tile-internal(tile)
Internal Functions in the Tile Package

Description

Internal functions in the tile package.

Author(s)

Christopher Adolph <cadolph@u.washington.edu>
Plot multiple graphic styles in a tiled layout, with automated customizable titles and axes.
Help pages for package ‘tile’ version 0.2
cbindfill    Combine uneven vectors or matrices
<table>
<thead>
<tr>
<th>lineplot</th>
<th>Summarize inferences using lineplots</th>
</tr>
</thead>
<tbody>
<tr>
<td>linesTile</td>
<td>Add a connected line segment to a tile plot</td>
</tr>
</tbody>
</table>
nightplot

function to do ...
-- P --

pointsTile  ~~function to do ... ~~,~~
polygonTile  ~~function to do ... ~~,~~
polylinesTile  ~~function to do ... ~~,~~
<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbindfill</td>
<td>Combine uneven vectors or matrices</td>
</tr>
<tr>
<td>riceplot</td>
<td><del>function to do ...</del></td>
</tr>
<tr>
<td>ropeladder</td>
<td><del>function to do ...</del></td>
</tr>
</tbody>
</table>
scatter

~~function to do ...~~
-- T --

<table>
<thead>
<tr>
<th>textTile</th>
<th>~~~function to do ... ~~~</th>
</tr>
</thead>
<tbody>
<tr>
<td>tile</td>
<td>Plot a tiled graphical layout</td>
</tr>
<tr>
<td>tileDraw</td>
<td>(Re-)draw a tile graphical object</td>
</tr>
</tbody>
</table>
cbindfill(tile)
Combine uneven vectors or matrices

Description

Take a sequence of vector or matrix arguments and combine by columns or rows, respectively, regardless of their length

Usage

cbindfill(...)
rbindfill(...)

Arguments

... vectors or matrices to be combined

Details

cbindfill and rbindfill combine a set of vectors of matrices into a single matrix by columns or rows, respectively. Unlike cbind or rbind, the inputs need not have the same number of rows or columns. Instead, the output matrix of cbindfill (rbindfill) has the number of rows (columns) as the longest input. Shorter inputs are filled out with NAs, not recycled.

Value

A matrix combining the ... arguments column-wise or row-wise.

Author(s)

Christopher Adolph <cadolph@u.washington.edu>

See Also

cbind, rbind

Examples

cbindfill(1:5, 1:8, 1:7)
rbindfill(1:5, 1:8, 1:7)
[Package tile version 0.2 Index]
lineplot(tile)
Summarize inferences using lineplots

Description

Initializes a line graphic with several optional features aimed at summarizing inferences from regression models: e.g., confidence intervals, perhaps created from simulations; clipping the plot to the convex hull to avoid unwarranted extrapolation; simple linear or robust fits to the data. If you simply want to draw a line on a tile plot, use linesTile instead.

This function does no plotting; it only creates a lineplot object to be drawn on one or more plots in a tiled arrangement of plots. To complete the drawing include the object as an input to tile. From tile, it is possible to set further options including plot and axis titles, axis ranges and labels, logged axes, and annotations to the plot.

Usage

lineplot(…)

Arguments

Any number of arguments given below. Must include exactly one horizontal dimension (x or top) and exactly one vertical dimension (y or right). All inputs should be identified by appropriate tags; i.e., use lineplot(x=myxvar, y=myyvar), not lineplot(myxvar,myyvar)

Details

Lineplots offer a plethora of data processing and formatting options. Confidence intervals (shown as shaded regions or dashed lines) can be calculated from simulations or posterior draws, or may be provided by the user. Alternatively, lineplot can add simple fit lines and CIs to the plotted data (e.g., a linear, robust, or loess fit). Optionally, only results inside the convex hull of the original data be drawn; alternatively, extrapolations outside this range can be flagged.

The graphical parameters for each element of the lineplot (including lines, shaded or dashed confidence intervals, and symbols or text marking points on the line) can be adjusted, often on a point-by-point basis.
Used in conjunction with \texttt{tile}, \texttt{lineplot} is a high-level plotting function, meaning that this combination will output finished plots. These plots cannot be easily modified after creation; rather, users can specify a wealth of graphics options within the initial calls to \texttt{lineplot} and \texttt{tile}, as documented below. For example, the graphical parameters for each element of the lineplot (including lines, shaded or dashed confidence intervals, and symbols or text marking points on the line) can be adjusted, often on a point-by-point basis. Users should carefully study the examples below and in \texttt{tile} before use.

