

Package ‘netplot’

July 23, 2026

Title Beautiful Graph Drawing

Version 0.4-0

Description A graph visualization engine that emphasizes on aesthetics at the same time providing default parameters that yield out-of-the-box-nice visualizations. The package is built on top of 'The Grid Graphics Package' and seamlessly work with 'igraph' and 'network' objects.

Depends R (>= 3.4.0), grid

License MIT + file LICENSE

Encoding UTF-8

Imports graphics, grDevices, igraph, network, sna, stats

Suggests knitr, rmarkdown, markdown, igraphdata, intergraph, ggraph, gridGraphics, ggplot2, gridExtra, gridBase, magrittr, tinytest

VignetteBuilder knitr

URL <https://github.com/USCCANA/netplot>,
<https://usccana.github.io/netplot/>

BugReports <https://github.com/USCCANA/netplot/issues>

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author George Vega Yon [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-3171-0844>>),
Porter Bischoff [aut] (ORCID: <<https://orcid.org/0009-0004-6742-6281>>)

Maintainer George Vega Yon <g.vegayon@gmail.com>

Repository CRAN

Date/Publication 2026-07-23 10:50:22 UTC

Contents

colorkey	2
colorRamp2	4
locate_vertex	5
make_colors	6
netplot-formulae	7
nplot	9
nplot_base	16
nplot_legend	19
npolygon	20
piechart	21
segments_gradient	24
set_gpar	24
Index	27

colorkey	<i>Function to create a color key</i>
----------	---------------------------------------

Description

Function to create a color key

Usage

```
colorkey(  
  x0,  
  y0,  
  x1,  
  y1,  
  cols = c("white", "steelblue"),  
  tick.range = c(0, 1),  
  tick.marks = seq(tick.range[1], tick.range[2], length.out = 5L),  
  label.from = NULL,  
  label.to = NULL,  
  nlevels = 100,  
  main = NULL,  
  relative = TRUE,  
  tick.args = list(),  
  label.args = list(),  
  main.args = list()  
)
```

Arguments

<code>x0, x1, y0, y1</code>	Numeric scalars. Coordinates of the lower left and upper right points where the color key will be drawn as proportion of the plotting region.
<code>cols</code>	Character scalar. Colors specifications to create the color palette.
<code>tick.range, tick.marks</code>	Numeric vectors specifying the range and the tickmarks respectively.
<code>label.from, label.to</code>	Character scalar. Labels of the lower and upper values of the color key.
<code>nlevels</code>	Integer scalar. Number of levels to extrapolate.
<code>main</code>	Character scalar. Title of the colorkey.
<code>relative</code>	Logical scalar. When TRUE the color key is drawn relative to the plotting region area taking <code>x0</code> , <code>x1</code> , <code>y0</code> , <code>y1</code> as relative location.
<code>tick.args, label.args, main.args</code>	Lists of arguments passed to graphics::text for drawing ticks, labels and main respectively.

Value

NULL.

Examples

```
set.seed(22231)

# A random figure
dat <- matrix(runif(100*3), ncol = 3)
col <- colorRamp2(c("blue", "white", "red"))

plot(
  dat[,1], dat[,2],
  col = rgb(col(dat[,3]), maxColorValue=255),
  cex=2, pch=20
)

# Pretty color key
colorkey(
  x0 = .60, y0 = .80,
  x1 = .95, y1 = .95,
  cols = c("blue", "white", "red"),
  main = "Some color scale"
)
```

colorRamp2	<i>A faster implementation of grDevices::colorRamp for linear interpolation.</i>
------------	--

Description

A faster implementation of [grDevices::colorRamp](#) for linear interpolation.

Usage

```
colorRamp2(x, alpha = TRUE, thresholds = NULL)
```

Arguments

x	A vector of colors.
alpha	Logical scalar. When TRUE This implementation of colorRamp can be 2 or more times faster than the grDevices version. It is intended for consecutive calls (i.e. in a loop) to improve performance. It is equivalent to the linear interpolation of the function colorRamp.
thresholds	A numeric vector of length length(x). Optional threshold levels so that the mixing can be different that even.

Value

A function as in [grDevices::colorRamp](#).

