c(0.5,1.5,2.5))

are.lim.distinct(l = c(0,1,2),r = c(0.5,3.5,2.5))

order.lim(l = c(0,6,4,6,50), r = c(1,5,6,9,8),
          b = c("[[", "]]", "[[", "]]", "[["))
```

`bedtext` *Writes the names of the beds in a litholog*

Description

Writes the names of the beds in a litholog. You can either place them at the centre of the beds or in their upper and lower part. You can also define a thickness below which the name won't be written, to avoid excessive text crowding the plot.

Usage

```
bedtext(labels, l, r, x = 0.2, arg = list(cex = 1), adj = c(0.5,
  0.5), ymin = NA, edge = FALSE)
```

Arguments

`labels`	the name of each bed
`l`	a vector of n left y (or dt, i.e. depth or time) interval limits for each bed
`r`	a vector of n right y (or dt, i.e. depth or time) interval limits for each bed

x	the position where to write the text (0.2 by default)
arg	a list of arguments to feed text(). Go see ?text to know which arguments can be provided. See ?merge.list for further information.
adj	one or two values in [0, 1] which specify the x (and optionally y) adjustment of the labels. c(0.5,0.5) is the default.
ymin	minimum thickness of the bed to write its name (if NA, a default value is calculated, but user input is best)
edge	whether to put the bed name at the edge of the beds (T) or in the center of the beds (F, is the default)

See Also

litholog obvisously

if your boundaries have to be recalculated: leftlog

other functions complementing litholog: infobar and ylink

Examples

```
l  <- c(0,4,5,8)
r  <- c(4,5,8,16)

x   <- c(4,5,3,4)
i <- c("B1","B2","B3","B4")

test <- litholog(l, r, x, i)

whiteSet(xlim = c(0,6), ylim = c(-10,30))

multigons(test$i, test$xy, test$dt, col = c(NA, "black", "grey","NA"))

bedtext(labels = i, r = r, l = l, edge = TRUE, x = 0.5,
        arg = list(col = c("black", "white", "white", "red")))
```

---

blackSet *Sets the plot environment to draw a long vertical data set*

Description

Sets the plot environment to draw a long dataset. It provides lines as supplementary scale, and axes with major and minor ticks.

Usage

```
blackSet(xlim, ylim, xtick = NA, ytick = NA, nx = 1, ny = 1,
  xaxs = "i", yaxs = "i", xarg = list(tick.ratio = 0.5),
  yarg = list(tick.ratio = 0.5, las = 1), v = T, abbr = "",
  skip = 0, targ = list(col = "black", lwd = 2), sarg = list(lty = 2,
  col = "black"))
```

Arguments

xlim, ylim	the x and y limits (e.g. xlim = c(-1,1))
xtick, ytick	the interval between each major ticks for x and y
nx, ny	the number of intervals between major ticks to be divided by minor ticks in the x and y axes
xaxs, yaxs	The style of axis interval calculation to be used for the x and y axes. By default it is "i" (internal): it just finds an axis with pretty labels that fits within the original data range. You can also set it to "r" (regular): it first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range. See ?par for further explanation
xarg, yarg	a list of arguments to feed to minorAxis() for the x and y axes. See the ?minorAxis help page for the possible arguments. See ?merge_list for further information.
v	whether the lines are vertical
abbr	text to be repeated on the lines at each major tick
skip	number of text redundancies to be skipped
targ, sarg	a list of arguments to feed to text() and segments() respectively. If set to NULL, does not add the corresponding element.

Value

A plotting environment to draw a long data set

See Also

Similar functions: whiteSet and greySet

To create axes with major and minor ticks: minorAxis

To print a plot in pdf: pdfDisplay

To automatically determine pretty interval limits: encase

Examples

```
y <- c(0,11,19,33)
x <- c(1,2,2.5,4)

a <- min(y)
b <- max(y)

f<- encase(a-1,b,5)

blackSet(c(0,4),f, ytick = 10, ny = 10, skip = 1)

points(x, y, pch=19)
```

casing *Finds values in a vector directly above and below a number*

Description

Finds values in a vector directly above and below a number

Usage

```
casing(x, into)
```

Arguments

x	a number
into	a vector where to find the values directly above and below x

Value

a vector of the values of "into" vector directly above and below x respectively

See Also

Similar function : encase

Examples

```
casing(0.21,c(0.3,0.4,0.1,0.2))
```

centresvg *Draws a pointsvg object around a given point*

Description

Draws a svg object imported as data frame using pointsvg around a given point.

Usage

```
centresvg(object, x, y, xfac = 1, yfac = 1, xadj = 0, yadj = 0,
  forget = NULL, front = NULL, back = NULL, standard = FALSE,
  keep.ratio = FALSE, col = NA, border = "black", density = NA,
  angle = 45, lty = par("lty"), lwd = par("lwd"), scol = border,
  slty = lty, slwd = lwd, plot = TRUE, output = FALSE)

centersvg(object, x, y, xfac = 1, yfac = 1, xadj = 0, yadj = 0,
  forget = NULL, front = NULL, back = NULL, standard = FALSE,
```

```
keep.ratio = FALSE, col = NA, border = "black", density = NA,
angle = 45, lty = par("lty"), lwd = par("lwd"), scol = border,
slty = lty, slwd = lwd, plot = TRUE, output = FALSE)
```

Arguments

object	a pointsvg object (svg object imported as data frame using pointsvg).
x, y	numeric vectors of coordinates where the object should be drawn.
xfac	the x size factor.
yfac	the y size factor.
xadj	value specifying the x adjustment of the drawing.
yadj	value specifying the y adjustment of the drawing.
forget	the elements that should be discarded, by their id or index (i.e. name or number of appearance).
front, back	the elements to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one of the original .svg file.
standard	whether to standardise (centre to (0,0), rescale so that extreme points are at -1 and 1) or not (T or F)
keep.ratio	if the object is to be standardised, whether to keep the x/y ratio (T or F)
col	the polygones background color. If density is specified with a positive value this gives the color of the shading lines.
border	the lines color.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise)
lty, lwd	the border line type and width, see ?par for details.
scol, slty, slwd	the colour, type and width of the shading lines.
plot	whether to add to a plot
output	whether to output the new object coordinates

Details

The centresvg and framesvg have a lot of similarities with the multigons function: the graphical parameters are mostly identical. However there is a strong distinction between the -svg functions and multigons: when providing several graphical arguments, multigons will attribute them to each polygon, whereas the .svg functions will use them for each repetition of the .svg object. Using the latter, the graphical parameters will be applied to all the elements of a drawing. If you want a finer personalisation you have to use multigons and multilines (or an hybrid of the two, yet to be coded).

See Also

Similar functions: framesvg and placesvg

Change the drawing: changesvg and clipsvg

Uses ignore to avoid drawing unnecessary objects

Examples

```
object <- example.ammonite

plot(c(-10,10), c(-10,10), type = "n")

centresvg(object, 5, 5, xfac = 2, yfac = 2,lty = 1,density = 20, angle = 45)

points(5,5,pch = 19, col = "blue")
```

---

changesvg *Changes a pointsvg object*

Description

Changes a svg object imported as data frame using pointsvg.

Usage

```
changesvg(object, forget = NULL, front = NULL, back = NULL,
  standard = FALSE, keep.ratio = F, round = FALSE, xdigits = 4,
  ydigits = 4, xinverse = FALSE, yinverse = FALSE)
```

Arguments

object	a pointsvg object (svg object imported as data frame using pointsvg).
forget	the elements that should be discarded, by their id or index (i.e. name or number of appearance).
front, back	the elements to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one of the original .svg file.
standard	whether to standardise (centre to (0,0), rescale so that extreme points are at -1 and 1) or not (T or F)
keep.ratio	if the object is to be standardised, whether to keep the x/y ratio (T or F)
round	whether to round the coordinates or not (T or F)
xdigits	the number of digits after the decimal to round to for x values
ydigits	the number of digits after the decimal to round to for y values
xinverse	whether to inverse the plotting for x values (T or F)
yinverse	whether to inverse the plotting for y values (T or F)

Value

A data.frame with x and y coordinates, ids for each object, and a type, either line (L) or polygon (P)

See Also

Importing .svg objects: pointsvg

Plot the drawing and change the coordinates :placesvg, centresvg and framesvg

Clip the drawing: clipsvg

Examples

```
object1 <- example.lense

opar <- par("mfrow")
par(mfrow = c(1,3))

plot(c(-1,1), c(-1,1), type = "n")
placesvg(object1)

plot(c(-1,1), c(-1,1), type = "n")
object2 <- changesvg(object1, forget = 1)
placesvg(object2)

plot(c(-1,1), c(-1,1), type = "n")
object3 <- changesvg(object1, forget = "P1", standard = TRUE)
placesvg(object3)

par(mfrow = opar)
```

---

clipsvg *Clips a standardised pointsvg object into a given frame*

Description

Clips a svg object imported as data frame using pointsvg if outside of a given frame. In other words it removes the elements of the svg that are entirely outside a given area.

Usage

```
clipsvg(object, xmin = -Inf, xmax = +Inf, ymin = -Inf, ymax = +Inf,
  by.entity = TRUE)
```

Arguments

object	a pointsvg object (svg object imported as data frame using pointsvg).
xmin, xmax, ymin, ymax	clipping coordinates, default to +-Inf (no clipping)
by.entity	whether to remove all entities having points out of the clipping zone (TRUE; default) or to only remove the points out it (FALSE, and to use on lines for better result)

See Also

centresvg, changesvg, framesvg and pointsvg

If you want to also keep the elements that are only partly inside the clipping region: ignore

Examples

```
# Simple use

object <- example.ammonite

plot(c(-1,1), c(-1,1), type = "n", ylab = "y", xlab = "x")

res.object <- clipsvg(object, xmax = 0.5)

abline(v = 0.5)

centresvg(object, 0, 0, lty = 2)
centresvg(res.object, 0, 0, col = "red", lwd = 2)

# Advanced used

object2 <- example.breccia

plot(c(-1,3), c(-1,11), type = "n", ylab = "y", xlab = "x")

object2replicated <- framesvg(object2, 0,2,c(0,4,8), c(2,6,10),
                              output = TRUE)

object2clipped    <- clipsvg(object2replicated, 0, 1.7, 1, 9)

rect(0, 1, 1.7, 9, border = "red")

placesvg(object2clipped, border = "red", lwd = 2)
```

---

convert *Converts x values having an index into n values defined by the same y index*

Description

Converts x values having an index (of y values for instance) into n values defined by the same index (but having possibly more values)

Usage

```
convert(x, xindex, n, nindex)
```

Arguments

x	a vector
xindex	the index for each x value (vector of same length than x)
n	a vector of the values into which to convert the x values
nindex	the index for each n value (vector of same length than n)

Examples

```
x      <- c(10,20)
xindex <- c(1,2)

n      <- seq(0.1,1,by = 0.1)
nindex <- 1:length(n)

convert(x,xindex,n,nindex)
```

---

convertAxis *Converts the axis following a given formula*

Description

Converts the axis following a given formula, and places ticks in the new axis value

Usage

```
convertAxis(side, formula, at.maj, at.min = NULL, labels = at.maj,
  tick.ratio = 0.75, line = NA, pos = NA, font = NA,
  lty = "solid", lwd = 1, lwd.ticks = lwd, col = NULL,
  col.ticks = NULL, hadj = NA, padj = NA, tcl = NA, ...)
```

Arguments

side	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
formula	the formula to be converted. Should be of the form y ~ f(x)
at.maj	a vector of the position and labels of the major ticks

at.min	a vector of the position of minor ticks
labels	his can either be a logical value specifying whether (numerical) annotations are to be made at the major tickmarks, or a character or expression vector of labels to be placed at the major tickpoints.
tick.ratio	the ratio of minor to major tick size
line, pos, font, lty, lwd, lwd.ticks, col, col.ticks, hadj, padj, tcl, ...	see ?axis function help page for these parameters

See Also

minorAxis

Examples

```
plot(1,1,type = "n", xlim = c(0,12), axes = FALSE ,xlab = "", ylab = "")

axis(3)

l <- seq_log(10^0,10^12,divide = TRUE)

convertAxis(1,y ~ log10(x),l[[1]],l[[2]])
```

---

dipfix *Fix Dip*

Description

Fix dip and strike of planes so that they fall in the correct quadrant. The provided quadrant is the determining factor. If unavailable or not helpful, the sign of the dip is used as determining factor.

Usage

```
dipfix(strike, dip, quadrant = NA, inverted = NA)
```

Arguments

strike	strike of the data; it is the angle from the north of the horizontal line of the plane. Corrected, its range goes from 0° to 360°.
dip	dip of the data; it is the angle from the horizontal taken on the line of the plane perpendicular to the one of the strike. In other words it is the plane's maximum angular deviation from the horizontal. It is positive downward, and ranges from +90° for straight down to -90° for straight up. Dip values in [-180,-90] or/and]90,180] indicate inversion of the plane.
quadrant	the quadrant where the plane dips downward. Accepted values are NA, 'N', 'S', 'W' or 'E' (lower- or uppercase alike). Is independant of inversion
inverted	whether the plane is upside down.

Details

the strike will be corrected as the orientation of the dip (i.e. downward) minus 90°; it ranges from 0 to 360°. It is determined firstly from the quadrant. If the quadrant is missing or not helpful (e.g. 'N' or 'S' for a strike of 0° or 180°, 'E' or 'W' for a strike of 90° or 270°), it is determined using the sign of the dip. Inversion will be indicated if the dip values are in [-180,-90] or/and]90,180], or simply if inverted = T. The inversion does not influence the calculation of the strike, dip and quadrant: whether the plane is upside down does not change these parameters output.

Value

a list of the corrected strike, dip and quadrant

See Also

fmod, incfix and transphere

Examples

```
strike   <- c(-60, 180,20,0,20)
dip      <- c(-60,20,-45,110,-90)
quadrant <- c("N",NA,NA,NA,"E")
inverted <- c(FALSE,TRUE,FALSE,TRUE,FALSE)

dipfix(strike,dip,quadrant,inverted)

dipfix(strike,dip,quadrant)
```

---

earinc *Recalculates inclination in equal area projection*

Description

Recalculates inclination in equal area projection

Usage

```
earinc(inc)
```

Arguments

inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010).

Examples

```
earinc(20)
```

earnet *Draws an equal area stereonet*

Description

Draws Equal Area Stereo-Net. Lambert azimuthal Equal-Area (Schmidt) from Snyder p. 185-186 (modified from RFOC package)

Usage

```
earnet(xlim = c(-1.1, 1.1), ylim = c(-1.1, 1.1), ndiv = 10,
  col = gray(0.7), border = "black", lwd = 1, orientation = TRUE,
  xh = "WE", add = FALSE)
```

Arguments

xlim, ylim	the x and y minimal limits. The actual limits can change to keep a x/y ratio of 1
ndiv	the number of intervals between each line crossing
col	the colour of the net
border	the colour of the border and crosshair
lwd	the line width
orientation	logical, whether to add captions indicating the orientation of the plot.
xh	orientation of the x axis: can be 'WE' or 'SN'. Has to be provided to earplanes and earpoints
add	logical, whether to add the circle to an existing plot
...	graphical parameters to feed to lines

References

Snyder, John P., 1987, Map Projections-a working manual, USGS-Professional Paper, 383p. pages 185-186, RFOC package

See Also

earinc, earplanes, earpoints and zijderveld

Examples