A call to \texttt{lineplot} *must* provide an orthogonal pair of the following inputs:

- \texttt{x}
  - co-ordinate vector of data to plot, attached to the x (bottom) axis. \( x \) may be plotted directly, or treated as simulation data to summarize (see parameter \texttt{simulates} below)
- \texttt{y}
  - co-ordinate vector of data to plot, attached to the y (left) axis; may be simulation data
- \texttt{top}
  - co-ordinate vector of data to plot, attached to the top axis; may be simulation data
- \texttt{right}
  - co-ordinate vector of data to plot, attached to the right axis; may be simulation data

Users will often wish to provide the following inputs:

- \texttt{simulates}
  - string, identifies one of the variables (\texttt{x}, \texttt{y}, \texttt{top}, or \texttt{right}) as simulation data (default = \texttt{NULL}, for no simulation data). If \texttt{simulates} is set, the other dimension of data will be treated as a matching list of scenarios over which to summarize the simulates. For example, to plot summaries of 1,000 simulates drawn from the conditional distribution of a response for each of 10 different values of a particular covariate, stack all 10,000 simulates in a single vector (e.g., \texttt{right}), then create a corresponding 10,000-vector listing the respective values of (say) \( x \) for each of these simulates. \texttt{lineplot} will then calculate confidence intervals each scenario, as requested in \texttt{ci} below
- \texttt{lower}
  - vector, lower bounds; used when
- \texttt{upper}
plot
  scalar or vector, the plot(s) in which this trace will be drawn; defaulting to the first plot. Plots are numbered consecutively from the top left, row-by-row; e.g., in a 2x3 tiling, the first plot in the second row is plot number 4.

The next set of inputs are all optional, and control the major features of lineplot. It is usually best to use either `ci` or `fit`, but not both.

`ci`
  list, parameters governing the appearance and calculation of confidence intervals from data in lower and upper or provided by the simulations defined in `simulates`:
    levels
    mark

`fit`
  list, parameters governing the appearance and calculation of simple fits to the two plotted dimensions:
    method
    The type of fit to apply: `linear` (default) fits a bivariate linear regression; `wls` fits a weighted linear regression; `robust` fits a XXX; `loess` fits a loess smoother
    ci
    mark
    col

`extrapolate`
  list, parameters governing the plotting of extrapolation outside the convex hull of the covariate data, using `whatif` in the package `WhatIf`:
    formula
    optional formula object, used to specify the estimated model; useful if the model contains interactions or functions of the covariates given in `data` below
    data
matrix or dataframe, the actual values of all covariates used to estimate the model (omit the constant and response variable)

cfact
matrix or dataframe, the counterfactual values of all the covariates (omit the constant and response variable), one row for each scenario. The order of columns must match data, and the order of rows must match the order of the scenarios. If scenarios are calculated from simulates, then the rows must be listed from the scenario with the smallest factor level to the highest

omit.extrapolated
If TRUE (default), then the plotted trace and CIs are clipped to the convex hull; if FALSE, then extrapolation outside the convex hull is printed in a lighter color or with dashed/dotted lines

The following graphics parameters mimic the inputs to grid::gpar. Starred (*) parameters accept vector as well as scalar input, so that different parameter values can be applied to different elements of the plot:

fill
  *Color for filling polygons; default is transparent
col
  *Color for lines, symbols, and text; default is black
lty
  *Line type; default is solid
lwd
  *Line width; default is 1
cex
  *Multiplier applied to fontsize; default is 1
lex
  *1
fontsize
  Size of the text in points; default is 12
lineheight
  Height of a line as a multiple of the size of text; default is 1.2
font
  Font face (alias for fontface); default is 1
fontfamily
  The font family; default is blank!
fontface
The font face (\code{bold}, \code{italic}, etc.); default is \code{plain}

alpha
    Alpha channel for transparency; default is 0.8

lineend
    "round"

linejoin
    "round"

linemitre
    10

These final graphical parameters are specific to tile-based plots, and are available for advanced users to tweak the appearance of traces. Starred (*) parameters accept vector as well as scalar input, so that different parameter values can be applied to different elements of the plot:

    markers
        *
    labels
        *
    labelxoffset
        *
    labelyoffset
        *
    size
        *1
    pch
        *1
    addArrow
        30
    lengthArrow
        unit(0.25, "inches")
    endsArrow
        XXX
    typeArrow
        XXX
    just
        XXX
    hjust
        XXX
    vjust
        XXX
    rot
        XXX
Graphical elements with lower layer will be drawn later in the plotting process, and hence appear on top of elements with higher layer. By default, layer is 10, though polygons in this trace will be plotted at layer + x, and text labels and markers at layer - x.