Examples

```
# Creating a function for 2 colors
myf <- colorRamp2(c("black", "steelblue"))
f <- colorRamp(c("black", "steelblue"))

plot.new()
plot.window(xlim = c(0,2), ylim = c(1, 11))

# These should be the same colors
rect(
  xleft  = 0,
  xright = 1,
  ybottom = 1:10,
  ytop    = 2:11,
  col = rgb(myf((1:10)/10), maxColorValue = 255)
)
rect(
  xleft  = 1,
  xright = 2,
  ybottom = 1:10,
  ytop    = 2:11,
  col = rgb(f((1:10)/10), maxColorValue = 255)
```

```

)

# Another example setting different thresholds
myf <- colorRamp2(c("black", "steelblue"))
myf2 <- colorRamp2(c("black", "steelblue"), thresholds=c(0, .7))

plot.new()
plot.window(xlim = c(0,2), ylim = c(1, 11))

# These should be the same colors
rect(
  xleft  = 0,
  xright = 1,
  ybottom = 1:10,
  ytop    = 2:11,
  col = rgb(myf((1:10)/10), maxColorValue = 255)
)
rect(
  xleft  = 1,
  xright = 2,
  ybottom = 1:10,
  ytop    = 2:11,
  col = rgb(myf2((1:10)/10), maxColorValue = 255)
)

```

locate_vertex

Find a vertex in the current plot

Description

This function is a wrapper of `grid::grid.locator()`, and provides a way to find the coordinates of a vertex in the current plot. It is useful to identify the vertex that is being clicked in a plot.

Usage

```
locate_vertex(x = NULL)
```

Arguments

`x` An object of class `netplot`

Details

This function only works in interactive mode. Once it is called, the user can click on a vertex in the plot. The function will return the name of the vertex, the x and y coordinates and the viewport where it is located. If `x` is not specified, the last plotted `netplot` object will be used.

Value

A list with the name of the vertex, the x and y coordinates and the viewport where it is located.

Examples

```
library(igraph)
library(netplot)
set.seed(1)
x <- sample_smallworld(1, 200, 5, 0.03)

# Plotting
nplot(x)

# Clicking (only works in interactive mode)
if (interactive()) {
  res <- locate_vertex()
  print(res)
}
```

make_colors

Create a vector of colors for vertices and edges

Description

Using vertex/edge attributes, these functions return vectors of colors that can be used either during the creation of the [nplot](#) object, or afterwards when changing gpar (graphical parameter) values with [set_gpar](#).

Usage

```
make_colors(dat, categorical = FALSE, color_map = grDevices::hcl.colors)

make_edges_colors(x, eattr, ...)

make_vertex_colors(x, vattr, ...)
```

Arguments

dat	A vector of data to generate the color from.
categorical	Logical. When TRUE sets the colors as categories.
color_map	A function to generate a palette.
x	A graph of class network or igraph.
...	Further arguments passed to make_colors.
vattr, eattr	Character. Names of either vertex or edge variables to be used for generating the colors.

Details

If no attribute is provided, then by default the colors are set according to indegree.
 x can be either a graph of class `igraph` or `network`.

Value

A vector of colors with the attribute `color_map`. The color map used to generate the colors.

Examples

```
data(UKfaculty, package="igraphdata")
col <- make_vertex_colors(UKfaculty, "Group")

if (require(magrittr)) {

  nplot(UKfaculty) %>%
    set_vertex_gpar("core", fill = col, col=col) %>%
    set_vertex_gpar("frame", fill = col, col=col, alpha=.7) %>%
    set_edge_gpar(col="gray50", fill="gray50", alpha=.5)
}
```

 netplot-formulae

Edge-color formulas in netplot

Description

Edge colors in both `nplot()` and `set_edge_gpar()` can be specified using a formula based on `ego()` and `alter()` (source and target). This way the user can set various types of combination varying the mixing of the colors, the alpha levels, and the actual mixing colors to create edge colors.

Usage

```
color_formula(x, col, alpha, env, type, mix = 1, postfix = NULL)

ego(...)

alter(...)
```

Arguments

x	An object of class <code>netplot</code> .
col	Any valid color. Can be a single color or a vector.
alpha	Number. Alpha levels
env, type, postfix	For internal use only.
mix	Number. For mixing colors between <code>ego</code> and <code>alter</code>
...	Passed to <code>color_formula</code> .

Details

The formula is one-sided and combines the special terms `ego()` (the source vertex) and `alter()` (the target vertex) with `+`. Each term draws its color from the corresponding endpoint's `vertex.color` unless a `col` is supplied, and accepts:

- `col`: the base color to use for that endpoint.
- `alpha`: transparency, from 0 (fully transparent) to 1 (opaque).
- `mix`: how much weight that endpoint's color receives when the two are blended along the edge.