```
par(mfrow = c(1,2))
earnet()
earnet(xh  = "SN")
par(mfrow = c(1,1))
```

earplanes *Draws planes on an equal area stereonet*

Description

Draws planes on an equal area stereonet (modified from RFOC package)

Usage

```
earplanes(strike, dip, quadrant = NA, hsphere = "l", ndiv = 10,
  a = list(col = "black", lwd = 1), l = list(lty = 1), u = list(lty =
  3), output = FALSE, plot = TRUE, xh = "WE", unique = TRUE)
```

Arguments

strike	strike of the data; it is the angle from the north of the horizontal line of the plane. It is corrected by the dipfix function.
dip	dip of the data; it is the angle from the horizontal taken on the line of the plane perpendicular to the one of the strike. It is corrected by the dipfix function.
quadrant	the quadrant were the plane dips downward. Accepted values are NA, 'N', 'S', 'W' or 'E' (lower- or uppercase alike) for correction by the dipfix function.
hsphere	the hemisphere onto which to project the data. Either "b" for both, "l" for lower, and u" for upper.
ndiv	the number of intervals between each 10° (in declination)
a, l, u	list of graphical parameters to feed lines() for the all lines, or for the lines of the upper (u) and lower (l) hemisphere (the two latter override a). See ?lines help page for the possible arguments. See ?merge_list for further information.
output	whether to return an output (position of the points making the lines in the stereographic projection)
plot	whether to plot
xh	orientation of the x axis: can be 'WE' or 'SN'.
unique	whether to only plot each similar plan once.

Value

the x,y coordinates of each projected plane

References

RFOC package

See Also

earnet, earpoints and dipfix

Examples

```
strike   <- c(45,  0)
dip      <- c(20, 65)

earnet()
earplanes(strike,dip,hsphere = "b")

encircle(earinc(dip))
```

---

earpoints *Draws points on an equal area stereonet*

Description

Draws points on an equal area stereonet (modified from RFOC package)

Usage

```
earpoints(dec, inc, hsphere = "b", double = FALSE, a = list(pch = 21,
  col = "black"), l = list(bg = "black"), h = list(bg = "grey"),
  u = list(bg = "white"), labels = NA, pos = 4, output = FALSE,
  plot = TRUE, xh = "WE")
```

Arguments

dec	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010). Values outside this range are corrected by the incfix function.
inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010). Values outside this range are corrected by the incfix function.
hsphere	the hemisphere onto which to project the data. The default is "b" for both: this useful in the case of oriented vectors rather than lines like for paleomagnetism. Other choices are "l" and "u" for lower and upper hemisphere.
double	whether to plot the equivalent point to one having an inclination of 0°(with dec = dec +180)
a, l, h, u	list of graphical parameters to feed points() for all points, or for the points of the upper (u) and lower (l) hemisphere, and of the samples having an inclination of 0° (h) (the three latter override a). See ?points help page for the possible arguments. See the example for illustration, and ?merge_list for further information.
labels	labels to each point
pos	position of each label (see text() help page)
output	whether to return an output (position of the points in the stereographic projection)
plot	whether to plot
xh	orientation of the x axis: can be 'WE' or 'SN'.

Value

the x,y coordinates of each point in the projection

References

Snyder, John P., 1987, Map Projections-a working manual, USGS-Professional Paper, 383p. pages 185-186, RFOC package

See Also

earnet, earplanes and incfix

Examples

```
earnet()

h <- 17
m <- 11

if(m < 10) a <- "0" else a <- ""

title(paste("Il est ", h, "h",a,m, sep = ""))

i1 <- seq(40, 100, by = 10)
i2 <- seq(0, -100, by = -10)
d1 <- rep(h * 30 + m * 0.5, length(i1))
d2 <- rep(m*6, length(i2))

inc <- c(i1,i2)
dec <- c(d1,d2)

earpoints(dec,inc)
```

---

encase *Encases two numbers between multiples of a given number*

Description

Encases two numbers between multiples of a given number

Usage

```
encase(x1, x2, n)
```

Arguments

x1	the first value of the interval
x2	the second value of the interval (can be higher or lower, but never equal to x1)
n	the number to find the multiples from

Value

the multiples of n directly encompassing x1 and x2

See Also

Similar function : `casing`

Examples

```
encase(5,1,5)
```

`encircle` *Draws circles*

Description

Draws circles

Usage

```
encircle(r = 1, x = 0, y = 0, ndiv = 360, plot = TRUE,
  add = TRUE, output = FALSE, ...)
```

Arguments

`r`	the radius of the circles (of length 1 or n)
`x`	the x value of the centre of the circles (of length 1 or n)
`y`	the y value of the centre of the circles (of length 1 or n)
`ndiv`	the number of segments making the circles
`plot`	whether to plot the circles
`add`	whether to add to an existing plot
`output`	whether to return an output
`...`	graphical parameters to feed to lines

Value

a list of x and y matrices having n rows, one for each circle

Examples

```
plot(0, 0, xlim = c(-1,1), ylim = c(-1,1), asp = 1)

encircle(lwd = 2)
encircle(r = seq(0.1,0.9,0.1))
```

enlarge *Expands the TRUE values of a T/F vector to their nth neighbours*

Description

Expands the TRUE values of a T/F vector to their nth neighbours

Usage

```
enlarge(x, n)
```

Arguments

x	a TRUE/FALSE vector (e.g. c(T,T,F,F,T,T))
n	the proximity order of the FALSE values neighbouring the TRUE values to be converted into TRUE (can be negative, should be convertible into an integer). For instance 1 means that the F values dirctly next to a T will be converted into T. 2 will apply that to the neighbours neigbhours, etc...

Value

a vector of T/F values, with the TRUE values expanded to their nth neighbours

Examples

```
# Creating a test dataset ----

y <- c(rep(c(0,1,0,-1),8),rep(-1,3),-1.5,
           rep(-1,2),rep(c(0,1,0,-1),8))
x <- 1:length(y)

df <- data.frame(x,y)

xclip <- c(20,48.5)
yclip <- c(-0.5,1.5)

conditions <- df$y > yclip[1] & df$y < yclip[2] &
              df$x > xclip[1] & df$x < xclip[2]

normt <- df[conditions,]

# Plotting supporting data ----

plot(df$x, df$y, type = "l", lty = 2, ylim = c(-2,2))

rect(xclip[1], yclip[1], xclip[2], yclip[2])

# See how the function reacts ----
```

```
embiggened <- enlarge(conditions,1)

test  <- df[embiggened,]

lines(test$x,test$y, lwd = 2, col = "blue")

points(normt$x,normt$y, type = "o", pch = 19,
       lty = 2, lwd= 2, col = "red")


legend(10, -1.6,
       legend = c(paste("Points initally isolated: they were chosen",
                        "to be the ones inside the rectangle"),
                  paste("Extension of the points: the first neighbours",
                        "of the points were added")),
       col = c("red", "blue"), pch = 19, lty = c(2,1), lwd = 2)
```

`every_nth` *Suppresses every n th element of a vector*

Description

Suppresses every n th element of a vector

Usage

```
every_nth(x, nth, empty = TRUE, inverse = FALSE)
```

Arguments

`x`	a vector (numbers, integers, characters, you name it)
`nth`	the multiple of position where the elements will be suppressed (nth + 1 actually) or kept (if inverse = T)
`empty`	whether the suppressed element should be replaced by ""
`inverse`	opposite reaction: n th elements only will be kept

Value

a vector with the remaining values

Author(s)

Adam D. Smith

See Also

practical usage of this function for axes: `minorAxis`

Examples

```
numvec <- 0:20

every_nth(numvec, 3)

every_nth(numvec, 3, empty = FALSE)

every_nth(numvec, 3, inverse = TRUE)

every_nth(numvec, 3, empty = FALSE, inverse = TRUE)
```

`fastPairs` *Scatterplot Matrices for large dataset*

Description

Makes a matrix of plots with a given data set of large size using hexagonal binning (similar result than the pairs function)

Usage

```
fastPairs(dat, bins = 60, base_size = 12, cor_size = 6)
```

Arguments

`dat`	data frame of all parameters to be compared
`bins`	numeric vector giving number of bins in both vertical and horizontal directions. Set to 60 by default.
`base_size`	size of the base texts (axis, titles, ...)
`cor_size`	size of the text for the correlation coefficient. Has different units (?) than base_size

Examples

```
fastPairs(iris[1:4],bins=10)
```

`flip.lim` *Inverts the intervals*

Description

Gives a negative of the intervals of a lim object

Usage

```
flip.lim(lim = NULL, l = NULL, r = NULL, b = "[]", xlim = NA)
```

Arguments

`lim`	an object convertible into a lim object: either a vector of length 2 or a list of n left (1st element) and n right (2ndt element) interval limits
`l`	a vector of n left interval limits
`r`	a vector of n right interval limits
`b`	a character vector for the interval boundaries rules: "[]" (or "closed") to include both boundaries points, "][" (or "()" and "open") to exclude both boundary points, "[[" (or "[)","right-open" and"left-closed") to include only the left boundary point, and "]]" (or "(]", "left-open", "right-closed") to include only the right boundary point. The notation is simplified to "[]", "[[", "]]" and "][" only.
`xlim`	the minimum and maximum of the new lim object (minimum and maximum of the old one if NA; is the default)

Value

a lim object of intervals in between the provided intervals

See Also

`as.lim`

Examples

```
l   <- c(1,3,5,7,9,10)
r   <- c(3,4,7,8,9,11)
b   <- "]["

xlim <- c(-1,15)

res <- flip.lim(l = l, r = r, b = b, xlim = xlim)

plot(1,1,type = "n", xlim = c(-4, 20), ylim = c(0.3, 1.8))
rect(l, 1.1, r, 1.4, col = "green", border = "darkgreen", lwd = 3)
rect(res$l, 1, res$r, 0.7, col = "red", border = "darkred", lwd = 3)
abline(v = xlim)
```

fmean *Fischer mean*

Description

Fischer mean

Usage

```
fmean(dec = NA, inc = NA, int = 1, x = NA, y = NA, z = NA,
  id = NA, cart = F)
```

Arguments

dec	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010). Values outside this range are corrected by incfix().
inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010). Values outside this range are corrected by incfix().
int	intensity of the data. Defaults to one (unit sphere).
x, y, z	cartesian coordinates. x is the North, y the East, and z straight down. If dec and inc are not provided they are used to be converted back in dec, inc and int data. Output is corrected by incfix().
id	a name for each point, identifying each group of points you would like to treat separately
cart	whether to output as cartesian coordinates, defaults to F

Value

a list of coordinates for the fischer mean, in cartesian form or dec, inc, int form

See Also

fmod, dipfix and incfix

Examples

```
dec <- c(rnorm(10, mean = 45, sd = 5), rnorm(10, mean = 20, sd = 5))
inc <- c(rnorm(10, mean = 45, sd = 5), rnorm(10, mean = 20, sd = 5))
id  <- c(rep(1, 10), rep(2, 10))

earnet()
earpoints(dec, inc)

fm <- fmean(dec, inc, id = id)
```

```
earpoints(fm $dec, fm$inc, l = list(bg = "red"))
```

`fmod` *Universal remainder function*

Description

Given a [xmin,xmax[or]xmin,xmax] interval, this function determines the remainder of each numeric relative to this interval. In other words if the interval was repeated over the whole numeric domain, this function determines where each value would be positioned in a given repetition.

Usage

```
fmod(x, xmax, xmin = 0, bounds = "[[")
```

Arguments

`x`	vector of floating point numbers
`xmax, xmin`	the limits of the interval
`bounds`	how to deal with boundaries (right- or left-open; '[[' or ']]')

See Also

`incfix`, `dipfix` and `transphere`

Examples

```
fmod(c(1260.23,360),360)

fmod(c(1260.23,360),360,bounds = "]]")

fmod(c(1260.23,360),360 + 180, 180)
```

`folder` *Creates a new folder where wanted if it does not exist yet*

Description

Creates a new folder where wanted if it does not exist yet

Usage

```
folder(dir, name)
```

Arguments

dir	directory containing the folder
name	name of the folder

Value

the directory of the folder itself

Examples

```
# # To run example uncomment all: put in a script, select all and use
# #  ctrl+shift+c
#
# folder(getwd(),"test")
```

---

formFunction *Converts a formula into a function*

Description

Converts a formula into a function

Usage

```
formFunction(formula)
```

Arguments

formula	the formula to be converted. Should be of the form y ~ f(x)

Examples

```
f <- formFunction(y ~ log10(x))

f(x=1:10)
```

framesvg *Draws a standardised pointsvg object into a given frame*

Description

Draws a svg object imported as data frame using pointsvg into a given frame.

Usage

```
framesvg(object, xmin, xmax, ymin, ymax, forget = NULL, front = NULL,
  back = NULL, standard = FALSE, keep.ratio = FALSE, col = NA,
  border = "black", density = NA, angle = 45, lwd = par("lwd"),
  lty = par("lty"), scol = border, slty = lty, slwd = lwd,
  plot = TRUE, output = FALSE)
```

Arguments

object	a pointsvg object (svg object imported as data frame using pointsvg).
xmin, xmax	the x value for the left and right side of the symbol
ymin, ymax	the y value for the low and high side of the symbol7
forget	the elements that should be discarded, by their id or index (i.e. name or number of appearance).
front, back	the elements to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one of the original .svg file.
standard	whether to standardise (centre to (0,0), rescale so that extreme points are at -1 and 1) or not (T or F)
keep.ratio	if the object is to be standardised, whether to keep the x/y ratio (T or F)
col	the polygones background color. If density is specified with a positive value this gives the color of the shading lines.
border	the lines color.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise)
lty, lwd	the border line type and width, see ?par for details.
scol, slty, slwd	the colour, type and width of the shading lines.
plot	whether to add to a plot
output	whether to output the new object coordinates

Details

The centresvg and framesvg have a lot of similarities with the multigons function: the graphical parameters are mostly identical. However there is a strong distinction between the -svg functions and multigons: when providing several graphical arguments, multigons will attribute them to each polygon, whereas the .svg functions will use them for each repetition of the .svg object. Using the latter, the graphical parameters will be applied to all the elements of a drawing. If you want a finer personalisation you have to use multigons and multilines (or an hybrid of the two, yet to be coded).

See Also

Similar functions: centresvg and placesvg

Change the drawing: changesvg and clipsvg

Uses ignore to avoid drawing unnecessary objects

Examples

```
# Simple use

object <- example.ammonite

xmin <- c(8,7)
xmax <- c(10,9)
ymin <- c(7,6)
ymax <- c(9,8)

plot(c(-10,10), c(-10,10), type = "n")

abline(v = unique(c(xmax, xmin)))
abline(h = unique(c(ymax, ymin)))

framesvg(object, xmin, xmax, ymin, ymax, col = c("white", "grey80"))