Value

A lineplot object, used only as an input to tile

Author(s)
Christopher Adolph <cadolph@u.washington.edu>

See Also

tile, linesTile

Examples

# Example 1: Linear regression on Swiss fertility;
# Tiled lineplots of counterfactual scenarios calculated by
# predict() and clipped to convex hull
data(swiss)

# Estimate model
lm.result <- lm(Fertility ~ ., data = swiss)

# Create counterfactual scenarios
cfactbaseline <- apply(swiss[,2:6],2,mean)
cfact1 <- cfact2 <- cfact3 <- cfact4 <- cfact5 <-
data.frame(matrix(cfactbaseline,nrow=101,ncol=5,byrow=TRUE,
dimnames=list(NULL,names(cfactbaseline))))
seq(0,100)

lm.pred1 <- predict(lm.result,newdata=cfact1,interval="confidence",1}
lm.pred2 <- predict(lm.result, newdata=cfact2, interval="confidence", level=0.95)
lm.pred3 <- predict(lm.result, newdata=cfact3, interval="confidence", level=0.95)
lm.pred4 <- predict(lm.result, newdata=cfact4, interval="confidence", level=0.95)
lm.pred5 <- predict(lm.result, newdata=cfact5, interval="confidence", level=0.95)

# Create traces of each set of counterfactuals
trace1 <- lineplot(x=cfact1[,1],
    y=lm.pred1[,1],
    lower=lm.pred1[,2],
    upper=lm.pred1[,3],
    ci=list(mark="dashed"),
    extrapolate=list(data=swiss[,2:6], cfact=cfact1,
                     omit.extrapolated=TRUE),
    col="blue",
    plot=1)

trace2 <- lineplot(x=cfact2[,2],
    y=lm.pred2[,1],
    lower=lm.pred2[,2],
    upper=lm.pred2[,3],
    ci=list(mark="dashed"),
    extrapolate=list(data=swiss[,2:6], cfact=cfact2,
                     omit.extrapolated=TRUE),
    col="red",
    plot=2)

trace3 <- lineplot(x=cfact3[,3],
    y=lm.pred3[,1],
    lower=lm.pred3[,2],
    upper=lm.pred3[,3],
    ci=list(mark="dashed"),
    extrapolate=list(data=swiss[,2:6], cfact=cfact3,
                     omit.extrapolated=TRUE),
    col="green",
    plot=3)

trace4 <- lineplot(x=cfact4[,4],
    y=lm.pred4[,1],
    lower=lm.pred4[,2],
    upper=lm.pred4[,3],
    ci=list(mark="dashed"),
    extrapolate=list(data=swiss[,2:6], cfact=cfact4,
                     omit.extrapolated=TRUE),
    col="brown",
    plot=4)
trace5 <- lineplot(x=cfact5[,5],
y=lm.pred5[,1],
lower=lm.pred5[,2],
upper=lm.pred5[,3],
ci=list(mark="dashed"),
extrapolate=list(data=swiss[,2:6],cfact=cfact5,
                omit.extrapolated=TRUE),
    col="violet",
    plot=5)

at.x <- c(0,20,40,60,80,100)
at.y <- c(50,60,70,80,90,100)

# Plot traces using tile
tc <- tile(trace1,
            trace2,
            trace3,
            trace4,
            trace5,
            RxC=c(2,3),
            limits=c(0,100,50,100),
            output=list(wide=7.5,outfile="lineplotExample1",type="pdf"),
            xaxis=list(at=at.x),
            yaxis=list(at=at.y),
            xaxistitle=list(type="all",labels=c(names(cfactbaseline)),
                yaxistitle=list(type="all",labels=c("Fertility")),
                plottitle=list(type="all",labels=c("Test")),
                maintitle=list(labels=c("Linear Model of Fertility")),
                gridlines=list(type="xy"),
                frame=TRUE
            )

# Example 2.1: Multinomial Logistic Regression of alligator food;
# Tiled lineplots using *manually simulated counterfactuals*, with
# extrapolation outside the convex hull flagged
#
# See Ex. 2.2 for an automated way to handle simulations

data(gator)
require(MASS)
require(nnet)

# Estimate MNL using the nnet library
mlogit.result <- multinom(food ~ size + female, Hess=TRUE)
pe <- mlogit.result$wts[c(6,7,8,10,11,12)]
    # point estimates
vc <- solve(mlogit.result$Hess)  # var-cov matrix

# Simulate counterfactual results, varying size and sex
sims <- 10000
simbetas <- mvrnorm(sims,pe,vc)  # draw parameters, using MASS:
sizerange <- seq(1,4,by=0.1)  # range of counterfactual size
femalerange <- c(0,1)  # range of counterfactual sexe
simycat1 <- simycat2 <- simycat3 <- cfactsize <- cfactfemale <- NULL
for (isex in 1:length(femalerange)) {  # loop over sex scenarios
  for (isize in 1:length(sizerange)) {  # loop over size scenarios
    # Set up a hypothetical X vector for this scenario
    hypx <- rbond(1, sizerange[isize], femalerange[isex])
    
    # Calculate simulated MNL denominators for this scenario
    simdenom <- (1 + exp(simbetas[,1:3] %*% hypx) + exp(simbetas[, 4:6] %*% hypx))
    