For instance, `~ ego(alpha = .1, col = "gray") + alter` (the default) fades each edge from a faint gray at the source to the target's color, producing the characteristic netplot look.

This grammar is specific to *edge colors*. To map other aesthetics (vertex color, shape, size, or edge width) from a graph attribute, pass a formula naming the attribute directly to `nplot()` (e.g. `vertex.color = ~ group`); see the "Mapping attributes with formulas" section of `nplot()`.

Value

Nothing. These functions are called internally when using formulas. `color_formula` modifies the environment `env`.

See Also

`nplot()` for the attribute-mapping formulas, and `set_edge_gpar()`.

Examples

```
if (require(gridExtra) & require(magrittr)) {
  library(igraph)
  net <- make_ring(4)

  set.seed(1)
  np <- nplot(net, vertex.color = grDevices::hcl.colors(4), vertex.size.range=c(.1, .1))
  np %<>% set_edge_gpar(lwd = 4)

  grid.arrange(
    np,
    np %>% set_edge_gpar(col =~ego + alter),
    np %>% set_edge_gpar(col =~ego(alpha=0) + alter),
    np %>% set_edge_gpar(col =~ego + alter(alpha=0)),
    np %>% set_edge_gpar(col =~ego(mix=0) + alter(mix=1)),
    np %>% set_edge_gpar(col =~ego(mix=1) + alter(mix=0))
  )
}
```

nplot*Plot a network*

Description

nplot() is the main function of the **netplot** package. It draws a network using the **grid** graphics system (the same engine that powers **ggplot2**), emphasizing aesthetics and providing sensible defaults that yield out-of-the-box nice visualizations. Compared with base igraph/network plots, nplot() auto-scales vertices and edges relative to the plotting device, draws truly curved edges, mixes edge colors from their endpoints, and fills the device efficiently.

Usage

```
nplot(  
  x,  
  layout,  
  vertex.size = 1,  
  bg.col = "transparent",  
  vertex.nsides = 10,  
  vertex.color = grDevices::hcl.colors(1),  
  vertex.size.range = c(0.01, 0.03, 4),  
  vertex.frame.color = NULL,  
  vertex.rot = 0,  
  vertex.frame.prop = 0.2,  
  vertex.label = NULL,  
  vertex.label.fontsize = NULL,  
  vertex.label.color = adjustcolor("black", alpha.f = 0.8),  
  vertex.label.fontfamily = "sans",  
  vertex.label.fontface = "plain",  
  vertex.label.show = 0.3,  
  vertex.label.range = c(5, 15),  
  edge.width = 1,  
  edge.width.range = c(1, 2),  
  edge.arrow.size = NULL,  
  edge.color = ~ego(alpha = 0.1, col = "gray") + alter,  
  edge.curvature = pi/3,  
  edge.line.lty = "solid",  
  edge.line.breaks = 5,  
  sample.edges = 1,  
  skip.vertex = FALSE,  
  skip.edges = FALSE,  
  skip.arrows = skip.edges,  
  add = FALSE,  
  zero.margins = TRUE,  
  edgelist  
)
```

```

## S3 method for class 'igraph'
nplot(
  x,
  layout = igraph::layout_nicely(x),
  vertex.size = igraph::degree(x, mode = "in"),
  bg.col = "transparent",
  vertex.nsides = 10,
  vertex.color = grDevices::hcl.colors(1),
  vertex.size.range = c(0.01, 0.03, 4),
  vertex.frame.color = NULL,
  vertex.rot = 0,
  vertex.frame.prop = 0.2,
  vertex.label = igraph::vertex_attr(x, "name"),
  vertex.label.fontsize = NULL,
  vertex.label.color = adjustcolor("black", alpha.f = 0.8),
  vertex.label.fontfamily = "sans",
  vertex.label.fontface = "plain",
  vertex.label.show = 0.3,
  vertex.label.range = c(5, 15),
  edge.width = igraph::edge_attr(x, "weight"),
  edge.width.range = c(1, 2),
  edge.arrow.size = NULL,
  edge.color = ~ego(alpha = 0.1, col = "gray") + alter,
  edge.curvature = pi/3,
  edge.line.lty = "solid",
  edge.line.breaks = 5,
  sample.edges = 1,
  skip.vertex = FALSE,
  skip.edges = FALSE,
  skip.arrows = !igraph::is_directed(x),
  add = FALSE,
  zero.margins = TRUE,
  edgelist
)

## S3 method for class 'network'
nplot(
  x,
  layout = sna::gplot.layout.kamadakawai(x, NULL),
  vertex.size = sna::degree(x, cmode = "indegree"),
  bg.col = "transparent",
  vertex.nsides = 10,
  vertex.color = grDevices::hcl.colors(1),
  vertex.size.range = c(0.01, 0.03, 4),
  vertex.frame.color = NULL,
  vertex.rot = 0,
  vertex.frame.prop = 0.2,
  vertex.label = network::get.vertex.attribute(x, "vertex.names"),

