Package: baffle (via r-universe)

October 18, 2024

Title Make Waffle Plots with Base Graphics

Version 0.2.2

Description Waffle plots are rectangular pie charts that represent a quantity or abundances using colored squares or other symbol. This makes them better at transmitting information as the discrete number of squares is easier to read than the circular area of pie charts. While the original waffle charts were rectangular with 10 rows and columns, with a single square representing 1%, they are nowadays popular in various infographics to visualize any proportional ratios.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

Imports grDevices

Suggests testthat (>= 3.0.0), knitr, rmarkdown, png

Config/testthat/edition 3

URL https://j-moravec.github.io/baffle/, https://github.com/j-moravec/baffle

BugReports https://github.com/j-moravec/baffle/issues

Repository https://j-moravec.r-universe.dev

RemoteUrl https://github.com/j-moravec/baffle

RemoteRef HEAD

RemoteSha ed400d8f232ebe76850e91134bbf1804bf28f9b9

Contents

`rasters` *Plot raster image*

Description

Plot raster image centered at ‘x‘ and ‘y‘ coordinates scaled to diameter ‘d‘.

Usage

```
rasters(x, y, image, d = 0.9, dx = NA, dy = NA, rotate = 0, ...)
```

Arguments

`x, y`	coordinates
`image`	raster image
`d`	**optional** diameter, see details
`dx, dy`	**optional** diameter in either coordinate direction
`rotate`	**optional** clockwise rotation in degrees (0-360°)
`...`	**optional** other parameters passed to [graphics::rasterImage()], such as ‘interpolate‘

Details

The ‘rasters()‘ function is a convenient wrapper around [graphics::rasterImage()] with similar interface to the [Shapes] functions available in this package (such as [square()], [circle()] and [rcpoly()]).

The raster image is plotted centered at the ‘x‘ and ‘y‘ coordinates and scaled to the diameter size ‘d‘. When ‘dx‘ and ‘dy‘ are ‘NA‘, the proportions of the raster are kept unchanged, otherwise they are scaled to the specified size in either direction. This scaling is done before rotation.

Unlike in ‘rasterImage‘, the rotation is performed clockwise and the rotation axis is the center of the raster (i.e., the provided x and y coordinates), rather than the bottom left coordinate ‘x0‘. This rotation is performed after scaling.

As of yet, the rotation is accurate only when the aspect ratio is set to 1 (‘asp=1‘) through the ‘graphics::plot.window()‘ call.

The ‘rasters()‘ function is fully vectorized.

Value

No return value, called for side effects

See Also

[grDevices::as.raster()] and [graphics::rasterImage()]

Examples

```
# create plotting window
plot.new(); plot.window(c(-1,1), c(-1,1), asp=1); axis(1); axis(2)

# create raster image, alpha is convenient when overplotting
img = matrix(adjustcolor("black", alpha.f=0.3), 3, 3)
img[2, 2] = adjustcolor("white", alpha.f=0.3)
img = as.raster(img)

rasters(0, 0, img)

# interpolate=FALSE makes quite a difference
rasters(0, 0, img, interpolate=FALSE)

# arguments are vectorized, standard recycling rules apply
rasters(0, 0, img, interpolate=FALSE, rotate=c(30, 60, 90))
rasters(c(-1, -0.5,  0.5, 1), c(1, 0.5, -0.5, -1), img, interpolate=FALSE)
```

---

round_rect *Plot a rectangle with rounded corners*

Description

Plot one or more rectangles with rounded corners.

Usage

```
round_rect(x1, y1, x2, y2, xr = 0.2, yr = 0.2, n = 10, ...)
```

Arguments

x1, y1	x and y coordinates of the bottom left corner
x2, y2	x and y coordinates of the top right corner
xr, yr	the proportion of length of a side of the rectangle, from which the rounded corners start. Values between 0 and 1 are permitted. See details.
n	the number of points used to approximate the curvature of the rounded corner
...	other parameters passed to [graphics::polygon()].

Details

The rounded corner is a part of a circle (one quarter) drawn between two points of neighbouring sides of a rectangle. The relative position of these two points determine how rounded will the final shape be. The position of these points is determined by the parameters ‘xr‘ and ‘yr‘, which determine the proportion of side x and y, from which the rounded corner is drawn. Values between 0 and 1 are permitted, but given the symmetricity of a rectangle, values larger than 0.5 are reflected back (modulo 0.5). When both xr and yr are 0 or 1, normal rectangle without rounded corners is drawn. When xr and yr are 0.5, ellipsis is drawn.