    # Add simulated probabilities for each category to storage v
    simycat1 <- c(simycat1, 1/simdenom)
    simycat2 <- c(simycat2, exp(simbetas[,1:3] %*% hypx)/simdenom)
    simycat3 <- c(simycat3, exp(simbetas[,4:6] %*% hypx)/simdenom)
    
    # Save hypothetical X's to storage vectors:
    # must match simulated probabilities element for element
    cfactsize <- c(cfactsize, rep(sizerange[isize], sims))
    cfactfemale <- c(cfactfemale, rep(femalerange[isex], sims))
  }
}

# Create one trace for each predicted category of the response, and
trace1 <- lineplot(x=cfactsize[cfactfemale==0],
  y=simycat1[cfactfemale==0],
  simulates="y",
  ci=list(mark="shaded", levels=0.67),
  extrapolate=list(data=cbind(size,female),
                   cfact=cbind(sizerange,rep(0,length(sizerange)),
                               omit.extrapolated=FALSE),
                   col="blue",
                   plot=1)
trace2 <- lineplot(x=cfactsize[cfactfemale==0],
  y=simycat2[cfactfemale==0],
  simulates="y",
  ci=list(mark="shaded", levels=0.67),
  extrapolate=list(data=cbind(size,female),
                   cfact=cbind(sizerange,rep(0,length(sizerange)),
                               omit.extrapolated=FALSE),
                   col="red",
                   plot=1)
trace3 <- lineplot(x=cfactsize[cfactfemale==0],
  y=simycat3[cfactfemale==0],
  simulates="y",
  ci=list(mark="shaded", levels=0.67),
  extrapolate=list(data=cbind(size,female),
                   cfact=cbind(sizerange,rep(0,length(sizerange)),
                               omit.extrapolated=FALSE),
                   col="red",
                   plot=1)
y = simycat3[cfactfemale == 0],
simulates = "y",
ci = list(mark = "shaded", levels = 0.67),
extrapolate = list(data = cbind(size, female),
                   cfact = cbind(sizerange, rep(0, leng
                   omit.extrapolated = FALSE),
                   col = "green",
                   plot = 1
               )

trace4 <- lineplot(x = cfactsize[cfactfemale == 1],
y = simycat1[cfactfemale == 1],
simulates = "y",
ci = list(mark = "shaded", levels = 0.67),
extrapolate = list(data = cbind(size, female),
                   cfact = cbind(sizerange, rep(1, leng
                   omit.extrapolated = FALSE),
                   col = "blue",
                   plot = 2
               )

trace5 <- lineplot(x = cfactsize[cfactfemale == 1],
y = simycat2[cfactfemale == 1],
simulates = "y",
ci = list(mark = "shaded", levels = 0.67),
extrapolate = list(data = cbind(size, female),
                   cfact = cbind(sizerange, rep(1, leng
                   omit.extrapolated = FALSE),
                   col = "red",
                   plot = 2
               )

trace6 <- lineplot(x = cfactsize[cfactfemale == 1],
y = simycat3[cfactfemale == 1],
simulates = "y",
ci = list(mark = "shaded", levels = 0.67),
extrapolate = list(data = cbind(size, female),
                   cfact = cbind(sizerange, rep(1, leng
                   omit.extrapolated = FALSE),
                   col = "green",
                   plot = 2
               )

at.x <- c(1, 2, 3, 4)
at.y <- c(0, 0.2, 0.4, 0.6, 0.8, 1)

# Plot traces using tile
tc <- tile(trace1,
           trace2,
           trace3,
           trace4,
           trace5,
           trace6)
trace4,
trace5,
trace6,
RxC = c(1,2),
limits = c(1,4,0,1),
output = list(wide=6.5,outfile="lineplotExample2",type="p
xaxis = list(at=at.x),
yaxis = list(at=at.y),
xaxistitle = list(type="all",labels=c("Size of alligator" yaxistitle = list(type="first",labels="Pr(Food preference undertitle = list(labels=c("Male","Female")), maintitle = list(labels="Food choice by alligator size"), gridlines = list(type="xy"), frame=TRUE
)

# Example 2.2: Multinomial Logistic Regression of alligator food;
# Tiled lineplots using 'preprocessed simulations', with extrapola
# (Alternative method for constructing Ex 2.1; output is identical)

data(gator)
require(MASS)
require(nnet)
require(simcf)

# Estimate MNL using the nnet library
mlogit.result <- multinom(food ~ size + female, Hess=TRUE)
pe <- mlogit.result$wts[c(6,7,8,10,11,12)]
    # point estimates
vc <- solve(mlogit.result$Hess)       # var-cov matrix