```

```

vertex.label.fontsize = NULL,
vertex.label.color = adjustcolor("black", alpha.f = 0.8),
vertex.label.fontfamily = "sans",
vertex.label.fontface = "plain",
vertex.label.show = 0.3,
vertex.label.range = c(5, 15),
edge.width = network::get.edge.attribute(x, "weight"),
edge.width.range = c(1, 2),
edge.arrow.size = NULL,
edge.color = ~ego(alpha = 0.1, col = "gray") + alter,
edge.curvature = pi/3,
edge.line.lty = "solid",
edge.line.breaks = 5,
sample.edges = 1,
skip.vertex = FALSE,
skip.edges = FALSE,
skip.arrows = !network::is.directed(x),
add = FALSE,
zero.margins = TRUE,
edgelist
)

```

```
## S3 method for class 'matrix'
```

```

nplot(
  x,
  layout,
  vertex.size = 1,
  bg.col = "transparent",
  vertex.nsides = 10,
  vertex.color = grDevices::hcl.colors(1),
  vertex.size.range = c(0.01, 0.03, 4),
  vertex.frame.color = NULL,
  vertex.rot = 0,
  vertex.frame.prop = 0.2,
  vertex.label = NULL,
  vertex.label.fontsize = NULL,
  vertex.label.color = adjustcolor("black", alpha.f = 0.8),
  vertex.label.fontfamily = "sans",
  vertex.label.fontface = "plain",
  vertex.label.show = 0.3,
  vertex.label.range = c(5, 15),
  edge.width = 1,
  edge.width.range = c(1, 2),
  edge.arrow.size = NULL,
  edge.color = ~ego(alpha = 0.1, col = "gray") + alter,
  edge.curvature = pi/3,
  edge.line.lty = "solid",
  edge.line.breaks = 5,

```

```

    sample.edges = 1,
    skip.vertex = FALSE,
    skip.edges = FALSE,
    skip.arrows = skip.edges,
    add = FALSE,
    zero.margins = TRUE,
    edgelist
)

## Default S3 method:
nplot(
  x,
  layout,
  vertex.size = 1,
  bg.col = "transparent",
  vertex.nsides = 10,
  vertex.color = grDevices::hcl.colors(1),
  vertex.size.range = c(0.01, 0.03, 4),
  vertex.frame.color = NULL,
  vertex.rot = 0,
  vertex.frame.prop = 0.2,
  vertex.label = NULL,
  vertex.label.fontsize = NULL,
  vertex.label.color = adjustcolor("black", alpha.f = 0.8),
  vertex.label.fontfamily = "sans",
  vertex.label.fontface = "plain",
  vertex.label.show = 0.3,
  vertex.label.range = c(5, 15),
  edge.width = 1,
  edge.width.range = c(1, 2),
  edge.arrow.size = NULL,
  edge.color = ~ego(alpha = 0.1, col = "gray") + alter,
  edge.curvature = pi/3,
  edge.line.lty = "solid",
  edge.line.breaks = 5,
  sample.edges = 1,
  skip.vertex = FALSE,
  skip.edges = FALSE,
  skip.arrows = skip.edges,
  add = FALSE,
  zero.margins = TRUE,
  ...,
  edgelist
)

## S3 method for class 'netplot'
print(x, y = NULL, newpage = TRUE, legend = TRUE, ...)