# Precision positioning

l <- c(1,2,3)
r <- c(0,1,2)
h  <- c(4,3,4)
i   <- c("B1","B2","B3")

basic.litholog <- litholog(l,r,h,i)

whiteSet(xlim = c(0,4), ylim = c(0,3), ytick = 1, ny = 10)

framesvg(example.lense, 0,3,1,2, forget = "P1", border = "red", lwd = 3)

multigons(basic.litholog$i, basic.litholog$xy, basic.litholog$dt)
```

greySet *Sets the plot environment to draw a long vertical data set*

Description

Sets the plot environment to draw a long dataset. It provides grey bands as supplementary scale, and axes with major and minor ticks.

Usage

```
greySet(xlim, ylim, xtick = NA, ytick = NA, nx = 1, ny = 1,
  xaxs = "i", yaxs = "i", xarg = list(tick.ratio = 0.5),
  yarg = list(tick.ratio = 0.5, las = 1), v = T, inverse = F,
  abbr = "", skip = 0, targ = list(col = "white", lwd = 2),
  rarg = list(border = NA, col = "grey85"))
```

Arguments

xlim, ylim	the x and y limits (e.g. xlim = c(-1,1))
xtick, ytick	the interval between each major ticks for x and y
nx, ny	the number of intervals between major ticks to be divided by minor ticks in the x and y axes
xaxs, yaxs	The style of axis interval calculation to be used for the x and y axes. By default it is "i" (internal): it just finds an axis with pretty labels that fits within the original data range. You can also set it to "r" (regular): it first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range. See ?par for further explanation
xarg, yarg	a list of arguments to feed to minorAxis() for the x and y axes. See the ?minorAxis help page for the possible arguments. See ?merge_list for further information.
v	whether the grey bands are vertical
inverse	inverse the bands position
abbr	text to be repeated in the grey bands each major tick
skip	number of text redundancies to be skipped
targ, rarg	a list of arguments to feed to text() and rect() respectively. If set to NULL, does not add the corresponding element.

Value

A plotting environment to draw a long data set

See Also

Similar functions: whiteSet and greySet

To create axes with major and minor ticks: minorAxis

To print a plot in pdf: pdfDisplay

To automatically determine pretty interval limits: encase

Examples

```
y <- c(0,11,19,33)
x <- c(1,2,2.5,4)

a <- min(y)
b <- max(y)

f<- encase(a-1,b,5)

greySet(c(0,4),f,abbr="abbr", ytick = 10, ny = 10)

points(x, y, pch=19)
```

---

ignore *Ignores useless objects*

Description

Ignores useless objects: this function will discard the polygons or polylines outside a certain range. This allows to avoid unnecessary work for multigons(), multilines(), centresvg() and framesvg().

Usage

```
ignore(i, x, y = NA, d = list(), j = unique(i), arg = list(),
  xlim = par("usr")[c(1, 2)], ylim = par("usr")[c(3, 4)],
  xlog = par("xlog"), ylog = par("ylog"))
```

Arguments

i	a polygon id for each x and y coordinate. If n objects are provided there should be n unique ids describing them, and the graphical parameters should be of length 1 or n.
x, y	numeric vectors of coordinates.
d	a list of named vectors going with i, x and y
j	a list of the ids in the order used for the arg arguments. By default they are in their order of appearance in i
arg	a list of arguments f length 1 or n.
xlim, ylim	the limits in x and y; if any object has all his points past one of these limits, it will be removed.
xlog, ylog	whether the axes have logarithmic scale

Value

a list of i, x, y, d, j and arguments.

See Also

Tributary functions: multigons, multilines, centresvg and framesvg

Examples

```
i <- c(rep("A1",6), rep("A2",6), rep("A3",6))
x <- c(1,2,3,3,2,1,4,5,6,6,5,4,7,8,9,9,8,7)
y <- c(1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6)

xlim <- c(2,5)
ylim <- c(0,1.5)

plot(c(0,10),c(0,10),type = "n")
rect(xlim[1], ylim[1], xlim[2], ylim[2])

multilines(i, x, y, lty = 3, col = "grey80")

res <- ignore(i, x, y, arg = list(lty =  1, lwd = 3,
              col = c("orange", "green", "red")),
              xlim = xlim, ylim = ylim)

do.call(multilines, res)
```

---

in.lim *Finds the intervals encompassing values*

Description

This function returns the intervals encompassing x values. This works only if the intervals (as lim objects) are non overlapping and non-adjacent (if certain boundaries are neighbouring, the boundary rule should exclude all, or all but one)

Usage

```
in.lim(x, lim = NULL, l = NULL, r = NULL, id = 1L, b = "][",
  index = FALSE)
```

Arguments

x	a vector values
lim	an object convertible into a lim object: either a vector of length 2 or a list of n left (1st element) and n right (2ndt element) interval limits. The intervals should be non-overlapping and non-adjacent.

l	a vector of n left interval limits
r	a vector of n right interval limits
id	a vector of n interval IDs (default is 1 for each interval)
b	a character vector for the interval boundaries rules: "[]" (or "closed") to include both boundaries points, "][" (or "()" and "open") to exclude both boundary points, "[[" (or "[)","right-open" and"left-closed") to include only the left boundary point, and "]]" (or "(]", "left-open", "right-closed") to include only the right boundary point.
index	whether the output should be a list of the initial vector and of the corresponding intervals in which they lay (index = FALSE, is the default), or simply the index of the intervals in the initial lim object (index = TRUE)

Value

a list of the intervals where the x values lay or a vector of their index

See Also

as.lim

Examples

```
x   <- c(99,1,3,5,2,4,5,6,9,4,8,20,26,52,42,24,25,12,40,10,16,17)

lim <- as.lim(l = c(100,10,20,27), r = c(99,12,27,42), b = "]]")

in.lim(x, lim = lim)

in.lim(x, lim = lim, index = TRUE)

# Applications to Stratigraphy

proxy <- proxy.example # This is a data.frame with (fake) magnetic
                       # susceptibility (ms) and depth (dt)

# Each sample was taken in a specific bed (not at the boundary between two,
# to make things easier). We will invoke the data of the beds (bed.example)
# and identify the lithology of each sample

res <- in.lim(proxy.example$dt,  # Position of each sample
              l = bed.example$l, # Left boundary of the beds
              r = bed.example$r, # Right boundary of the beds
              id = bed.example$litho) # Lithology of each bed (if you wanted
                                      # to know the name of the bed each
                                      # sample is in you would have put
                                      # bed.example$id)

proxy$litho <- res$id # The result provides the id (here the lithology) of
                      # each interval encompassing the measurement (x, here
                      # proxy.example$dt)
```

```
plot(proxy$ms, proxy$dt, type = "l", xlim = c(-2*10^-8, 8*10^-8))

shale <- subset(proxy, proxy$litho == "S")
points(shale$ms, shale$dt, pch = 4)

limestone <- subset(proxy, proxy$litho == "L")
points(limestone$ms, limestone$dt, pch = 19)

chert <- subset(proxy, proxy$litho == "C")
points(chert$ms, chert$dt, pch = 21, bg = "white")

legend(6.2*10^-8, 25, legend = c("Shale", "Limestone", "Chert"),
       pch = c(4,19,21), bg = c(NA, NA, "white"))
```

`incfix` *Fix Inclination*

Description

Fix inclination and declination so that they fall in the correct quadrant and hemisphere (modified from RFOC package)

Usage

```
incfix(dec, inc, hsphere = "b")
```

Arguments

`dec`	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010). Values outside this range are corrected by this function.
`inc`	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010). Values outside this range are corrected by this function.
`hsphere`	the hemisphere onto which to project the data. Either "b" for both. This is the default and useful for paleomagnetism. In this case positive and negative values of inc are permitted. Or "l" for lower, and "u" for upper, allowing only negative or positive inc values respectively.

Details

Quadrants are determined by the sine and cosine of the dip angle: co = cos(dip), si = sin(dip), quad[co>=0 & si>=0] = 1, quad[co<0 & si>=0] = 2, quad[co<0 & si<0] = 3 and quad[co>=0 & si<0] = 4. Samples at inc == 0° and inc == 90° are taken as exceptions (cf. code). Be cautions with the floating point error however, round if needed.

See Also

fmod, dipfix and transphere

Examples

```
incfix(591,-425,"b")
incfix(591,-425,"u")
incfix(591,-425,"l")
```

---

infobar *Draws rectangles with text in them*

Description

Draws rectangles with text in them, typically to delimit (stratigraphical) intervals (e.g. magnotochrons, but also lithostratigraphy,...)

Usage

```
infobar(xmin, xmax, ymax, ymin, labels = NA, m = list(), t = list(),
  srt = 90, family = par("family"), xpd = par("xpd"))
```

Arguments

xmin, xmax, ymin, ymax	x and y limits for the rectangles. You can either provide 1 or n of each (if you want to have always the same x limits but multiple and different y ones it is possible)
labels	a 1 or n character vector (i.e. text) specifying the text to be written in the rectangle. You can write "" for no text.
m, t	a list graphical parameters (of length 1 or n) to feed multigons() for m and to text() for t. See respective help pages ?multigons and ?text for the possible arguments. See the example for illustration, and ?merge_list for further information.
srt, family, xpd	further graphical parameters, see ?par for information

See Also

Similar functions: multigons, bedtext, nlegend and ylink

To deal with intervals: as.lim and related functions

Examples

```
labels <- c("High 5", "Low 5", "5")
ymin <- c(10,-10,2.5)
ymax <- c(20,0, 7.5)

plot(c(0,6),c(-20,20), type = "n")

infobar(xmin = 0, xmax = 1, ymin = ymin, ymax = ymax, labels,
        m = list(col = c("grey","grey", "red"),
                 border = "black", density = 10),
        t = list(cex = 1.5, col = "white"))
```

`leftlog` *Finds bed intervals in a "litholog()"-like data frame*

Description

Determines the interval of bed boundaries at the far left of a litholog. This is used when the welding of varying bed boundaries changes these intervals, and that you want to use bedtext() to print the name of the beds on the log.

Usage

```
leftlog(i, dt, xy, warn = TRUE)
```

Arguments

`i`	the id of the polygons in the "litholog()"-like data frame
`dt`	the depth of the polygons in the "litholog()"-like data frame
`xy`	the x values (i.e. hardness) of the polygons in the "litholog()"-like data frame
`warn`	whether you want to be annoyed

Value

a list of minima (l) and maxima (r) of boundaries corresponding to each bed (id)

See Also

`litholog`, `weldlog` and `bedtext`

Examples

```
l <- c(0,1,2,3,4)
r <- c(1,2,3,4,5)
h   <- c(4,3,4,3,4)
i <- c("B1","B2","B3","B4","B5")
log  <- litholog(l, r, h, i)

whiteSet(xlim = c(-1,5), ylim = c(-1,6))

title("leftlog() gets the bed names in the right position")

multigons(log$i, log$xy, log$dt, lty = 3)

seg1 <- sinpoint(4, 0, 0.25, pos = 1, phase=0)
seg2 <- sinpoint(4, 0, 0.25, pos = 1, phase=1)

welded <- weldlog(log, dt = c(2,3), seg = list(seg1, seg2), add.dt = 0.5)

multigons(welded$i, welded$xy, welded$dt, lwd = 3, lty = 2, border = "red")

old.log.interval <- leftlog(log$i, log$dt, log$xy)
new.log.interval <- leftlog(welded$i, welded$dt, welded$xy)

bedtext(labels = new.log.interval$id,
        l= new.log.interval$l,
        r= new.log.interval$r,
        arg = list(col = "red"))
```

---

litholog *Creates a litholog*

Description

Creates basic coordinates of polygons to draw a simple litholog with rectangles

Usage

```
litholog(l, r, h, i)
```

Arguments

l, r	the height of each delimitation (upper and lower; l and r stand for left and right boundaries of the interval, their order does not matter)
h	the hardness of each bed
i	the id of each bed

Value

A table of depth (dt) and xy value (i.e. hardness, or simply the x position if your litholog is vertical) of rectangles for each bed. Each bed is defined by an id (or name), which is the variable i in the table.

See Also

For a more detailed explanation of how to make a litholog: StratigrapheR

How to prepare the plot background for the litholog: whiteSet

How to draw the litholog: multigons

How to add the names of the beds in the litholog: bedtext

How to plot in pdf: pdfDisplay

To add personalised boundaries between beds: weldlog

To have open beds at the extremities of the log. More generaly to transform a polygon into a polyline and control the part that is not drawn: multilines and shift

To add details and drawings: centresvg and framesvg

Go further with interval data (between two boundaries, as there often is in stratigraphy): as.lim and related functions.

Complementary functions: infobar and ylink

Examples

```
l <- c(1,2,3)  # left boundary of the bed interval (upper or lower)
r <- c(0,1,2)  # right boundary of the bed interval (upper or lower)
h  <- c(4,3,4) # hardness (arbitrary)
i   <- c("B1","B2","B3") # Bed name

basic.litholog <- litholog(l,r,h,i) # Generate data frame of the polygons
                                    # making the litholog

whiteSet(xlim = c(0,4), ylim = c(0,3), ytick = 1, ny = 10) # Plot background
multigons(basic.litholog$i, basic.litholog$xy, basic.litholog$dt) # Draw log
```

---

mat.lag *Find the "next" or "previous" values in a matrix.*

Description

Find the "next" or "previous" values in a matrix.

Usage

```
mat.lag(m, n = 1L, default = NA)

mat.lead(m, n = 1L, default = NA)
```

Arguments

m	a matrix of values
n	a positive integer of length 1, giving the number of positions to lead or lag by
default	value used for non-existent rows. Defaults to NA.

Examples

```
m <- matrix(1:120, ncol = 12)

mat.lag(m)
mat.lead(m)
```

---

merge_list *Method for merging lists by name*

Description

This is a method that merges the contents of lists based on the name of the elements. In the case of identical names, the order of the lists determines which element is kept.

Usage

```
merge_list(l1, l2, ...)
```

Arguments

l1	the first list.
l2	the list which will supply additional elements to l1 that are not already there by name.
...	additionnal lists, that bring elements if they are not existing by name in the ones before.

Details

if a name appears more than once in a list, only the first one will be kept. This is particularly useful if you want to still be able to provide whichever argument you want to a function inside another function. See the advanced use in the examples to see how to do it.

Value

A merged list of all lists provided, each element (determined by its name) appearing only once.

See Also

To get a better understanding of this, go see ?do.call and ?list

Examples

```
# Simple use

a <- list(lty = c(2,4), mar = 4, plot = TRUE)
b <- list(mar = "hype",lty = "hype",  pink = TRUE)
d <- list(lty = FALSE, pink = "Yikes", mar = "ldkfj", test = "Successful")

merge_list(a,b,d)

# Advanced use

# We will plot points with different parameters for each lithology (see also
# the example in ?in.lim)

advanced.ex <- function(line.args = list(lty = 3, col = "grey"),
                        all = list(pch = 21, cex = 2),
                        chert = list(bg = "white"),
                        limestone = list(bg = "black"),
                        shale = list(bg = "red"),
                        main = "")
{

  # Preparation of plot and necessary data frames

  plot(proxy.example.litho$ms, proxy.example.litho$dt, type = "n",
       xlim = c(-2*10^-8, 8*10^-8), main = main)

  shale.df <- subset(proxy.example.litho, proxy.example.litho$litho == "S")
  limestone.df <- subset(proxy.example.litho, proxy.example.litho$litho == "L")
  chert.df <- subset(proxy.example.litho, proxy.example.litho$litho == "C")