# Alternative code for simulations above which calculates CIs to pas # to lineplot & tile (also could try Zelig)
sims <- 10000
simbetas <- mvrnorm(sims,pe,vc)       # draw parameters, using MASS:
simb <- array(NA, dim = c(sims,3,2))  # re-arrange simulates to arr
simb[,1] <- simbetas[,1:3]           # for MNL simulation
simb[,2] <- simbetas[,4:6]           # range of counterfactual size
sizerange <- seq(1,4,by=0.1)        # range of counterfactual sexe
femalerange <- c(0,1)

# Create full factorial set of counterfactuals
xhyp <- setfactorial(size = sizerange, female = femalerange)

# Simulate expected probabilities
mlogit.qoi1 <- mlogitsimev(xhyp,simb,ci=0.67)
# Create one trace for each predicted category of the response, and
trace1 <- lineplot(x=xhyp\$size[xhyp\$female==0],
    y=mlogit.qoi1$pe[xhyp\$female==0,1],
    lower=mlogit.qoi1$lower[xhyp\$female==0,,1],
    upper=mlogit.qoi1$upper[xhyp\$female==0,,1],
    ci=list(mark="shaded"),
    extrapolate=list(data=cbind(size,female),
        cfact=xhyp[xhyp\$female==0,],
        omit.extrapolated=FALSE),
    col="blue",
    plot=1)

trace2 <- lineplot(x=xhyp\$size[xhyp\$female==0],
    y=mlogit.qoi1$pe[xhyp\$female==0,2],
    lower=mlogit.qoi1$lower[xhyp\$female==0,,2],
    upper=mlogit.qoi1$upper[xhyp\$female==0,,2],
    ci=list(mark="shaded"),
    extrapolate=list(data=cbind(size,female),
        cfact=xhyp[xhyp\$female==0,],
        omit.extrapolated=FALSE),
    col="red",
    plot=1)

trace3 <- lineplot(x=xhyp\$size[xhyp\$female==0],
    y=mlogit.qoi1$pe[xhyp\$female==0,3],
    lower=mlogit.qoi1$lower[xhyp\$female==0,,3],
    upper=mlogit.qoi1$upper[xhyp\$female==0,,3],
    ci=list(mark="shaded"),
    extrapolate=list(data=cbind(size,female),
        cfact=xhyp[xhyp\$female==0,],
        omit.extrapolated=FALSE),
    col="green",
    plot=1)

trace4 <- lineplot(x=xhyp\$size[xhyp\$female==1],
    y=mlogit.qoi1$pe[xhyp\$female==1,1],
    lower=mlogit.qoi1$lower[xhyp\$female==1,,1],
    upper=mlogit.qoi1$upper[xhyp\$female==1,,1],
    ci=list(mark="shaded"),
    extrapolate=list(data=cbind(size,female),
        cfact=xhyp[xhyp\$female==1,],
        omit.extrapolated=FALSE),
    col="blue",
    plot=2)

trace5 <- lineplot(x=xhyp\$size[xhyp\$female==1],
```r
y = mlogit.qoi1$pe[xhyp$female==1, 2],
lower = mlogit.qoi1$lower[xhyp$female==1, 2],
upper = mlogit.qoi1$upper[xhyp$female==1, 2],
ci = list(mark = "shaded"),
extrapolate = list(data = cbind(size, female),
cfact = xhyp[xhyp$female==1, ],
omit.extrapolated = FALSE),

col = "red",
plot = 2
)

trace6 <- lineplot(x = xhyp$size[xhyp$female==1],
y = mlogit.qoi1$pe[xhyp$female==1, 3],
lower = mlogit.qoi1$lower[xhyp$female==1, 3],
upper = mlogit.qoi1$upper[xhyp$female==1, 3],
ci = list(mark = "shaded"),
extrapolate = list(data = cbind(size, female),
cfact = xhyp[xhyp$female==1, ],
omit.extrapolated = FALSE),

col = "green",
plot = 2
)

at.x <- c(1, 2, 3, 4)
at.y <- c(0, 0.2, 0.4, 0.6, 0.8, 1)

# Plot traces using tile
tc <- tile(trace1, trace2, trace3, trace4, trace5, trace6,
RxC = c(1, 2),
limits = c(1, 4, 0, 1),
output = list(wide = 6.5, outfile = "lineplotExample2", type = "pdf"),
xaxis = list(at = at.x),
yaxis = list(at = at.y),
xxaxistitle = list(type = "all", labels = c("Size of alligator"))
yyaxistitle = list(type = "first", labels = c("Pr(Food preference"))
undertitle = list(labels = c("Male", "Female")),
maintitle = list(labels = "Food choice by alligator size"),
gridlines = list(type = "xy"),
frame = TRUE
)
```

---

[Package tile version 0.2 Index]
Add a connected line segment to a tile plot

Description

Initializes a simple line graphic. For more advanced features suitable for summarizing inference from a model, use instead lineplot.