Value

No return value, called for side effects

`Shapes` *Plot convex polygons*

Description

A collection of functions for plotting polygonal shapes.

Usage

```
square(x, y, d = 0.9, ...)

rectangle(x, y, d = 0.9, dx = d, dy = d, ...)

rounded_square(x, y, d = 0.9, r = 0.2, n = 10, ...)

rounded_rectangle(
  x,
  y,
  d = 0.9,
  dx = d,
  dy = d,
  r = 0.2,
  rx = r,
  ry = r,
  n = 10,
  ...
)

circle(x, y, d = 0.9, n = 1000, ...)

ellipse(x, y, d = 0.9, dx = d, dy = d, n = 1000, ...)

rcpoly(x, y, n, d = 0.9, rotate = 0, ...)

shapes(x, y, n, d = 0.9, dx = d, dy = d, rotate = 0, ...)
```

Arguments

x, y	coordinates
d	**optional** diameter, see details
...	**optional** graphical parameters ‘col‘, ‘border‘, ‘lty‘ and ‘lwd‘ passed to [graphics::polygon()] or [graphics::rect()]
dx, dy	**optional** diameter in either coordinate direction
r, rx, ry	**optional** for rounded corner, the proportion of side marking the start of rounded corner.
n	the number of vertices of polygon, with the minimum of three (triangle). Large ‘n‘, such as ‘n=1000‘ approximate circle. The vertices start at the 12 o’clock position and are placed clockwise in a regular intervals. For ‘rounded_square‘ and ‘rounded_rectangle‘, the number of vertices that approximate the circular shape of the rounded corners. Small ‘n‘, such as ‘n=10‘ should have sufficient smoothness for most applications.
rotate	**optional** clockwise rotation in degrees (0-360°), not available for ‘square‘ and ‘rectangle‘

Details

Polygons are drawn centered on the ‘x‘ and ‘y‘ coordinates, with a diameter ‘d‘ (or ‘dx‘ and ‘dy‘). Typically, different shapes are obtained through a parametrization of the ‘shapes()‘ function, which draws a convex polygon using the [graphics::polygon()], with the exception of ‘square()‘ and ‘rectangle()‘ function, which use the [graphics::rect()] function instead, behave slightly differently, and should be slightly faster, and the ‘rounded_square()‘ and ‘rounded_rectangle()‘, which use ‘round_rect()‘ function.

The diameter parameter ‘d‘ is interpreted differently depending for ‘square()‘ and ‘rectangle()‘ and for other polygonal functions build on the ‘shapes()‘ function (‘circle()‘, ‘ellipse()‘ and ‘rcpoly()‘). For the ‘square()‘ and ‘rectangle()‘, the diameter is the size of the square, ‘d=1‘ thus fills the whole 1x1 tile. For ‘shapes()‘ function, ‘d‘ is the diameter of the inscribed circle to the square of size ‘d‘. This is more convenient solution to prevent accidental overplotting when individual shapes are plotted next to each other in regular intervals, the distance between such points would be equal to the diameter in both cases. See examples.

All shapes function accept the graphical parameters ‘col‘, ‘border‘, ‘lty‘ and ‘lwd‘, which are passed to the [graphics::polygon()] and [graphics::rect()]. Apart of a different default values, they behave in the same way.

All parameters are vectorized and will recycle as required, with the typical warning if parameters are not multiply of each other. This can be used to create pleasant geometric images. See examples.

Value

No return value, called for side effects

Functions

- `square()`: draw squares
- `rectangle()`: draw rectangles

- `rounded_square()`: draw squares with rounded corners
- `rounded_rectangle()`: draw rectangles with rounded corners
- `circle()`: draw circles
- `ellipse()`: draw ellipses
- `rcpoly()`: draw regular convex polygons
- `shapes()`: draw convex polygons

See Also

[graphics::polygon()] and [graphics::rect()] for the underlying plotting functions

Examples

```
plot(0, 0) # create plotting window

# Following calls are equivalent
square(0, 0, 1)
rectangle(0, 0, 1)
rectangle(0, 0, dx=1, dy=1)

# Not equivalent to `square`
rcpoly(0, 0, 4, d=1)

# Same output as `square`, but not equivalent
rcpoly(0, 0, 4, d=sqrt(2), rotate=45)

# Vectorizing parameters
plot(0, 0)
rotate = seq(0, 18, by=30)
d = seq(1, by=-0.1, length.out = length(rotate))
rcpoly(0,0,3, border="red", lwd=3, rotate=rotate, d=d)
```

`waffle` *Make a waffle chart.*

Description

Waffle chart is a pie chart that visually represents abundances with the number of squares. The waffle chart consists of squares plotted on a rectangular lattice according to a design matrix that is constructed from a vector of abundances or can be provided directly.