  # Important part:

  # We use the do.call function, which calls a given function and provides
  # its arguments via a list. It is that list that is created by merge list.
  # for the lines function, we provide x and y coordinates, a personalised
  # list of arguments (line), and the default parameters. In this order the
  # personalised arguments override the default ones, but the latter are used
  # in the absence of personalised arguments

  line.args <- merge_list(list(x = proxy.example.litho$ms,
                               y = proxy.example.litho$dt),
                          line.args, # personalised list of arguments
                          list(lty = 3, col = "grey") # default parameters
  )

  do.call(lines, args = line.args)

  # Same procedure for the points of each lithology, but we add an 'all'
  # argument that applies for each point

  chert.args <- merge_list(list(x = chert.df$ms,
```

```
                              y = chert.df$dt), # Coordinates
                            chert, # Personalised arguments for cherts
                            all,   # Personalised arguments for all points
                            list(bg = "red"),       # Default arguments
                            list(pch = 21, cex = 2) # Default arguments
    )

    limestone.args <- merge_list(list(x = limestone.df$ms,
                                      y = limestone.df$dt),
                                 limestone, all,
                                 list(bg = "red"), list(pch = 21, cex = 2))

    shale.args <- merge_list(list(x = shale.df$ms, y = shale.df$dt),
                             shale, all,
                             list(bg = "red"), list(pch = 21, cex = 2))

    do.call(points, args = chert.args)
    do.call(points, args = limestone.args)
    do.call(points, args = shale.args)

  }

  omfrow <- par()$mfrow

  par(mfrow = c(1,3))

  advanced.ex(main = "Default")

  advanced.ex(main = "Change line and all",
              line.args = list(lty = 1),
              all = list(pch = 22))

  advanced.ex(main = "Personalise more",
             line.args = list(lty = 1, col = "black"),
              all = list(pch = 22),
              shale = list(pch = 4))

  par(mfrow = omfrow)
```

`mid.lim` *Provides mid-points intervals in an ordered vector*

Description

Provides mid-points intervals in an ordered vector

Usage

```
mid.lim(x, id = 1L, b = "[]")
```

Arguments

x	an ordered vector
id	a vector of n interval IDs (default is 1 for each interval)
b	a character vector for the interval boundaries rules, see as.lim help page for details

Value

a lim object of intervals with boundaries at midway between the x values

See Also

as.lim

Examples

```
mid.lim(c(1,3,7,20,45,63))
```

---

minorAxis *Adds an axis with minor ticks to a plot*

Description

Adds an axis with minor ticks to a plot, but with the possibility to have no superposition of minor ticks on major ticks, allowing to export a clean plot in vector format. It is based on the minor.tick function in the Hmisc package.

Usage

```
minorAxis(side, n = NULL, at.maj = NULL, at.min = NULL,
  range = NULL, tick.ratio = 0.5, labels.maj = TRUE, line = NA,
  pos = NA, outer = FALSE, font = NA, lty = "solid", lwd = 1,
  lwd.ticks = lwd, col = NULL, col.ticks = NULL, hadj = NA,
  padj = NA, extend = FALSE, tcl = NA, ...)
```

Arguments

side	an integer (here 1,2,3 or 4) specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and, 4=right.
n	the number of intervals defined by the minor ticks
at.maj	the positions at which major tick-marks are to be drawn. By default (when NULL) tickmark locations are computed, see the "Details" part in the ?axis help page.
at.min	the positions at which minor tick-marks are to be drawn. This parameter overrides n.

range	the range of the axis
tick.ratio	ratio of lengths of minor tick marks to major tick marks. The length of major tick marks is retrieved from par("tcl") unless specified otherwise.
labels.maj	this can either be a logical value specifying whether (numerical) annotations are to be made at the major tickmarks, or a character or expression vector of labels to be placed at the major tickpoints.
line, pos, outer, font, lty, lwd, lwd.ticks, col, col.ticks, hadj, padj, tcl, ...	see the ?axis function help page for the other parameters
extend	whether to add minor ticks even outside the major ticks (T) or not (F)

See Also

Set a plot environment with minorAxis: whiteSet, blackSet and greySet

This function is based on every_nth, which suppresses values every multiple of a given number.

Examples

```
plot(c(0,1), c(0,1), axes = FALSE, type = "n", xlab = "", ylab = "")

minorAxis(1, n = 10, range = c(0.12,0.61))

minorAxis(3, n = 10, extend=FALSE)
```

---

multigons *Draws several polygons*

Description

Draws several polygons. This function expands on the polygon() function from base R graphics. The difference is that several polygons can be drawn in one line by providing a polygon id: i. To each polygon you can provide different graphical parameters (i.e. colour, shading, etc). On the contrary of the polygon() function the graphical parameters of the shading lines can be independent of the border lines.

Usage

```
multigons(i, x, y, j = unique(i), forget = NULL, front = NULL,
  back = NULL, density = NA, angle = 45, border = "black",
  col = NA, lty = par("lty"), lwd = par("lwd"), scol = border,
  slty = lty, slwd = lwd, lend = 0, ljoin = 0, lmitre = 10)
```

Arguments

i	a polygon id for each x and y coordinate, i.e. the name of each polygon. If you want to give each polygon a different aspect you should provide a vector of n elements (if you have three polygons "A1", "A2" and "A3" with "A2" that should be blue you should provide the colours of all three: e.g. `col = c("white", "blue", "white")`)
x, y	numeric vectors of x and y coordinates
j	a list of the ids (names) in the order used for the graphical parameters (e.g. colour, shading, etc...). By default they are in their order of appearance in `i`
forget	the polygons that should not be drawn, by their id or index (i.e. name or number of appearance).
front, back	the polygons to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one defined by `j`, and if `j` is absent by the order in `i`.
density	the density of shading lines, in lines per inch. The default value of NA means that no shading lines are drawn. A zero value of density means no shading nor filling whereas negative values and NA suppress shading.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise).
border	the colour to draw the border. The default is black. Use border = NA to omit borders.
col	the colour for filling the polygon. The default, NA, is to leave polygons unfilled.
lty, lwd	the border line type and width, see ?par for details.
scol, slty, slwd	the colour, type and width of the shading lines.
lend, ljoin, lmitre	additional graphical parameters, see ?par for details.

Details

In the case you want shading this function will draw three overlapping polygons: one for the background, one for the shading lines and one for the border. `multigons` shares similarities with `centresvg` and `framesvg`, but allows more advanced control of each element.

See Also

Similar functions: `multilines`, `infobar`

Complementary function: `shift`

Uses `ignore` to avoid drawing unnecessary objects

Examples

```
# Simple use:

i <- c(rep("A1",6), rep("A2",6), rep("A3",6)) # Polygon ids
x <- c(1,2,3,3,2,1,2,3,4,4,3,2,3,4,5,5,4,3)   # x coordinates
y <- c(1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6)   # y coordinates
```

```
plot(c(-1,7), c(-2,9), type = "n", xlab = "", ylab = "", main = "Simple use")

multigons(i, x, y,
          front = "A2", # This gets the polygon A2 in front of all others
          density = c(NA, 5, 10),  # Different shading density
          scol = "darkred", # Same shading colour
          col = c("black", "grey", "white"), # Different background colour
          lwd = 2, # Width of border lines for all polygons
          slty = 2, # Shading lines type, same for all polygons
          slwd = 1) # Shading lines width, same for all polygons

# Advanced use:
# Lets first create more polygons

i2 <- c(i, rep("A4",6), rep("A5",6), rep("A6",6))
x2 <- rep(x,2)
y2 <- c(y, y - 4)

# Then lets attribute a group to each of them: lets say blue and red polygons

groups <- data.frame(j = c("A1", "A2", "A3", "A4", "A5","A6"),
                     group = c("blue", "red", "blue", "red", "red", "blue"),
                     stringsAsFactors = FALSE)

# Then lets attribute different graphical parameters for each group

legend <- data.frame(group = c("red", "blue"),
                     col = c("red", "blue"),
                     density = c(10,20),
                     scol = c("darkred", "darkblue"),
                      stringsAsFactors = FALSE)

 # Now that you have a data frame saying which polygon is in which group,
 # and one providing distinct graphical parameters for each group, you can
 # join the two with help of the dplyr package:

 library(dplyr)

 parameters <- left_join(groups, legend, by = "group")

 # Then simply apply them to multigons:

 plot(c(0,12), c(-3,7), type = "n", xlab = "", ylab = "",
      main = "Advanced use")

 multigons(i2,x2,y2,
           forget = c("A1"),     # If you want to avoid drawing one polygon
           front = c("A2","A3"), # Puts A2 in front and A3 right behind
           col = parameters$col,
           density = parameters$density,
           scol = parameters$scol,
           lwd = 2)
```

```
# Another way (more advanced, but with interesting programming applications)
# to code this:

all.parameters <- merge_list(list(i = i2, x = x2 + 6, y = y2),
                             as.list(parameters),
                             list(lwd = 3, slwd = 2, slty = 2))

all.parameters <- all.parameters[!names(all.parameters) == "group"]

do.call(multigons, all.parameters)
```

`multilines` *Draws several lines*

Description

Draws several polylines or group of points. This function expands on the lines() and points functions from base R graphics. The difference is that several lines and group of points can be drawn in one line by providing an id: i. To each line and group of point you can provide different graphical parameters (i.e. colour, type, etc).

Usage

```
multilines(i, x, y, j = unique(i), forget = NULL, front = NULL,
  back = NULL, type = "l", col = "black", bg = NA, pch = 19,
  lty = par("lty"), lwd = par("lwd"), cex = par("cex"), lend = 0,
  ljoin = 0, lmitre = 10)
```

Arguments

`i`	a line id for each x and y coordinate, i.e. the name of each polyline. If you want to give each line a different aspect you should provide a vector of n elements (if you have three lines "A1", "A2" and "A3" with "A2" that should be blue you should provide the colours of all three: e.g. `col = c("white", "blue", "white")`)
`x, y`	numeric vectors of x and y coordinates
`j`	a list of the ids (names) in the order used for the graphical parameters (e.g. colour, shading, etc...). By default they are in their order of appearance in `i`
`forget`	the lines that should not be drawn, by their id or index (i.e. name or number of appearance).
`front, back`	the lines to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one defined by `j`, and if `j` is absent by the order in `i`.
`type`	character indicating the type of plotting. For this function it is limited to "l" (lines, is the default), "p" (points) and "o" (points overplotting lines).

col	the color to draw the line. The default is black.
bg	background (fill) color for the open plot symbols given by pch = 21:25.
pch	plotting 'character', i.e., symbol to use. See ?points for further details
lty, lwd	the line type and width, see ?par for details.
cex	haracter (or symbol) expansion: a numerical vector. This works as a multiple of par("cex")
lend, ljoin, lmitre	additional graphical parameters, see ?par for details.

See Also

multigons

Complementary function: shift

Uses ignore to avoid drawing unnecessary objects

Examples

```
i <- c(rep("A1",6), rep("A2",6), rep("A3",6))
x <- c(1,2,3,3,2,1,4,5,6,6,5,4,7,8,9,9,8,7)
y <- c(1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6)

plot(c(0,10),c(0,7),type = "n")

multilines(i, x, y, j = c("A3", "A1", "A2"), lty =  c(1,2,3), lwd = 2,
           type = c("l", "o", "o"), pch = c(NA,21,24), cex = 2)
```

---

neatPick *Interactive user modification of the arguments of a repeated function*

Description

This opens a shiny app that will allow to manipulate the arguments of a function interactively, with different conditions that the user can provide a priori and modify at will

Usage

```
neatPick(fun, n, args = list(), class.args = list(), pick = NA,
  fix = NA, buttonswidth = 2, text = "output", textwidth = 4,
  plotwidth = 800, plotheight = 600, args.only = F, width = 10,
  height = 10, name = "fig", dir = getwd(), gfile = "onePDF",
  openfile = TRUE, folder = "Rfig", gfun = "jpeg", ext = ".jpeg",
  gargs = list(units = "in", res = 300), pargs = list(ps = 12, cex =
  1.5))
```

Arguments

`fun`	the function to be applied n times.
`n`	number of runs.
`args`	the arguments to be supplied to fun. Should be a list of each argument to be supplied to fun, having n elements stored indiscriminately in list or in vector form.
`class.args`	the class of the arguments, in a list. This is useful when the starting arguments are NA
`pick`	which arguments to be able to adapt interactively
`fix`	which arguments that cannot be chosen interactively (if pick is NA)
`buttonswidth`	the width of the buttons panel (integer from 1 to 12)
`text`	which information to send to the text panel. The default is the output of the current element (ni); "output". Can be the whole dataset of arguments; "all". Otherwise the panel does not show.
`textwidth`	the width of the text panel (integer from 1 to 12)
`plotwidth`	the width of the plot panel (arbitrary units)
`plotheight`	the height of the plot panel (arbitrary units)
`args.only`	whether to be only allowed to download and return the arguments (this simplifies things and makes the workflow more efficient)
`width, height, name, dir, gfile, openfile, folder, gfun, ext, gargs`	arguments to be supplied to neatPicked, the equivalent of neatPick without interactivity: it runs the function for each ni and saves the output (normal and graphical). In neatPick this happens when the button 'Run and Download Output' is clicked. See ?neatPicked function help page for details.
`pargs`	the arguments to transmit to par(), in neatPick and neatPicked

Details

This is a complicated function. A few basics:

neatPick works using the formals() function. It evaluates the arguments and their default values of any function that you provide without parentheses, like this for instance: `formals(multigons)`.

neatPick is capable of providing interaction with arguments of class integer or numeric (e.g. 10, or 13,58745), character (e.g. "BlipBlapBLoup") and logical (T or F), as long as for each iteration (n) the length of the argument is one (you cannot use arguments like xlim = c(0,1), however you can use xmin = 0 and xmax = 1 for instance). But you can provide a different value for each iteration n (if n = 3, you can provide col = c("red", "blue", "green") in the args list of arguments)

You can chose which arguments are interactive or not using the 'pick' and 'fix' arguments.

To return the arguments or the output, you have to click on 'End & Return Arguments' or 'End & Return Output', respectively.

You can also save the obtained output and arguments via the download buttons: you geta .RData file were the arguments are in the object saved.arguments and the output is in the saved.output object. The arguments can also be found at saved.output$args. The arguments can be provided to the args argument of the same neatPick function to rework the changes you made.

Examples