This function does no plotting; it only creates a linesTile object to be drawn on one or more plots in a tiled arrangement of plots. To complete the drawing include the object as an input to tile. From tile, it is possible to set further options including plot and axis titles, axis ranges and labels, logged axes, and annotations to the plot.

Usage

linesTile(...)

Arguments

... Any number of arguments given below. Must include exactly one horizontal dimension (x or top) and exactly one vertical dimension (y or right). All inputs should be identified by appropriate tags; i.e., use linesTile(x=myxvar, y=myyvar), not linesTile(myxvar, myyvar)

Details

linesTile simply creates a line or series of connected line segments to add to a tile plot. It serves as the tile equivalent to the base graphics lines or grid graphics linesGrob, and is primarily useful for annotating plots which also use other traces. If you need to draw a large number of disconnected line segments, creating a single trace made by polylinesTile will be much faster than creating a large number of linesTile traces.

The parameters available for constructing a linesTile trace are similar to other trace functions for tile.

trace functions, then layout function

Value
A linesTile object, used only as an input to tile

Author(s)

Christopher Adolph <cadolph@u.washington.edu>

See Also

tile, polylinesTile, lineplot

[Package tile version 0.2 Index]
nightplot(tile)
**function to do ... **

**Description**

~~ A concise (1-5 lines) description of what the function does. ~~

**Usage**

`nightplot(...)`

**Arguments**

...~~Describe ... here~~

**Details**

~~ If necessary, more details than the description above ~~

**Value**

~Describe the value returned If it is a LIST, use

```
comp1 Description of 'comp1'
comp2 Description of 'comp2'
```

...

**Warning**

....

**Note**

~~further notes~~

~Make other sections like Warning with section{Warning }{....} ~

**Author(s)**

~~who you are~~
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

###---- Should be DIRECTLY executable !! ----
###-- ==> Define data, use random,
###-- or do help(data=index) for the standard data sets.

## The function is currently defined as
function(...){
  args <- list(...,graphic="nightplot")
  args
}

[Package tile version 0.2 Index]
pointsTile(tile)
~~function to do ... ~~

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

pointsTile(...)

Arguments

... ~~Describe ... here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

~~further notes~~

~Make other sections like Warning with section{Warning }{....} ~

Author(s)

~~who you are~~
## Examples

```r
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random, 
##-- or do help(data=index) for the standard data sets.

## The function is currently defined as
function(...){
  args <- list(...,graphic="points")
  args
}
```

[Package tile version 0.2 Index]
polygonTile(tile)
~~function to do ... ~~

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

polygonTile(...)

Arguments

... ~~Describe ... here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

~~further notes~~

~Make other sections like Warning with section{Warning }{....} ~

Author(s)

~~who you are~~
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

```r
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random,
##-- or do help(data=index) for the standard data sets.

## The function is currently defined as
function(...){
  args <- list(...,graphic="polygon")
  args
}
```

[Package tile version 0.2 Index]
polylinesTile(tile)
~~function to do ... ~~

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

polylinesTile(...)

Arguments

... ~~Describe ... here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

~~further notes~~

~Make other sections like Warning with section{Warning }{....} ~

Author(s)

~~who you are~~
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

###---- Should be DIRECTLY executable !! ----
###-- ==>  Define data, use random,
###--       or do help(data=index) for the standard data sets.

##
## The function is currently defined as
##
function(...){
  args <- list(...,graphic="polylines")
  args
}

[Package tile version 0.2 Index]
riceplot(tile)

R Documentation
function to do ...

Description

A concise (1-5 lines) description of what the function does.

Usage

riceplot(...)

Arguments

Describe ... here

Details

If necessary, more details than the description above

Value

Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

further notes

Make other sections like Warning with section{Warning }{....} ~

Author(s)

who you are
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

###---- Should be DIRECTLY executable !! ----
###-- ==> Define data, use random,
###-- or do help(data=index) for the standard data sets.

[Package tile version 0.2 Index]
ropeladder(tile)
function to do ...

Description

A concise (1-5 lines) description of what the function does.

Usage

ropeladder(...)

Arguments

... Describe ... here...

Details

If necessary, more details than the description above

Value

Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'
...

Warning

....

Note

further notes

Make other sections like Warning with section{Warning} {...} ~

Author(s)

who you are
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~
scatter(tile)
~~function to do ... ~~

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

scatter(...)  

Arguments

...~~Describe ... here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

~~further notes~~

~Make other sections like Warning with section{Warning }{....} ~

Author(s)

~~who you are~~
References

~put references to the literature/web site here ~

See Also

~~objects to See Also as help, ~~~

Examples

###---- Should be DIRECTLY executable !! ----
###-- ==> Define data, use random,
###-- or do help(data=index) for the standard data sets.