```

Arguments

<code>x</code>	A graph. It supports networks stored as <code>igraph</code> , <code>network</code> , and <code>matrices</code> objects (see details).
<code>layout</code>	Numeric two-column matrix with the graph layout in x/y positions of the vertices.
<code>vertex.size</code>	Numeric vector of length <code>vcount(x)</code> . Absolute size of the vertex from 0 to 1. Can also be a one-sided formula (e.g. <code>~ degree</code>) naming a numeric vertex attribute to map sizes from (see "Mapping attributes with formulas").
<code>bg.col</code>	Color of the background.
<code>vertex.nsidess</code>	Numeric vector of length <code>vcount(x)</code> . Number of sides of the vertex. E.g. three is a triangle, and 100 approximates a circle. Can also be a one-sided formula (e.g. <code>~ group</code>) naming a vertex attribute; each unique value is then mapped to a distinct shape (see "Mapping attributes with formulas").
<code>vertex.color</code>	Vector of length <code>vcount(x)</code> . Vertex HEX or built in colors. Can also be a one-sided formula (e.g. <code>~ group</code>) naming a vertex attribute to color vertices by (see "Mapping attributes with formulas").
<code>vertex.size.range</code>	Numeric vector of length 2 or 3, or <code>NULL</code> . Relative size for the minimum and maximum of the plot, and curvature of the scale. The third number is used as <code>size^rel[3]</code> . If <code>NULL</code> , scaling is suppressed and <code>vertex.size</code> is used as is.
<code>vertex.frame.color</code>	Vector of length <code>vcount(x)</code> . Border of vertex in HEX or built in colors.
<code>vertex.rot</code>	Vector of length <code>vcount(x)</code> in Radians. Passed to npolygon , elevation degree from which the polygon is drawn.
<code>vertex.frame.prop</code>	Vector of length <code>vcount(x)</code> . What proportion of the vertex does the frame occupy (values between 0 and 1).
<code>vertex.label</code>	Character vector of length <code>vcount(x)</code> . Labels.
<code>vertex.label.fontsize</code>	Numeric vector.
<code>vertex.label.color</code>	Vector of colors of length <code>vcount(x)</code> .
<code>vertex.label.fontfamily</code>	Character vector of length <code>vcount(x)</code> .
<code>vertex.label.fontface</code>	See grid::gpar
<code>vertex.label.show</code>	Numeric scalar. Proportion of labels to show as the top ranking according to <code>vertex.size</code> .
<code>vertex.label.range</code>	Numeric vector of size 2 or 3. Relative scale of <code>vertex.label.fontsize</code> in points (see grid::gpar).

<code>edge.width</code>	Numeric vector of length <code>ecount(x)</code> . Relative edge widths. Values are normalized and then mapped to the range specified by <code>edge.width.range</code> , unless <code>edge.width.range</code> is <code>NULL</code> . For <code>nplot.igraph</code> and <code>nplot.network</code> , defaults to the "weight" edge attribute if present; otherwise all edges use width 1. Can also be a one-sided formula (e.g. <code>~ weight</code>) naming a numeric edge attribute (see "Mapping attributes with formulas").
<code>edge.width.range</code>	Numeric vector of length 2, or <code>NULL</code> . The minimum and maximum line widths (in points) to use when mapping <code>edge.width</code> values. For example, <code>c(1, 4)</code> maps the smallest edge weight to 1pt and the largest to 4pt. If <code>NULL</code> , scaling is suppressed and <code>edge.width</code> is used as is.
<code>edge.arrow.size</code>	Vector of length <code>ecount(x)</code> from 0 to 1.
<code>edge.color</code>	A vector of length <code>ecount(x)</code> . In HEX or built in colors. Can be <code>NULL</code> in which case the color is picked as a mixture between ego and alters' <code>vertex.color</code> values.
<code>edge.curvature</code>	Numeric vector of length <code>ecount(x)</code> . Curvature of edges in terms of radians.
<code>edge.line.lty</code>	Vector of length <code>ecount(x)</code> . Line types in R (e.g.- 1 = Solid, 2 = Dashed, etc).
<code>edge.line.breaks</code>	Vector of length <code>ecount(x)</code> . Number of vertices to draw (approximate) the arc (edge).
<code>sample.edges</code>	Numeric scalar between 0 and 1. Proportion of edges to sample.
<code>skip.vertex</code> , <code>skip.edges</code> , <code>skip.arrows</code>	Logical scalar. When <code>TRUE</code> the object is not plotted.
<code>add</code>	Logical scalar.
<code>zero.margins</code>	Logical scalar.
<code>edgelist</code>	An edgelist.
<code>y, ...</code>	Ignored
<code>newpage</code>	Logical scalar. When <code>TRUE</code> calls <code>grid::grid.newpage</code> .
<code>legend</code>	Logical scalar. When <code>TRUE</code> it adds a legend.