```
# # Use for stratigraphy: uncomment the entire example script to make it work
# # (remove the first # of each line; you can use the
# # shortcut Ctrl + Shift + c on the whole script)
# #
# # You create a simple function. The one below creates sinusoidal waves between
# # x0 = 0 and x1 = 1. You want to personalise the amplitude (delta), the y
# # offset (pos, see ?sinpoint for more details), the phase (phase, expressed
# # in multiples of pi), the number of waves between x0 and x1, and the number
# # of intervals between each discrete point (nint).
# # So you set all these as arguments of the function. This function can also
# # have a graphical output of one plot (which can be subdivided if necessary
# # using par(mfrow)). And the function can return output.
#
# fun <- function(delta = 1, pos = 1, phase = 1.5, nwave = 1, nint = 50)
# {
#
#   res <- sinpoint(1, 0, delta = delta, pos = pos, phase = phase,
#                   nwave = nwave, nint = nint)
#
#   plot(res$x,res$y)
#
#   return(res)
#
# }
#
# # Once this simple function is coded, it can be integrated to neatPick(). The
# # argument n defines to number of different realisations of the function.
#
# # WHEN YOU ARE HAPPY WITH THE OUTPUTS, click on 'END & RETURN ARGUMENTS'
#
# a <- neatPick(fun, n = 10, args.only = TRUE)
#
# # If you have clicked right (on the 'END & RETURN ARGUMENTS' button), the
# # arguments will be returned and stored in a;
#
# a
#
# # These arguments can then serve for a more efficient function:
#
# seg <- sinpoint(1, 0, delta = a$delta, pos = a$pos, phase = a$phase,
#                 nwave = a$nwave, nint = a$nint)
#
# # Basically neatPick applies a for loop to fun, but if you work on a large
# # dataset, you can also create a function that can handle the arguments more
# # efficiently. This is what sinpoint does here
#
# # Now you can see the results imported in R and do whatever you want with:
#
# plot(seg$x, seg$y, type = "n")
#
# multilines(seg$i, seg$x, seg$y)
```

```
#
# # You can even rework your initial changes:
#
# b <- neatPick(fun, n = 10, args.only = TRUE, args = a)
```

---

neatPicked *Runs neatPick without user input*

Description

Is the user input free version of neatPick. Runs a function n times, with its arguments n times different. The graphical output is stored into a n pages pdf or a n files folder. The output of the function is accumulated in a list.

Usage

```
neatPicked(fun, n, args = NA, width = 10, height = 10,
  output = "all", name = "Fig", dir = getwd(), gfile = "onePDF",
  openfile = TRUE, track = TRUE, folder = "My file", gfun = "jpeg",
  ext = ".jpeg", gargs = list(units = "in", res = 300),
  pargs = list())
```

Arguments

fun	the function to be applied n times.
n	number of runs.
args	the arguments to be supplied to fun. Should be a list of each argument to be supplied to fun, having n elements stored indiscriminately in list or in vector form.
width, height	the width and height of the graphics region. In inches by default, can be adapted if onePDFfile = FALSE
output	the kind of output : "function" for the accumulated outputs of the function (list of n elements), "all" to add args, and everything else to output nothing
name	the names of the graphic file(s)
dir	the directory of the file or of the folder of files
gfile	whether to create a single pdf with n pages ("onePDF"; default) or a folder of n graphical files ("gfun"). If anything else is given ("none for instance"), it won't produce graphical files. This reduces computation speed by a little more than 15 percents (one try of 1000 samples with simple graphs).
openfile, track	parameters for pdfDisplay()
folder	the name of the folder containing the n graphical files

gfun	a non-empty character string naming the graphical function to be called to create the n graphical files
ext	the extension of the n graphical files
gargs	list of arguments transmitted to the graphical function
pargs	list of arguments transmitted to the par() function

Value

the accumulated outputs of fun (and arguments if asked) if asked

Examples

```
# # To run example uncomment all: put in a script, select all and use
# #  ctrl+shift+c
#
# fun <- function(x, y, xlim = c(-1,1),...)
# {
#   plot(x, y, xlim = xlim,...)
#
#   return(paste(x, y, paste(xlim, collapse = "; "), sep = "; "))
# }
#
# args <- list(x = list(-0.5, 1) , y = c(0.8, 0.8), pch = c(2,4),
#              xlim = list(c(-1,1), c(-20,20)))
#
# temp <- tempfile()
# dir.create(temp)
#
# neatPicked(fun, 2, args = args, width = 5, height = 5, dir = temp)
```

---

nlegend *New legend element*

Description

Prepares a plotting environment for a new element of a multifigure legend

Usage

```
nlegend(temp = FALSE, t = "Text", xt = 1.3, xmax = 5,
  xmin = -1.2, ymax = 1.5, ymin = -ymax)
```

Arguments

temp	whether to plot a template for visualisation
t	text to provide the legend
xt	the x position of the text
xmin, xmax, ymin, ymax	the x and y limits for the plotting area

See Also

multigons, bedtext, infobar and ylink

Examples

```
opar <- par("mar")

par(mar = c(0,0,0,0))

layout(matrix(1:6, 6, 1, byrow = TRUE))

nlegend(t = paste("Shaded stuff. By the way you can\nwrite",
                  "text in several lines if needed"))

rect(-1,-1,1,1, density = 10)

nlegend(TRUE, t = "Text: left side at x = 1.3 (default xt value)")

par(mar = opar)
```

---

pdfDisplay *Generates PDF and SVG figures*

Description

Takes an ensemble of figures, represented by a function g(), and generates a PDF (or SVG if specified). The PDF can be visualised immediatly on the default PDF reader.

Usage

```
pdfDisplay(g, name, ext = ".pdf", dir = getwd(), width = 10,
  height = 10, parg = list(), track = T, openfile = T,
  output = F, warn = F)
```

Arguments

g	the plot function to be exported and looked at
name	the name of the document
ext	the extension of the document: ".pdf" by default, but ".svg" works also.
dir	the file where the document will be saved (by default the working directory, getwd())
width	the width of the drawing area (in inches)
height	the height of the drawing area (in inches)
parg	list of arguments transmitted to the par() function
track	whether to generate different files for each rerun of pdfDisplay with identical 'name'. The name will be followed by '_(i)' where i is the version number. With this you avoid closing your pdf file at each rerun if your pdf reader is not able to deal with (to my knowledge only SumatraPDF is able)
openfile	should the pdf file be opened (for the moment works only in Windows). Use SumatraPDF as default pdf reader to be able to write over current file
output	whether to output the output of g() or not
warn	useless vestigial parameter, kept for compatibility with StratigrapheR 0.0.1

Details

The width and height you provide will not exactly be respected. I could not find a pdf printing function that respects dimensions scrupulously for R base graphics.

Value

the output of the g() function if output = TRUE

Examples

```
# # To run example uncomment all: put in a script, select all and use
# #  ctrl+shift+c
# temp <- tempfile()
# dir.create(temp)
#
# g1   <- function() plot(1,1)
#
# pdfDisplay(g1(),"TestGraph", dir = temp,
#            parg = list(mar = c(6,6,6,6), ps = 24,lwd = 4))
#
# g1   <- function() plot(1,1, col = "red")
#
# pdfDisplay(g1(), "TestGraph", dir = temp,
#            parg = list(mar = c(6,6,6,6), ps = 24,lwd = 4))
#
```

placesvg *Draws a pointsvg object*

Description

Draws a svg object imported as data frame using pointsvg, with its importation coordinates (or with standardisation).

Usage

```
placesvg(object, forget = NULL, front = NULL, back = NULL,
  standard = FALSE, keep.ratio = FALSE, col = NA, border = "black",
  density = NULL, angle = 45, lwd = par("lwd"), lty = par("lty"),
  scol = border, slty = lty, slwd = lwd)
```

Arguments

object	a pointsvg object (svg object imported as data frame using pointsvg).
forget	the elements that should be discarded, by their id or index (i.e. name or number of appearance).
front, back	the elements to be put in front and back position, by their id or index (i.e. name or number of appearance). By default the order is the one of the original .svg file.
standard	whether to standardise (centre to (0,0), rescale so that extreme points are at -1 and 1) or not (T or F)
keep.ratio	if the object is to be standardised, whether to keep the x/y ratio (T or F)
col	the polygones background color. If density is specified with a positive value this gives the color of the shading lines.
border	the lines color.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise)
lty, lwd	the border line type and width, see ?par for details.
scol, slty, slwd	the colour, type and width of the shading lines.

See Also

centresvg and framesvg

Examples

```
object <- example.ammonite

plot(c(-2,2), c(-2,2), type = "n")

placesvg(object, lty = 1,density = 20, angle = 45)
```

`planepoints` *Describes planes by points*

Description

Gives the coordinates (dec and inc) of three perpendicular directions to describe planes.

Usage

```
planepoints(strike, dip, quadrant = NA, inverted = NA)
```

Arguments

`strike`	strike of the data; it is the angle from the north of the horizontal line of the plane. It is corrected by dipfix().
`dip`	dip of the data; it is the angle from the horizontal taken on the line of the plane perpendicular to the one of the strike. It is corrected by dipfix().
`quadrant`	the quadrant were the plane dips downward. Accepted values are NA, 'N', 'S', 'W' or 'E' (lower- or uppercase alike) for correction by dipfix().
`inverted`	whether the plane is inverted or not. The default is NA, it assumes that no bed is inverted.

Details

The directions are x for dip-direction line (direction of maximum downward dip), y for the horizontal line, z for the upper pole; additionally a magnitude is given to use y as a rotation axis to get the plane back at the horizontal. If the plane is inverted, y, z and mag will be changed, accordingly, with a rotation of 180° around x for y and z.

Value

a list of x, y and z declinations and inclinations (dec and inc), and a rotation magnitude

Examples

```
strike   <- c(-60, 180,20)
dip      <- c(-60,20,-45)
quadrant <- c("N","W",NA)
inverted <- c(FALSE,FALSE,TRUE)

res <- planepoints(strike,dip,quadrant,inverted)

deci <- c(res$x$dec, res$y$dec, res$z$dec)
inci <- c(res$x$inc, res$y$inc, res$z$inc)

earnet()

earplanes(strike,dip,quadrant,hsphere = "l")
earpoints(deci,inci)
```

`pointsvg` *Converts line, rect, polygon and polyline class SVG objects into data frames*

Description

Converts 'line', 'rect', 'polygon' and 'polyline' class SVG objects into data frames. **ONLY THESE CLASSES OF OBJECTS CAN BE IMPORTED.** If you have bezier or spline curves, they will be stored as 'path' class objects that cannot be imported here. The same goes for 'rect' objects that are transformed (rotation, etc...).

Usage

```
pointsvg(file, standard = TRUE, keep.ratio = FALSE, round = TRUE,
  xdigits = 4, ydigits = 4, xinverse = FALSE, yinverse = TRUE,
  warn = T)
```

Arguments

`file`	a .svg file
`standard`	whether to standardise (centre to (0,0), rescale so that extreme points are at -1 and 1) (T or F)
`keep.ratio`	if the object is to be standardised, whether to keep the x/y ratio (T or F)
`round`	whether to round the coordinates (T or F)
`xdigits`	the number of digits after the decimal to round to for x values
`ydigits`	the number of digits after the decimal to round to for y values
`xinverse`	whether to inverse the plotting for x values (T or F)
`yinverse`	whether to inverse the plotting for y values (T or F)
`warn`	whether you want to be annoyed

Details

This function is quite empirical. There is no guarantee it is bug free. If you have .svg files that should work but do not, you can email me: <sébastien.wouters@doct.uliege.be>

Value

A data.frame with x and y coordinates, ids for each object, and a type, either line (L) or polygon (P)

See Also

Plot the drawing: placesvg,

Plot the drawing and change the coordinates :centresvg and framesvg

Change the drawing: changesvg and clipsvg

Examples

```
# To show you how to import, we first have to have a svg file to import. The
#following lines of code will create a svg in a temporary files:

svg.file.directory <- tempfile(fileext = ".svg") # Creates temporary file
writeLines(example.ammonite.svg, svg.file.directory) # Writes svg in the file

print(paste("An example .svg file was created at ", svg.file.directory,
            sep = ""))

# The pointsvg function allows to import simple svg drawings into R

ammonite.drawing <- pointsvg(file = svg.file.directory) # Provide file

plot(c(-2,2), c(-2,2), type = "n")

placesvg(ammonite.drawing)

# If you want to import your own .svg file uncomment the following line:

# pointsvg(file.choose())
```

---

rebound *Simplifies boundary indicators for lim objects*

Description

Simplifies boundary indicators for lim objects: from the wide range supported by R ("[]", "[)", "(]", "()", "[[", "]]", "][", "open", "closed", "left-open", "right-open", "left-closed", "right-closed") to "[]", "[[", "]]" and "][" only

Usage

```
rebound(b, na.errors = F)
```

Arguments

b	a vector of boundary indicators
na.errors	whether to replace all other values by NA (rather than simply stopping the function)

Value

a simplified vector of boundary indicators ("[]", "[[", "]]" and "][" only)

See Also

as.lim

Examples

```
bounds <- c("[]", "[)", "(]", "()",
            "[[", "]]", "][",
            "open", "closed",
            "left-open", "right-open",
            "left-closed", "right-closed")

rebound(bounds)
```

---

repitch *Converts pitch into declination and inclination*

Description

Finds the declination and inclination of a line defined by a pitch on a plane

Usage

```
repitch(pitch, strike, dip, quadrant = NA)
```

Arguments

pitch	pitch (or rake) of the data; it is the angle between the strike of the plane and a line. It is taken from the left side going downward along the dip, and is positive downward.
strike	strike of the data; it is the angle from the north of the horizontal line of the plane. It is corrected by the dipfix function.

dip	dip of the data; it is the angle from the horizontal taken on the line of the plane perpendicular to the one of the strike. It is corrected by the dipfix function.
quadrant	the quadrant were the plane dips downward. Accepted values are NA, 'N', 'S', 'W' or 'E' (lower- or uppercase alike) for correction by the dipfix function.

Value

a list of declination and inclination of the defined lines

References

Eric Carlson of the Colorado School of Mines is acknowledged for his rake to plunge calculator on which this function is based.

See Also

dipfix, incfix and transphere

Examples

```
strike <- c(90, 135, 135, 135)
dip    <- c(0,  65,  65,  65)
pitch  <- c(40, 40,  140, -40)

earnet()
earplanes(strike,dip,hsphere = "b", a = list(col = "red", lwd = 2))

res <- repitch(pitch = pitch, strike = strike, dip = dip)

earpoints(dec = res$dec, inc = res$inc)
```

---

reposition *Core correction*

Description

Core correction : declination and inclination are corrected for cores of given declination, inclination and rotation

Usage

```
reposition(dec, inc, cdec = 0, cinc = 90, crot = 0)
```

Arguments

dec	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010).
inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010).
cdec	declination of the core.
cinc	inclination of the core.
crot	rotation of the core; it is the angle of rotation around the core direction clockwise between the measurement and the actual core orientation. In others words it is the magnitude of the rotation to apply clockwise to the measured data using the core direction as an axis.

See Also

rotate and restore

Examples

```
# ----

d <- zeq_example

dec <- d$Dec
inc <- d$Inc

cdec <- 75
cinc <- 45
crot <- 90

par(mfrow = c(2,2))

earnet()
earpoints(dec,inc)
earpoints(0, 90, l = list(cex = 2))
earpoints(0, 90, l = list(col = "red", bg = "red"))
title("1. Laboratory projection,
      axis for rotating the specimen")

# Roll ----

roll <- reposition(dec, inc, crot = 90)

earnet()
earpoints(roll$dec,roll$inc)
earpoints(0, 90, l = list(cex = 2))
earpoints(90, 0, h = list(col = "red", bg = "pink"), double = TRUE)
title("2. Correction of the specimen rotation,
      in red the axis for tilting the specimen")
```

```
# Tilt ---

tilt <- reposition(dec, inc, cinc = cinc ,crot = crot)

earnet()
earpoints(0, cinc, l = list(cex = 2))
earpoints(tilt$dec, tilt$inc)
earpoints(0,90, l = list(col = "red", bg = "red"))
title("3. Correction of the specimen inclination,
      in red the axis for rotating the tilted specimen")