## The function is currently defined as
function(...){
  args <- list(..., graphic="scatter")
  args
}

[Package tile version 0.2 Index]
Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage
textTile(...)

Arguments

... ~~ Describe ... here ~~

Details

~~ If necessary, more details than the description above ~~

Value

~ Describe the value returned If it is a LIST, use

comp1 Description of 'comp1'
comp2 Description of 'comp2'

...

Warning

....

Note

~~ further notes ~~

~ Make other sections like Warning with section{Warning }{....} ~

Author(s)

~~ who you are ~~
References

~put references to the literature/web site here ~

See Also

~~~objects to See Also as help, ~~~

Examples

###---- Should be DIRECTLY executable !! ----
###-- ==> Define data, use random,
###-- or do help(data=index) for the standard data sets.

### The function is currently defined as
function(...){
  args <- list(...,graphic="text")
  args
}

[Package tile version 0.2 Index]
Plot a tiled graphical layout

Description

A generic layout function for plotting tiled arrangements of graphic styles found in the tile package, including scatterplots, lineplots, dotplots and more specialized formats. Includes fine control of titles, axes, axis transformation, and rugs for all plots.

Usage

```r
tile(..., RxC = NULL, output = list(), limits = NULL, 
  xaxis = list(), yaxis = list(), topaxis = list(), rightaxis = list() 
  xaxistitle = list(), yaxistitle = list(), topaxistitle = list(), rightaxistitle = list(), 
  plottitle = list(), maintitle = list(), undertitle = list(), rowtitle = list(), 
  gridlines = list(), frame = FALSE, leeway = 0, width = list(), height = list(), 
  defaults = list(), layoutonly = FALSE, verbose = FALSE)
```

Arguments

- **...** Any number of traces, as supplied by tile graphic functions such as `lineplot`, `scatter`, `ropeladder`, `nightplot`, `riceplot`, `linesTile`, `pointsTile`, `textTile`, `polylinesTile`, or `polygonTile`

- **RxC** A 2-vector giving the number of rows and columns of the tiling of plots. Default is to place all plots in a single row

- **output** A list controlling the output device:
  - `outfile` required: Name of the file where the graphic is to be saved; an appropriate extension will be applied automatically
  - `wide` required: Width of the output device, in inches
  - `pointsize` Default pointsize for the device; default is 12
  - `family` Font family for the device; default is Helvetica
  - `type` Device type; either `pdf` (default), `postscript`, or
A vector (or matrix) of limits to the plotting region, similar to `usr` for base graphics, but generalized to the four axis set-up of `tile`. The usual case is an 8-vector, \((xmin, xmax, ymin, ymax, topmin, topmax, rightmin, rightmax)\). If no plots have right axes, this could be a 6-vector; if none have right or top axes, it could be a 4-vector, and so on. If an element of `limits` is `NA`, then that limit is automatically computed based on the data plotted. If no limits are provided, all limits are computed. To set different limits for each plot, provide a matrix with one row of limits for each plot.

A list; controls the plotting and format of the x-axes. WHEN IS THIS DRAWN?:

- **at**
  - Vector or matrix of locations to draw ticks (if a matrix, one row for each plot; NAs are ignored). Default is to have `tile` select at automatically

- **labels**
  - Vector or matrix of labels to apply at tick locations (if a matrix, one row for each plot). Default is to use at

- **tick.length**
  - Length of ticks; default is `-0.5`

- **label.loc**
  - Location of tick labels relative to axis baseline; default is `1.25`

- **ticks**
  - Show tick marks; default is `TRUE`

- **ntics**
  - If ticks are automatic, how many to draw? Default is `5`

- **major**
  - Plot the main axis line? Default is `TRUE`

- **col**
  - Color for all axis elements; default is `black`

- **lwd**
  - Line width of axis elements; default is `0.5`
xaxis

  cex 1
  fontsize cfs
  add

  Should this axis be draw? May be a vector, with values for each plot. Default is for tile to automatically determine whether to draw this axis

log

  Logical or numeric; should this axis be logged? FALSE (default), TRUE, or a positive number (to choose a scale other than the natural log)

rug

  A list; adds and formats a rug (1-d plot) for this axis:
    data
      NULL
    col
      ???
    lwd
      0.25
    pch
      0.5
    type
      Character string; the type of rug to draw:
      lines for thin lines marking each datapoint (default); density for a smoothed histogram; dots for a histogram made of dots for each datapoint; jitter for a strip filled with jittered dots for each datapoint
      kernel
      gaussian

yaxis

  A list; controls the plotting and format of the y-axes; see xaxis for parameters

topaxis

  A list; controls the plotting and format of the top-axes; see xaxis for parameters

rightaxis

  A list; controls the plotting and format of the right-axes; see
rightaxis xaxis for parameters

xaxistitle A list; places titles for the x-axes and controls their format:

yaxistitle A list; places titles for the y-axes and controls their format; see xaxistitle for parameters

topaxistitle A list; places titles for the top-axes and controls their format; see xaxistitle for parameters

rightaxistitle A list; places titles for the right-axes and controls their format; see xaxistitle for parameters