Usage

```
waffle(
  x,
  f = NULL,
  ...,
  nrow = NULL,
  ncol = NULL,
  byrow = TRUE,
  from = "bottomleft",
  stacked = TRUE,
  gap = 0,
  horiz = TRUE,
  add = FALSE
)

waffle.mat(x, f = square, ..., add = FALSE)

design(
  x,
  nrow = NULL,
  ncol = NULL,
  byrow = TRUE,
  from = "bottomleft",
  stacked = TRUE,
  horiz = TRUE,
  gap = 0
)
```

Arguments

x	a vector of abundances or a design matrix (see details)
f	**optional** a shape function (see details)
...	**optional** other parameters passed to the ‘f‘ function
nrow	**optional** the number of rows
ncol	**optional** the number of columns
byrow	**optional** fill matrix by rows or by columns
from	**optional** starting position from which the matrix is filled, one of "bottomleft", "bottomright", "topleft" or "topright".
stacked	**optional** if FALSE, produce an unstacked waffle chart, see details
gap	**optional** gap between unstacked subwaffles, works only for unstacked waffle charts
horiz	**optional** if FALSE, vertical instead of horizontal waffle chart is produced, works only for unstacked waffle charts
add	**optional** whether to add to a current plot

Details

The ‘waffle()‘ function accepts a vector of abundances and plots a waffle chart. This is done by first constructing a design matrix using the ‘design()‘ function, which is then parsed to the ‘waffle.mat()‘ function.

The ‘design()‘ functions construct a design matrix according. The design matrix is filled by an integer vector derived from the abundance vector ‘x‘. Each integer correspond to the order of abundances in ‘x‘, with the quantity equal to the value in ‘x‘. This means that for the ‘x=c(3,5)‘, the design matrix will be filled with three ‘1‘ and five ‘2‘, all other cells of the design matrix are set as unknown values ‘NA‘.

By default, the design matrix is filled by row, starting from the bottom left corner, this can be changed by setting the variables ‘byrow‘ and ‘from‘. By setting ‘byrow=FALSE‘, the matrix is filled by columns first.

If ‘ncol‘ or ‘nrow‘ is not specified, a squared matrix that will fit the sum of abundances will be constructed.

By default, the design matrix is stacked, i.e., all the elements are placed in a single matrix object, similarly how barplot produces stacked lines. By specifying ‘stack=FALSE‘, the ‘design()‘ unstacks the elements, each are then placed in their own matricies, which are then connected by ‘NA‘ elements of size ‘gap‘. More complex

If the input vector ‘x‘ is a named vector, the names are preserved in the ‘levels‘ attribute. These levels are not currently used, but might be used in the future for automatic legend creation and subsetting.

The function ‘waffle.mat()‘ accepts a custom-made design matrix and thus allows a better control of colored regions. It is called internally by the ‘waffle()‘ function, which serves as an easy to use interface for the ‘waffle.mat()‘. For this reason, the ‘waffle.mat()‘ does not checks for the validity of input arguments and does not set a default colors.

The assumed and allocated coordinate system is from ‘0‘ to ‘ncol‘ for x and from ‘0‘ to ‘ncol‘ for y. Squares are filled from top right corner of this coordinate system.

If ‘add=FALSE‘, a new window with a fixed aspect ratio x/y=1 is allocated so that plotted polygons are squares (by default). This might cause the plot margins, and thus the main title, to be quite far away. In this case, plotting the title using ‘text()‘ instead of ‘title()‘ might be a better idea. In this case, the coordinates might be: `text(x=(ncol+2)/2, y=nrow+1,...)`

Value

‘design()‘ returns a design matrix (see details), in addition, if ‘x‘ is a named vector, the names are preserved in the ‘levels‘ attribute. ‘waffle()‘ and ‘waffle.mat()‘ do not have a return value

See Also

the ‘waffle‘ package for a ‘ggplot2‘ version.

Examples

```
waffle(c(50,25,25))
waffle(c(25,75), col=c("darkorchid", "lightgray"))
waffle(c(14,8,4), nrow=3)
```

```
# custom design matrix with a more complex structure
cols = palette.colors(3, "Set 1")
design_mat = matrix(NA, 7,10)
design_mat[1,] = 1
design_mat[2,1:8] = 1
design_mat[4,] = 2
design_mat[5,1:4] = 2
design_mat[7,1:5] = 3
waffle.mat(design_mat, col=cols)
```

Index