# Orient ---

orient <- reposition(dec, inc, cdec = cdec, cinc = cinc ,crot = crot)

earnet()
earpoints(cdec, cinc, l = list(cex = 2))
earpoints(orient$dec, orient$inc)
title("4. Full geographical repositioning,
      the big dot is the core orientation")

par(mfrow = c(1,1))

# ----
```

`restore` *Plane correction*

Description

Plane correction : declination and inclination are corrected for planes of given strike, dip, quadrant and inversion

Usage

```
restore(dec, inc, strike, dip, quadrant = NA, inverted = NA,
  percent = 100)
```

Arguments

`dec`	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010).
`inc`	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010).
`strike`	strike of the plane used for correction; it is the angle from the north of the horizontal line of the plane. It is corrected by dipfix().

dip	dip of the plane used for correction; it is the angle from the horizontal taken on the line of the plane perpendicular to the one of the strike. It is corrected by dipfix().
quadrant	the quadrant were the plane dips downward. Accepted values are NA, 'N', 'S', 'W' or 'E' (lower- or uppercase alike) for correction by dipfix().
inverted	whether the plane is inverted or not. The default is NA, it assumes that no bed is inverted.
percent	the percentage of correction (can be of length >= 1), by default it is 100 (%), bringing the plane to the horizontal.

See Also

rotate and reposition

Examples

```
dec <- c(90,210)
inc <- c(20,60)

strike <- c(0,120)
dip    <- c(20,60)
inverted <- c(FALSE,TRUE)

res <- restore(dec = dec, inc = inc, strike = strike, dip = dip,
               quadrant = NA, inverted = inverted,
               percent = seq(20,100, by = 20))

earnet()
earplanes(strike, dip)
earpoints(dec,inc)
earpoints(round(res$dec,2), round(res$inc,2), a = list(pch = 22))
```

---

rmatrix *Rotation matrix*

Description

Computes a rotation matrix for a given rotation axis and angle based on Tauxe et al. (2010).

Usage

```
rmatrix(dec, inc, mag, as.data.frame = FALSE)
```

Arguments

dec	declination of the rotation axis; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010).
inc	inclination of the rotation axis; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010).
mag	magnitude of rotation (following the notation of the Stereonet software) a positive rotation is clockwise looking in the direction of the given declination and inclination)
as.data.frame	logical, whether to output the matrix as a data frame. This is used when multiple arguments are provided to simplify and boost calculations.

References

- Tauxe, L., 2010. Essentials of Paleomagnetism. University of California Press.
- Allmendinger, R. W., Cardozo, N. C., and Fisher, D., 2013, Structural Geology Algorithms: Vectors & Tensors: Cambridge, England, Cambridge University Press, 289 pp.
- Cardozo, N., and Allmendinger, R. W., 2013, Spherical projections with OSXStereonet: Computers & Geosciences, v. 51, no. 0, p. 193 - 205, doi: 10.1016/j.cageo.2012.07.021

Examples

```
rmatrix(135,20,60)

rmatrix(c(135,0),c(20,90),c(60,90), as.data.frame = TRUE)
```

---

rotate *Spherical rotation around fixed axes*

Description

Spherical rotation around given rotation axes

Usage

```
rotate(dec, inc, rdec, rinc, rmag)
```

Arguments

dec	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010).
inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010).

rdec	declination of the rotation axes (of length 1 or n).
rinc	inclination of the rotation axes (of length 1 or n).
rmag	magnitude of rotation (following the notation of the Stereonet software): a positive rotation is clockwise looking in the direction of the given declination and inclination; of length 1 or n).

References

- Tauxe, L., 2010. Essentials of Paleomagnetism. University of California Press.
- Allmendinger, R. W., Cardozo, N. C., and Fisher, D., 2013, Structural Geology Algorithms: Vectors & Tensors: Cambridge, England, Cambridge University Press, 289 pp.
- Cardozo, N., and Allmendinger, R. W., 2013, Spherical projections with OSXStereonet: Computers & Geosciences, v. 51, no. 0, p. 193 - 205, doi: 10.1016/j.cageo.2012.07.021

See Also

rmatrix, restore and reposition

Examples

```
earnet()

inc <- seq(0,85,5)
dec <- rep(0,length(inc))

earpoints(dec,inc)

rdec <- rep(0, length(inc))
rinc <- rep(90, length(inc))

mag <- 90
rmag  <- seq(mag, 0, by = -mag/(length(inc)-1))

rot <- rotate(dec,inc,rdec,rinc,rmag)

earpoints(dec = round(rot$dec,digits = 2), inc = round(rot$inc,digits = 2),
          l = list(bg = "green"),
          u = list(bg = "blue"),
          h = list(bg = "yellow"))

earpoints(dec = 0, inc = 90, l = list(bg = "red"))
```

seq_log — *Gives the repartition of values for a log 10 scale between a given interval*

Description

Gives the repartition of values for a log 10 scale between a given interval

Usage

```
seq_log(x1, x2, divide = FALSE)
```

Arguments

x1	the first value of the interval
x2	the second value of the interval (can be higher or lower, but never equal to x1)
divide	whether to divide the result for major values (1,10,100) and minor values (2,3,...,20,30,...)

Value

the repartition of values for a log 10 scale between x1 and x2

See Also

convertAxis

Examples

```
x1 <- 101
x2 <- 0.29

seq_log(x1, x2)
seq_log(x1, x2, divide = TRUE)
```

shift — *Circular shift*

Description

Circular shift

Usage

```
shift(x, n = 1L, names = T)
```

Arguments

x	a vector (characters, numerics, integers,...), data.frame or list
n	a positive integer of length 1, giving the number of positions to shift by (positive values generate lag)
names	whether the names of the elements or rows should also shift

Value

the same object than the input, but with a shifted order

Examples

```
# Simple use

shift(x = c(6,8,10,12,2,4), n = 2)

# Applications to litholog generation

l <- c(1,2,3)
r <- c(0,1,2)
h  <- c(4,3,4)
i   <- c("B1","B2","B3")

basic.litholog <- litholog(l,r,h,i)

whiteSet(xlim = c(0,4), ylim = c(0,3),
         xaxs = "r", yaxs = "r",   # This gives a little room to the graph
         ytick = 1, ny = 10)

multigons(basic.litholog$i, basic.litholog$xy, basic.litholog$dt,
          forget = "B1", lwd = 2)

openbed <- subset(basic.litholog, basic.litholog == "B1")

openbed <- shift(openbed, -1)

lines(openbed$xy, openbed$dt, lwd = 2)
```

---

simp.lim *Joins and orders adjacent or overlapping lim objects of same ID*

Description

Joins and orders adjacent or overlapping lim objects of same ID

Usage

```
simp.lim(lim = NULL, l = NULL, r = NULL, id = 1L, b = "[]")
```

Arguments

lim	an object convertible into a lim object: either a vector of length 2 or a list of n left (1st element) and n right (2ndt element) interval limits
l	a vector of n left interval limits
r	a vector of n right interval limits
id	a vector of n interval IDs (default is 1 for each interval)
b	a character vector for the interval boundaries rules: "[]" (or "closed") to include both boundaries points, "][" (or "()" and "open") to exclude both boundary points, "[[" (or "[)","right-open" and"left-closed") to include only the left boundary point, and "]]" (or "(]", "left-open", "right-closed") to include only the right boundary point. The notation is simplified to "[]", "[[", "]]" and "][" only.

Value

a lim object of the joined intervals

See Also

as.lim

Examples

```
l   <- c(50,2,4,6,50,8,50,51,50,80)
r   <- c(50,5,6,9,8,2,51,51,50,80)
id  <- c("i1", "i1", "i1", "i1", "i2","i2","i2","i2","i2","i2")
b   <- c("[]", "][", "][", "]]", "][","[[","][","][","][","][")

simp.lim(l = l, r = r, id = id, b = b)
```

---

sinpoint *Gives a table of equally sampled points following a sinusoidal function*

Description

Gives a table of equally sampled points following a sinusoidal function

Usage

```
sinpoint(x, y, delta, x0 = 0, pos = 1, phase = 1.5, nwave = 1,
  nint = 50)
```

Arguments

x	the x value of the end of the interval
y	the y offset (see next parameter)
delta	the difference between the min- and maxima in y
x0	the x value of the beginning of the interval (0 as default)
pos	an integer specifying the kind of vertical offset; should the sinusoidal function be shifted so that y is the first value (pos = 1, is the default), the last value (2),the minimum (3) or the maximum (4) of the function
phase	the phase of the function at x0 in multiples of pi (1.5 as default; begins at its lowest)
nwave	number of complete sinuses waves (1 as default)
nint	number of intervals for the sampling (50 as default)

Value

a table of points following a sinusoidal function

Examples

```
res <- sinpoint(c(4,5), 5, 1, x0 = c(0,1), pos = 3)

plot(res$x, res$y)

multilines(res$i, res$x, res$y, col = c("black" ,"red"), type = "o")
```

StratigrapheR *StratigrapheR: integrated stratigraphy for R*

Description

This package includes bases for litholog generation: graphical functions based on R base graphics (e.g. multigons()), interval gestion functions (with the as.lim() function, and other related .lim functions) and simple svg importation functions (e.g. pointsvg()) among others. It also includes stereographic projection functions (e.g. the earnet(), earpoints() and earplanes() functions; ear standing for equal area), and other functions made to deal with large datasets while keeping options to get into the details of the data. **IF YOU WANT TO START LEARNING HOW TO CREATE LITHOLOGS WITH STRATIGRAPHER GO SEE THE EXAMPLE BELOW.**

A StratigrapheR() function is provided: it generates organisational charts for common use of the functions in the package

Usage

```
StratigrapheR(i = 1:3)
```

Arguments

i	the index(es) of the organisational charts of the functions in the StratigrapheR package

Details

Package: StratigrapheR

Type: R package

Version: 0.0.6 (June 2019)

License: GPL-3

Note

If you want to use this package for publication or research purposes, please cite Wouters, S., Da Silva, A.C. Crucifix, M., Sinnesael, M., Zivanovic, M., Boulvain, F., Devleeschouwer, X., 2019, Litholog generation with the StratigrapheR package and signal decomposition for cyclostratigraphic purposes. Geophysical Research Abstracts Vol. 21, EGU2019-5520, 2019, EGU General Assembly 2019. <http://hdl.handle.net/2268/234402>

Author(s)

Sébastien Wouters

Maintainer: Sébastien Wouters <sébastien.wouters@doct.uliege.be>

Examples

```
# # To run example uncomment all: put in a script, select all and use
# #  ctrl+shift+c
#
# # This is an example of litholog generation script, along with some
# # explanations: if you want to start somewhere, start here. You may run the
# # whole thing and follow the explanations.
#
# library(StratigrapheR)
# library(dplyr) # very useful package, used here for joining data frames
#
# # You may want to change your working directory for this, the example will
# # generate .pdf and .txt files;
# # setwd()
#
# # If you want to have an organisational chart of the functions:
# pdfDisplay(StratigrapheR(), "Organisational Chart StratigrapheR",
#            width = 9, height = 7.5, track = FALSE)
#
# # Bed dataset ----
#
# bed.example
#
# # this dataset should include the description of each bed with :
```

```
# # - l - the position of the base of each bed (in cm or m) - l stands for the
# #   left side or boundary of an interval-
# # - r - the position of the top of each bed (in cm or m) - r stands for the
# #   right side or boundary of an interval-
# # - litho - the lithology, basics are for instance C for chert, S for shale, L
# #   for limestone... but you can include anything you want in any way you want
# # - h - relief or hardness of each bed
# # - id - is the bed identification, number (e.g. B1, B2, ...)
# #   you can also include other columns with anything else you find useful for
# #   each bed such as color or lithofacies
#
#
#
# # Ponctual elements datasets ----
#
# fossil.example
# boundary.example
# chron.example
#
# # These dataset(s) should include any ponctual information you want in the log,
# # such as the position of particular fossils, bioturbations, minerals, tectonic
# # features, etc...
#
# # We will also see how to add proxy information with:
#
# proxy.example
#
#
#
# # Work the datasets ----
#
# # Basic litholog (rectangles) --
# # it will take the basic data (l, r, h, id)
#
# basic.log <- litholog(l = bed.example$l, r = bed.example$r,
#                       h = bed.example$h, i = bed.example$id)
#
# # Define the legend for each lithology ----
# # for each lithology you can provide a color (col), a density of shading
# # (density) and orientation for the lines (angle)
#
# legend <- data.frame(litho = c("S", "L", "C"),
#                      col = c("grey30", "grey90", "white"),
#                      density = c(30, 0,10),
#                      angle = c(180, 0, 45), stringsAsFactors = FALSE)
#
# bed.legend <- left_join(bed.example,legend, by = "litho")
#
#
#
# # Plot a basic litholog ----
#
# # Be warned that the most efficient way to generate a litholog is to put it
```

```
# # in a function. We will see this lower in the exaplanation. The three first
# # lithologs generated in the R plot window are simply an example to help you
# # understand the functions in StratigrapheR
#
# # First prepare the plot using whiteSet(): this provides a clean drawing area
#
# whiteSet(xlim = c(0,10), ylim = c(-1,77), ytick = 5, ny = 5) # Prepare plot
# title("Using litholog() and bedtext()")
#
# # Then add the polygons making the litholog. This is done with a single function
# # identifying each polygon by the id of points. The graphical parameters of the
# # polygons can be adapted to fit the legend, polygon by polygon.
#
# multigons(basic.log$i, x = basic.log$xy, y = basic.log$dt,
#           col = bed.legend$col,
#           density = bed.legend$density,
#           angle = bed.legend$angle)
#
# # You can further add the name of each bed on each corresponding polygon
#
# bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
#         x = 0.5,  # x position where to centre the text
#         ymin = 3) # ymin defines the minimum thickness for the beds where text
#                   # will be added, making for a clean litholog
#
#
#
# # Vectorised drawing: example of importation ----
#
# # This creates a svg in one of your temporary files, to show how to import svg
# # files
# svg.file.directory <- tempfile(fileext = ".svg")
# writeLines(example.ammonite.svg, svg.file.directory)
# print(paste("An example .svg file was created at ", svg.file.directory,
#             sep = ""))
#
# # The pointsvg function allows to import simple svg drawings into R
# ammonite.drawing <- pointsvg(file = svg.file.directory)
#
# # If you want to import your own .svg file uncomment the following line:
# # pointsvg(file.choose())
#
# # Other data frames of vectorised drawings are imbedded into the
# # StratigrapheR package for this example : example.ammonite.svg (to see how to
# # use pointsvg), example.ammonite, example.belemnite and example.liquefaction
#
# # Now that ammonite.drawing is available, lets see what it looks like
#
# whiteSet(ylim = c(-1,1), xlim = c(-1,1)) # Plot
# box()
#
# title("ammonite.drawing")
#
```