A list; places titles above each plot and controls their format. All parameters may be scalar or vector (if different results are desired for each title):

labels
   NULL
cex
   1
col
   black
fontsize
   12
fontface
   plain
rot
   default varies by type of axis
x
   0.5
y
   0.5
type
   NULL
add
   FALSE

maintitle A list; places a single title above the entire graphic and control its format. See plottitle for parameters

A list; places titles below each plot and controls their format.
undertitle  See plottitle for parameters

rowtitle  A list; places titles to the left of each row of plots and controls their format. See plottitle for parameters

columntitle  A list; places titles above each column of plots and controls their format. See plottitle for parameters

A list controlling the printing of gridlines in the plotting regions. Each input may be a scalar (to set the parameter globally for all plots) or a vector (to set the parameter plot by plot):

gridlines  type  
* A string containing the first letters of each axis which should have gridlines attached to its ticks; e.g., "xy" for gridlines on the x and y axes or "t" for gridlines on the top axis. Default is no gridlines

lwd  *Width of gridlines. Default is 0.15

col  *Color of gridlines. Default is gray50

lty  *Line type of gridlines. Default is solid

frame  Logical; draw a frame around plots (use a vector to set for specific plots). Default is FALSE

leeway  When finding plot region limits automatically, add a little leeway beyond data limits. Default is 0

width  A list of default widths of various plot elements

height  A list of default heights of various plot elements

defaults  List object with default settings for title heights and widths, and other tile parameters

layoutonly  Logical; draw only axes, titles, and frames (default is FALSE)

verbose  Logical; display progress report for error checking (default is FALSE)

Details  Explain concept—many of the rec of info vis in one package, with extensive
facilities for both EDA and summarizing inference from regression-like models

The tile package operates in two stages. The user first creates one or more traces to be plotted. A trace is a single set of data to be added to the plot, and may be as simple as a text label or a series of line segments, or as complex as a set of points combined with a best fit line and that fit line's various confidence intervals printed as shaded regions. All the parameters affecting the construction and appearance of the trace are set using one of the many functions in the tile package. These include several functions for creating primitives <add table here>

And several functions for constructing complex traces for model or data exploration <add table here>

After creating all the traces you wish to plot, XXX

Explain traces—and note that next thing to read is file for style of interest, including examples (which may involve not only learning how to create traces, but how to generate the data to add to the trace)

Explain role of tile: drawing everything outside the plotting area, and controlling the transformation of the plotting area, axes and rugs, titles

Explain draw-at-once philosophy, including layers and output to file, and grid based graphics

Explain axes, including transformation, attachment, and automatic limits

Explain rugs, and warn some styles may not make sense with logged axes

Explain titles

Explain widths and heights

Explain annotation across plots

Note possibility of extension, and contact

Value

tile is mainly called for the side-effect of saving a graphic to the requested
device. However, it does return a list containing all the user inputs, default settings, and numerous internal values. This list contains a copy of the entire plot in the slot grob; to draw this graphical object to the current device, give the grob as input to \texttt{tileDraw}.

**Author(s)**

Christopher Adolph \texttt{<cadolph@u.washington.edu>}

**See Also**

\texttt{lineplot, scatter, ropeladder, nightplot, riceplot, linesTile, pointsTile, textTile, polylinesTile, polygonTile}

[Package \textit{tile} version 0.2 \texttt{Index}]
tileDraw(tile)

R Documentation
(Re-)draw a tile graphical object

Description

This internal function draws a tile graphical object to the current device. It is mainly of use of an internal function; most users should simply use tile to create and save tile graphics to a file.

Usage

tileDraw(grob, defaultlayer=10)

Arguments

grob A tile or grid graphical object; e.g., the object in the slot grob in the value returned by tile
defaultlayer The layer to draw any sub-elements of grob. Layers with higher numbers are drawn first, then lower numbers are drawn on top.

Details

This internal function is provided for completeness, and is not necessary to create tile graphics with detailed annotations, which can be done through careful construction of a single call to tile.

However, if users wish to add to or modify an existing tile grob, they may do so by directly editing the grob returned by tile, then sending it to this function to place it on the current device. Care should be taken to set up the current device to have appropriate height, width, and pointsize.

Value

tileDraw is mainly called for the side-effect of drawing a tile grob. It invisibly returns the grob after drawing.

Author(s)

Christopher Adolph <cadolph@u.washington.edu>

See Also
tile

[Package tile version 0.2 Index]