```
# placesvg(ammonite.drawing)
#
# # The placesvg() function plots any pointsvg-like dataset, which is a data frame
# # with a column x, y, id (for each polygon or polyline) and type (polygone or
# # line). Note that only polygons and polylines drawings can be imported by
# # pointsvg()
#
# # You can see that the ammonite drawing is centred on 0,0, and has its maxima
# # and minima at 1 and -1 respectively, for x and y alike. To plot a drawing
# # at the right position and ratio, you can use the centresvg and framesvg
# # functions
#
# # For that you have to provide information about the position, for instance:
#
# y.ammonite <- fossil.example$dt[fossil.example$type == "ammonite"]
# y.ammonite
#
# # y.ammonite is the y position (or depth) where each ammonite should be drawn.
# # It is provided via a vector of any length (i.e. you can have any number of y
# # positions and of corresponding ammonites), as long as all the other parameters
# # are of length 1 or of same length (i.e. you could provide two values for x if
# # you want the two ammonite drawings to have a different x position)
#
# # First build the log
#
# whiteSet(xlim = c(0,10), ylim = c(-1,77), ytick = 5, ny = 5)
# title("Using pointsvg() and centresvg()")
#
# multigons(basic.log$i, x = basic.log$xy, y = basic.log$dt,
#           col = bed.legend$col,
#           density = bed.legend$density,
#           angle = bed.legend$angle)
#
# bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
#         x = 0.5,  ymin = 3)
#
# # Then add the drawings
#
# centresvg(ammonite.drawing,
#           x = 7, # this is an arbitrary x position for each ammonite drawing
#           y = y.ammonite,
#           xfac = 0.75,   # Correction factor for the ratio in x
#           yfac = c(3,5)) # Correction factor for the ratio in y. As the other
#                          # parameters it can be adapted for each drawing
#                          # individually
#
# # The centresvg() function will take a data frame outputted by pointsvg() - or
# # from changesvg(), and even centresvg() and framesvg() if the output is TRUE as
# # these two functions can output drawings with modified coordinates -.
#
#
#
# # Dealing with bed thickness changes ----
```

```
#
# # You can also weld changes of bed thickness at bed boundaries to the basic log
#
# # For instance we can define here two types of sinuosidal boundaries. If you
# # want you can even design a different type of 'wiggle' for each boundary.
#
# s1 <- sinpoint(5,0,0.5,nwave = 1.5)
# s2 <- sinpoint(5,0,1,nwave = 3, phase = 0)
#
# # You can also weld lines you have drawn in svg and imported with pointsvg().
# # However there are a few rules to use them as boundaries in StratigrapheR:
# # you have to think about their coordinates. The function welding the 'wiggles'
# # of the boundaries to the rectangles of the log, weldlog(), will require to set
# # what you consider to be the beginning of the wiggle (at the left of the
# # litholog) at 0,0 (if you run with the default parameters of weldlog, which is
# # advised if you start), and define their coordinates to suit the scale of the
# # litholog
#
# # You can use centresvg() or framesvg() to change the coordinates, setting the
# # output argument to TRUE (and the plot argument to FALSE if you don't want to
# # plot)
#
# s3 <- framesvg(example.liquefaction, 1, 4, 0, 2, plot = FALSE, output = TRUE)
#
# # In framesvg(), rather than providing the point to center the drawing on, and
# # correction in x and y (as centresvg does), you provide the maxima and minima
# # in x and y
#
# # With the function wedlog, we combine the lithological log we created
# # (basic.log) with the wavy bed boundaries we created. We provide the log
# # -parameter log-, the position of the joints we would lie to change -dt-, the
# # segments that are going to be welded to the basic log -seg, as a list of
# # data frames, by default having the first column for the xy coordinates and
# # second for dt coordinates- and j making the link between the boundaries
# # position -dt- and the segments -seg-.
#
# # For each j corresponds a respective dt of same index (for each dt corresponds
# # a j at the same position), and each j refers to the index or the name of a
# # segment in the list of segments.
#
# # with the function wedlog, we combine the lithological log we created
# # (basic.log) with the wavy bed boundaries we created. So you can use any
# # wiggle you define on your own and weld it to the log
#
# final.log <- weldlog(log = basic.log, dt = boundary.example$dt,
#                      seg = list(s1 = s1, s2 = s2, s3 = s3),
#                      j = c("s1","s1","s1","s3","s2","s2","s1"))
#
# # Lets see the result of the welding
#
# whiteSet(xlim = c(-3,8), ylim = c(-1,77), ytick = 5, ny = 5) # Prepare plot
#
# # This plot is going to serve to explain other functions;
```

```
# title("Using weldlog(), infobar(), simp.lim() and minorAxis()")
#
#
# multigons(final.log$i, x = final.log$xy, y = final.log$dt,
#           col = bed.legend$col,
#           density = bed.legend$density,
#           angle = bed.legend$angle)
#
# bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
#         x = 0.5, ymin = 3)
#
#
#
# # Defining and drawing specific intervals  ----
#
# # Lets say we would like to plot the position of magnetochrons. For that we
# # firstly define a legend for each type of interval, here for normal and reverse
# # polarity
#
# legend.chron <- data.frame(polarity = c("N", "R"),
#                            bg.col = c("black", "white"),
#                            text.col = c("white", "black"),
#                            stringsAsFactors = FALSE)
#
# # Then we set the legend for each chron
# chron.legend <- left_join(chron.example,legend.chron, by = "polarity")
#
# # There are three chrons, but what we did can be applied to any number of them,
# # as long as they are identified by a column (or more, left_join can merge using
# # more than one column)
#
# # Using this legend we can draw rectangles with text in it using the infobar()
# # function. In this function we define the coordinates of each rectangle
# # (linked to dt for y, and different for each rectangle, but constant in x)
# # the text to be in the rectangles with the labels parameter, and graphical
# # parameters to be used by the multigons() and text() functions embedded in the
# # infobar() function. The number of rectangles is n, and the length of the y, x,
# # and labels elements can be 1 or n (i.e. the same n for each parameter).
#
# # You can provide a list of graphical parameters such as the colour for the
# # rectangles and the text, as long as the length of each parameter
# # in that list is 1 or n.
#
# # Notice that this function shares has a lot in common with litholog() and
# # multigons() in functionality and arguments. Note that you could obtain a
# # similar result using litholog(), multigons() and bedtext(). You would simply
# # need to code more :-)
#
# infobar(-2.5, -2, chron.legend$l, chron.legend$r,
#         labels = chron.legend$polarity,
#         m = list(col = chron.legend$bg.col),
#         t = list(col = chron.legend$text.col),
#         srt = 0)
```

```
#
#
#
# # Treat data sets made of intervals (as happens a lot in geology) ----
#
# # As you have seen with litholog, intervals are dealt with by defining lim
# # objects having a left and right boundary (l and r), an id and a boundary rule.
# # Whichever of l and r is the maxima or minima usually does not
# # matter. StratigrapheR offers a few functions to treat lim objectss. Here
# #  we will see the simp.lim() function, but if you want more info go see the
# # ?as.lim help page, and the functions in its See Also part.
#
# # simp.lim: this functions merges intervals of same id (if adjacent or
# # overlapping)
#
# # Basically, the lim objects are boundaries, for instance in the form [0,1[
# # which would indicate an interval going from 0 to 1, zero included but 1 not.
# # simp.lim takes the left and right boundaries, assumes that each boundary
# # is included in the interval (by default b = "[]"), and simplifies the interval
# # by merging them by id, which gives the litholical information in merged
# # rectangles (with S, C and L indicating shales, cherts and limestones in this
# # case).
#
# litho.intervals <- simp.lim(l = bed.legend$l, r = bed.legend$r,
#                       id = bed.legend$litho)
#
# # The resulting list needs to be transformed into a data frame to merge with the
# # legend.
#
# litho.intervals <- data.frame(litho.intervals, stringsAsFactors = FALSE)
#
# # Note the parameter stringsAsFactors that is set to FALSE, which is usually
# # required when you create data frames to avoid problems, for instance using
# # left_join()
#
# colnames(litho.intervals)[3] <- "litho" # Change a column name to be able to merge
#                                  # legend and data
#
# litho.intervals.legend <- left_join(litho.intervals,legend, by = "litho")
#
# infobar(-1.25, -0.75, litho.intervals.legend$l, litho.intervals.legend$r,
#         m = list(col = litho.intervals.legend$col,
#                  density = litho.intervals.legend$density,
#                  angle = litho.intervals.legend$angle))
#
# # As you can see if you look closely at the "Using weldlog(), infobar() and
# # simp.lim()" plot, the subdivisions between beds of same lithology is gone.
# # This is the result of the simp.lim() function by interval manipulation
#
#
#
# # Add sample position with axis ----
#
```

```
# # If you want you can also show where every sample is using the minorAxis()
# # function, which allows disticntion between major and minor ticks
#
# at.min <- every_nth(proxy.example$dt, 5, empty = FALSE)
# at.maj <- every_nth(proxy.example$dt, 5, inverse = TRUE, empty = FALSE)
# labels.maj <- every_nth(proxy.example$name, 5, inverse = TRUE, empty = FALSE)
#
# # The every_nth function allows here to skip samples regularly (to avoid having
# # too much text)
#
# minorAxis(side = 4,                # Right-sided axis
#           at.min = at.min,         # dt/y position of minor ticks
#           at.maj = at.maj,         # dt/y position of major ticks
#           labels.maj = labels.maj, # Text to add at major ticks
#           tick.ratio = 0.5,        # Length ratio between minor and major ticks
#           pos = 6,                 # x position
#           las = 1,                 # Orientation of text
#           lwd = 0 ,                # Width of axis line to 0 removes the line
#           lwd.ticks = 1)           # Width of axis ticks to 1 to keep the ticks
#
#
#
#
# # Final litholog generation: getting it in a convenient function ----
#
# # Once the final design for the lithology is established, it can be integrated
# # into a graphical function which will draw every component of the final
# # litholog with each desired feature.
#
# # The most efficient way to generate the litholog is to directly put it in a
# # reusable function so that you do not do all the work twice. However you need
# # some of the data sets we've prepared, in this case bed.example,
# # fossil.example, boundary.example, chron.example (that are already imbedded
# # in StratigrapheR), final.log, bed.legend, chron.legend and litholeg (that
# # are created in this script)
#
# # If you do not want to run all unnecessary functions whenever you want to draw
# # your log, a good trick is to save all the necessary data.frames needed in
# # the litholog drawing function (here one.log) and load them in it. You just
# # need to have the saving file (here one.log.txt) in your working directory (see
# # ?setwd and ?getwd help pages for further help on that)
#
# save(final.log, bed.legend, chron.legend, litho.intervals.legend,
#      file = "one.log.txt")
#
# one.log <- function(xlim = c(-2.5,7), ylim = c(-1,77),
#                     xarg = NULL,  # this is transmitted to whiteSet: if set to
#                                   # NULL its allows to avoid drawing the x axis
#                     yarg = list(tick.ratio = 0.5, las = 1),
#                     main = "Final litholog")
# {
#   load("one.log.txt") # Load the saved data frames
#
```

```
#   whiteSet(xlim = xlim, ylim = ylim, ytick = 5, ny = 5,
#            xarg = xarg, yarg = yarg)
#
#   title(main = main)
#
#   multigons(final.log$i, x = final.log$xy, y = final.log$dt,
#             col = bed.legend$col,
#             density = bed.legend$density,
#             angle = bed.legend$angle)
#
#   bedtext(labels = bed.example$id, l = bed.example$l, r = bed.example$r,
#           x = 0.5, edge = TRUE)
#
#   centresvg(example.ammonite, 6,
#             fossil.example$dt[fossil.example$type == "ammonite"],
#             xfac = 0.5)
#
#   centresvg(example.belemnite, 6,
#             fossil.example$dt[fossil.example$type == "belemnite"],
#             xfac = 0.5)
#
#   infobar(-2, -1.5, chron.legend$l, chron.legend$r,
#           labels = chron.legend$id,
#           m = list(col = chron.legend$bg.col),
#           t = list(col = chron.legend$text.col))
#
#   infobar(-1, -0.5, litho.intervals.legend$l, litho.intervals.legend$r,
#           labels = litho.intervals.legend$litho, srt = 0)
# }
#
# # This graphical function can then be used as a standalone function, or
# # integrated in a for loop to draw the entirety in a succesion of panels
# # (typically in pdf form)
#
# # Indeed, if you go back to the definition of the one.log() function, you can
# # see that we gave it a parameter, ylim. That parameter defines the range of dt
# # that is covered in the plot. So you can plot a smaller part of the log:
#
# one.log(ylim = c(18,53), main = "Final litholog from dt 18 to 53")
#
# # Or you can create a second function that creates a loop of the log if you want
# # to generate an ensemble of sheets that placed end to end would create a
# # complete litholog
#
# # Basically can want to set up the scale (i.e. the y -or dt- interval of the
# # litholog seen for each plot -or pdf page-: if you want to see each time an
# # interval of 30 y-units of the litholog on each plot/pdf page, can set the
# # parameter 'interval' of the following function to 30)
#
# repeated.log <- function(start = 0, interval = 20)
# {
#   omar <- par("mar")
#
```

```
#   par(mar = c(1,4,3,2)) # This allows to define the margins as you wish
#
#   l1 <- seq(start,max(final.log$dt),interval)
#   l2 <- seq(start,max(final.log$dt),interval) + interval
#
#   for(i in length(l1):1)
#   {
#     one.log(ylim = c(l1[i],l2[i]),
#             main = paste("Repeated litholog, part from dt", l1[i], "to", l2[i]))
#   }
#
#  par(mar = omar)
#
# }
#
# repeated.log()
#
# # Printing and seeing you litholog in pdf ----
#
# # The next function, pdfDisplay, generates a pdf of a graphical function.
# # Any function producing plots such as repeated.log() can be inserted into it to
# # generate plots. These plots will all be of the same size. I believe this
# # function might not work on every computer. And its openfile argument, which
# # causes the pdf to open, only works in Windows. If You are working with
# # Windows, I recommend using SumatraPDF as your default pdf reader: this will
# # allow pdfs to be changed while they are being visualised.
#
# pdfDisplay(repeated.log(), width = 10, height = 15,
#            name = "StratigrapheR_Example_a", track = FALSE)
#
#
#
# # Plotting data -e.g. time-series data of a proxy - along the litholog ----
#
# # Now lets say you want to plot information along the litholog. For that we will
# # work in a graphical function that we will provide to pdfDisplay. Note that
# # it is not possible to base yourself on the repeated.log() function, because
# # it will print all the plots succesively without allowing modification or
# # addition
#
# # One way of working is to create two plots next to each other and provide
# # identical y axis parameters
#
# graphical.function.1 <- function()
# {
#
#   opar <- par("mar","mfrow")
#
#   par(mar = c(3,4,3,2),
#       mfrow = c(1,2)) # This creates two windows where to plot sucessively
#
#   # Plot the litholog on the left
#
```

```
#   one.log(main = "")
#
#   # Plot the other data on the right
#
#   blackSet(xlim = c(-2*10^-8,8*10^-8),
#            ylim =  c(-1,77), # It is important to define identical y limits
#                              # between the litholog and the proxy
#            ytick = 5, ny = 1,
#            targ = NULL)
#
#   lines(proxy.example$ms, proxy.example$dt, type = "o", pch = 19)
#
#   par(mar = opar$mar, mfrow = opar$mfrow)
#
# }
#
# pdfDisplay(graphical.function.1(), width = 10, height = 15,
#            name = "StratigrapheR_Example_b", track = FALSE)
#
# # If you want to put that repeated litholog in A4 format, the best way is to
# # use LaTeX. The following lines of code will create a TeX file that would
# # do that, test it if you want:
#
# writeLines(log.loop.tex, paste(getwd(),"log.loop.tex", sep = "/"))
#
# # Another way to work this out is to create more space than needed on the
# # litholog plot and to add elements
#
# graphical.function.2 <- function()
# {
#   omar <- par("mar")
#
#   par(mar = c(3,4,3,2))
#
#   # Plot the litholog with room for the rest
#
#   one.log(main = "", xlim = c(-3,16), xarg = list())
#
#   par(fig = c(0.5,1, 0, 1),  # 'fig' defines the overlapping plotting window
#                              # dimensions x1, x2, y1 and y2
#       new = TRUE)               # 'new' allows addition to a preexisting plot
#
#   # The graphical parameter 'fig' that you can set using the par() function
#   # allows you to define a new plotting region overlapping the original one.
#   # This allows you to redefine x axes values. But again using this you have to
#   # be careful to provide the right y limits between the litholog and the proxy.
#   # Be aware that the functions white-, black- and greySet() set the xaxis and
#   # yaxis to "i", which means that the limits you provide in x and y are the
#   # actual limits of the plot (while the default setting of xaxis and yaxis are
#   # "r", which extends the data range by 4 percent at each end)
#
#   blackSet(xlim = c(-2*10^-8,8*10^-8),
#            ylim =  c(-1,77),
```

```
#            ytick = 5, ny = 1,
#            targ = NULL,
#            xarg = list(side = 3))
#
#   lines(proxy.example$ms, proxy.example$dt, type = "o", pch = 19)
#
#   par(mar = omar)
#
# }
#
# pdfDisplay(graphical.function.2(), width = 8, height = 15,
#            name = "StratigrapheR_Example_c", track = FALSE)
#
```

`StratigrapheR.examples` *Data for examples*

Description

Supporting data sets to use in the examples. They will be used in the examples. example.ammonite.svg and log.loop.tex are meant to generate their respective .svg and .tex files.

Details

Litholog drawing data bed.example, boundary.example, example.ammonite, example.ammonite.svg, example.belemnite, example.breccia, example.lense, example.liquefaction, fossil.example, proxy.example, proxy.example.litho

Magnetostratigraphical data chron.example

Litholog exportation script log.loop.tex

Oriented data zeq_example

`tie.lim` *Discretises lim objects*

Description

Discretises continuous lim objects by constant interpolation

Usage

```
tie.lim(lim = NULL, l = NULL, r = NULL, y = NULL, xout = NULL,
  id = 1L, to.lower = T, quiet = F)
```

Arguments

lim	an object convertible into a lim object: either a vector of length 2 or a list of n left (1st element) and n right (2ndt element) interval limits, and of n interval IDs. In this case the lim objects have to be ordered, by ids, dependently to each other, and from left to right. For each id the lim objects heve to cover the entire interval from the lowest to the highest value, without overlap.
l	a vector of n left interval limits
r	a vector of n right interval limits
y	a vector of n values to discretise
xout	a vector of numeric values specifying where interpolation is to take place. It will be identical for each id. If NULL the result will be continuous (points of a continous line).
id	a vector of n interval IDs (default is 1 for each interval)
to.lower	whether to take the left (lower) or right point for interpolation at adjacent points
quiet	whether to warn if the sampling interval is prone to miss the smallest intervals.

See Also

as.lim

Examples

```
l  <- matrix(1:30, ncol = 3, byrow = FALSE)
r  <- matrix(2:31, ncol = 3, byrow = FALSE)
id <- matrix(rep(c("C1", "C2", "C3"),10), ncol = 3, byrow = TRUE)
y  <- matrix(rep(1:10,3), ncol = 3, byrow = FALSE)
xout <- seq(-2,32,0.5)

res  <- tie.lim(l = l, r = r,  y = y, xout = xout, id = id)

cont <- tie.lim(l = l, r = r,  y = y, id = id)

plot(res$x, res$y, pch = 19, col = "red")

lines(cont$x[,1], cont$y[,1])
lines(cont$x[,2], cont$y[,2])
lines(cont$x[,3], cont$y[,3])
```

---

transphere *Convertion between declinaison/inclination/intensity and cartesian coordinates*

Description

Convertion between declinaison/inclination/intensity and cartesian coordinates (modified from RFOC package)

Usage

```
transphere(dec = NA, inc = NA, int = 1, x = NA, y = NA, z = NA,
  into = "other")
```

Arguments

dec	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010). Values outside this range are corrected by incfix().
inc	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010). Values outside this range are corrected by incfix().
int	intensity of the data. Defaults to one (unit sphere).
x, y, z	cartesian coordinates. x is the North, y the East, and z straight down. If dec and inc are not provided they are used to be converted back in dec, inc and int data. Output is corrected by incfix().
into	overriding parameter for generalisation: if "dii" dec, inc and int will remain as they are, and if "xyz" cartesian coordinates will remain as they are

Value

a list of coordinates, in cartesian form or dec, inc, int form following the input

See Also

fmod, dipfix and incfix

Examples

```
transphere(dec = c(65,135), inc = c(32,74))

l <- transphere(dec = c(65,135), inc = c(32,74))
transphere(x = l$x, y = l$y, z = l$z)
```

---

weld *Combines segments with "litholog()"-like data frame*

Description

Adds segments to the polygon forming the bed of a log in a "litholog()"-like data frame.

Usage

```
weld(log, dt, xy, begin, end, erase = "none", order = "current")
```

Arguments

log	a "litholog()"-like data frame on which the new segment needs to be welded.
dt	the dt value for each point of the added segment.
xy	the xy value for each point of the added segment.
begin	the row of log after which the segment will be added.
end	the row of log before which the segment will be added (end should be superior to begin).
erase	erase the begin point ('begin'), end point ('end'), both ('both') or only the points in between ('none').
order	the order of the added points : can be the current order ('current'), the current order inversed ('inverse'), or ordered by xy ('xy' or '-xy') or dt ('dt' or '-dt').

Value

a "litholog()"-like data frame with the bed that comprises the begin and end row having the segment welded to it.

See Also

litholog and weldlog

Examples

```
l <- c(1)
r <- c(2)
h <- c(4)
i <- c("B1")
log <- litholog(l, r, h, i)

seg <- sinpoint(4, 1, 0.25, pos = 2, phase = 0.5)
welded <- weld(log, seg$y, seg$x, 3, 4, order = "inverse", erase = "both")

plot(c(-1,5),c(0,3),type = "n")

multigons(log$i,log$xy,log$dt)
points(seg$x,seg$y)

multigons(welded$i, welded$xy, welded$dt, lty = 2, lwd = 3, border = "red")
```

---

weldlog *Changes boundaries segments in basic lithologs*

Description

Adds personalised segements to bed boundaries of lithologs from "litholog()"-like data frames

Usage

```
weldlog(log, dt, seg, j = 1:length(dt), col.xy = 1, col.dt = 2,
  auto.dt = T, add.dt = 0, omit1 = NULL, omit2 = NULL, warn = T)
```

Arguments

log	a "litholog()"-like data frame on which the new segments need to be welded.
dt	the position of the n boundaries to change.
seg	a list of n dataframes having xy and dt coordinates for the segments that are going to be welded to the log.
j	the indexes of the segments attributed to each boundary or the names of these segments. Should be of same length than dt.
col.xy	the number of the column for the xy coordinates in the seg dataframes.
col.dt	the number of the column for the dt coordinates in the seg dataframes.
auto.dt	whether to automatically add the dt value to the dt of the segments (with the add.dt value when it is not zero)
add.dt	a value to add to the dt of the segments for each boundary (in addition of the value of the dt parameter). Should be of length 1 or of same length than dt.
omit1, omit2	the dt of the boundary for which either the upper or lower bed should not be welded to (1 and 2 depending on the order of the beds in the original log)
warn	whether you want to be annoyed (beginners should find it useful to be annoyed)

Value

a "litholog()"-like data frame, with new bed boundaries

See Also

Complementary functionlitholog

Underlying function: weld

To generate sinuoisidal segments: sinpoint To generate a lot of different sinuoisidal segments: see the example in neatPick

To import and adapt .svg files as segments: pointsvg, framesvg, centresvg and changesvg

Examples

```
l <- c(0,1,2,3,4)
r <- c(1,2,3,4,5)
h   <- c(4,3,4,3,4)
i <- c("B1","B2","B3","B4","B5")
log  <- litholog(l, r, h, i)

whiteSet(xlim = c(-1,5), ylim = c(-1,6))

multigons(log$i, log$xy, log$dt, lty = 3)
```

```
seg1 <- sinpoint(4, 0, 0.25, phase=0.5)
seg2 <- sinpoint(4, 0, 0.25, phase=1.5)

welded <- weldlog(log, dt = c(2,3,4), seg = list(seg1 = seg1, seg2 = seg2),
                  j = c("seg1", "seg2", "seg2"))

multigons(welded$i, welded$xy, welded$dt, lwd = 3, lty = 2, border = "red")
```

---

whiteSet *Sets the plot environment to draw a long data set*

Description

Sets the plot environment to draw a long dataset. It is without background, and with only axes with major and minor ticks.

Usage

```
whiteSet(xlim, ylim, xtick = NA, ytick = NA, nx = 1, ny = 1,
  xaxs = "i", yaxs = "i", xarg = list(tick.ratio = 0.5),
  yarg = list(tick.ratio = 0.5, las = 1), add = FALSE)
```

Arguments

xlim, ylim	the x and y limits (e.g. xlim = c(-1,1))
xtick, ytick	the interval between each major ticks for x and y
nx, ny	the number of intervals between major ticks to be divided by minor ticks in the x and y axes
xaxs, yaxs	The style of axis interval calculation to be used for the x and y axes. By default it is "i" (internal): it just finds an axis with pretty labels that fits within the original data range. You can also set it to "r" (regular): it first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range. See ?par for further explanation
xarg, yarg	a list of arguments to feed to minorAxis() for the x and y axes. See the ?minorAxis help page for the possible arguments. See ?merge_list for further information.
add	whether to add to an existing plot

See Also

Similar functions: greySet and blackSet

To create axes with major and minor ticks: minorAxis

To print a plot in pdf: pdfDisplay

To automatically determine pretty interval limits: encase

Examples

```
y <- c(0,11,19,33)
x <- c(1,2,2.5,4)

a <- min(y)
b <- max(y)

f <- encase(a-1,b,5)

whiteSet(c(0,4), f, ytick = 5, ny = 5, xaxs = "r")

points(x, y, pch=19)
```

`ylink` *Draws connection lines to connect two points in y*

Description

Draws connection lines to connect two points in y

Usage

```
ylink(y1, y2, x1, x2, ratio = 0.1, xi1 = NA, xi2 = NA, l = list(lty
  = 3))
```

Arguments

`y1, y2`	y positions (you can provide several ones at once)
`x1, x2`	x positions (you can provide several ones at once)
`ratio`	the ratio of the breaking points of the lines (from the start or end to the centre)
`xi1, xi2`	x positions of the breaking points of the lines.
`l`	a list of arguments to feed lines(). Go see ?lines to know which arguments can be provided. See ?merge.list for further information.

See Also

`multilines`, `bedtext`, `infobar` and `nlegend`

Examples

```
plot(c(0,6),c(-20,20), type = "n")

infobar(ymin = c(-20,0), ymax = c(0,20), xmin = 1, xmax = 0,
        m = list(col = c("black", "white")))

infobar(ymin = c(-20,10), ymax = c(10,20), xmin = 5, xmax = 6,
        m = list(col = c("black", "white")))
```

```
ylink(c(0,12),c(10,20), x1 = 1, x2 = 5, ratio = 0.2,
      l = list(lty = c(1,3), lwd = 2))
```

`zijderveld` *Draws a Zijderveld plot*

Description

Draws a Zijderveld plot: it projects 3D points (having declination, inclination and intensity) in 2D, horizontally and vertically.

Usage

```
zijderveld(dec, inc, int, xh = "WE", xv = xh, centre = F,
  xlim = NA, ylim = NA, unit = NA, xlab = "", ylab = "",
  labels = NA, nlabels = 1, h = list(pch = 19), v = list(pch = 21,
  bg = "white"), f = list(pch = 21, bg = "white", cex = 1.5),
  t = list(pos = 3, offset = 0.5), l = list(), anchored = T,
  style = "branches", tcl = 0.2, orientation = TRUE,
  scientific = NA, decimals = 10, add = FALSE)
```

Arguments

`dec`	declination of the data; it is the angle from the north taken on an horizontal plane. It is measured clockwise from North and ranges from 0 to 360° (Tauxe 2010). Values outside this range are corrected by incfix().
`inc`	inclination of the data; it is the angle from the horizontal, is positive downward, and ranges from +90° for straight down to -90° for straight up (Tauxe, 2010). Values outside this range are corrected by incfix().
`int`	intensity of the data.
`xh`	orientation of the x axis for the horizontal points: can be 'SN' or 'WE'.
`xv`	orientation of the x axis for the horizontal points: can be 'SN', 'WE' or 'modified' (for the latter the horizontal projection of the vector given by the square root of the addition of the squared horizontal components).
`centre`	logical, whether the [0,0] point should be in the centre of the plot. Is ignored if xlim and/or ylim are defined.
`xlim, ylim`	the x and y minimal limits. The actual limits can change to keep a x/y ratio of 1.
`unit`	the tick interval.
`xlab, ylab`	the titles for the axes.
`labels`	a character vector of labels to add to each point.
`nlabels`	the number of labels to skip (for clarity).

h, v, f, t, l	list of graphical parameters to feed the graphical functions: h, v and f are fed to points() for the horizontal, vertical and first points respectively; t is fed to the text() for the labels and l is fed to lines() for the lines joining each horizontal and vertical points. See ?points, ?text and ?lines help page for the possible arguments. See ?merge_list for further information.
anchored	logical, whether the lines should be anchored to the [0,0] point.
style	the style of the plot: 'branches', 'box0', 'box1', or 'box2'. The boxes are advised when zooming using xlim and/or ylim.
tcl	The length of tick marks (see par() help page).
orientation	logical, whether to add captions indicating the orientation of the plot.
scientific	logical or NA, whether have scientific notation (e.g. -1.0E-06) or not (e.g. 0.00015). If NA, R will be left only judge.
decimals	the number of decimals if scientific is T or F. Having not enough decimals can lead to override the unit parameter, but the tick labels will be correctly aligned.
add	logical, whether to add the plot to an existing plot.

Details

By default horizontal projection is made of black points, vertical of white points.

References

- Tauxe, L., 2010. Essentials of Paleomagnetism. University of California Press.

See Also

earnet

Examples

```
zd <- zeq_example

ori <- par()$mfrow

par(mfrow = c(1,2))

zijderveld(dec = zd$Dec, inc = zd$Inc, int = zd$Int,
           xh = "WE", unit = 10^-5)

zijderveld(dec = zd$Dec, inc = zd$Inc, int = zd$Int,
           style = "box1", scientific = FALSE, decimals = 5,
           labels = zd$Treat, nlabels = 2)

par(mfrow = ori)
```

Index