Details

Vertex and edge aesthetics can be set directly (passing a vector) or, conveniently, mapped from graph attributes using a *formula* interface (see the "Mapping attributes with formulas" section below). The returned object is a **grid** grob, so it can be further edited with `set_vertex_gpar()` / `set_edge_gpar()`, combined with other grid graphics (e.g. via `gridExtra::grid.arrange()`), or annotated with a legend through `nplot_legend()`.

When `x` is of class `matrix`, it will be passed to `igraph::graph_from_adjacency_matrix()`.

Value

An object of class `c("netplot", "gTree", "grob", "gDesc")`. The object has an additional set of attributes:

- `.xlim`, `.ylim` vector of size two with the x-axis/y-axis limits.
- `.layout` A numeric matrix of size `vcount(x) * 2` with the vertices positions
- `.edgelist` A numeric matrix, The edgelist.

In the case of `nplot.default`, an object of class `netplot` and `grob` (see [grid::grob](#)) with the following slots:

- `children` The main grob of the object.
- `name` Character scalar. The name of the plot
- `.xlim` and `.ylim` Two vectors indicating the limits of the plot
- `.layout` A two-column matrix with the location of the vertices.
- `.edgelist` A two-column matrix, an edgelist.
- `.N` Integer. The number of vertices.
- `.M` Integer. The number of edges.

The children grob contains the following two objects:

- `background` a grob rectangle.
- `graph` a `gTree` that contains each vertex and each edge of the figure.

Mapping attributes with formulas

Several aesthetics can be mapped directly from graph attributes by passing a one-sided formula naming the attribute, instead of building the vector by hand. The mapping depends on the aesthetic:

- `vertex.color = ~ attr` colors vertices by the vertex attribute `attr`. Character/factor attributes are mapped to a categorical palette, numeric attributes to a continuous gradient, and logical attributes to two colors. When used this way, `print()`-ing the resulting plot also draws a matching legend: a categorical key for discrete attributes and a continuous color bar for continuous ones.
- `vertex.nsides = ~ attr` maps each unique value of `attr` to a distinct vertex shape (triangle, square, pentagon, ...).
- `vertex.size = ~ attr` and `edge.width = ~ attr` scale sizes/widths from a numeric vertex/edge attribute.
- `edge.color` uses a different, richer formula grammar based on `ego()` and `alter()` to mix the endpoints' colors; see [netplot-formulae](#).

For `vertex.nsides`, `vertex.size`, and `edge.width` the right-hand side of the formula is *evaluated* with the graph's attributes in scope, so besides bare names you can use expressions, e.g. `edge.width = ~ log1p(weight)` or `vertex.size = ~ degree ^ 2`.

For example, `nplot(x, vertex.color = ~ gender, vertex.size = ~ degree)` colors vertices by the gender attribute and sizes them by degree. The same attribute-mapping formulas also work in [set_vertex_gpar\(\)](#).

See Also

[nplot_base](#)

Examples

```

library(igraph)
library(netplot)
set.seed(1)
x <- sample_smallworld(1, 200, 5, 0.03)

plot(x) # ala igraph
nplot(x) # ala netplot

# Mapping aesthetics from vertex attributes using formulas
V(x)$grp <- sample(letters[1:3], vcount(x), replace = TRUE)
V(x)$deg <- degree(x)
nplot(
  x,
  vertex.color = ~ grp, # color by the categorical attribute
  vertex.nsides = ~ grp, # and give each group a distinct shape
  vertex.size = ~ deg # size by a numeric attribute
)

```

nplot_base

nplot using base graphics

Description

nplot using base graphics

Usage

```

nplot_base(
  x,
  layout = igraph::layout_nicely(x),
  vertex.size = igraph::degree(x, mode = "in"),
  bg.col = "transparent",
  vertex.nsides = 10,
  vertex.color = grDevices::hcl.colors(1),
  vertex.size.range = c(0.01, 0.03, 4),
  vertex.frame.color = grDevices::adjustcolor(vertex.color, red.f = 0.75, green.f = 0.75,
    blue.f = 0.75),
  vertex.rot = 0,
  vertex.frame.prop = 0.1,
  edge.width = NULL,
  edge.width.range = c(1, 2),
  edge.arrow.size = NULL,
  edge.color = NULL,
  edge.color.mix = 0.5,
  edge.color.alpha = c(0.1, 0.5),
  edge.curvature = pi/3,
  edge.line.lty = "solid",

```



```

    edge.line.breaks = 5,
    sample.edges = 1,
    skip.vertex = FALSE,
    skip.edges = FALSE,
    skip.arrows = skip.edges,
    add = FALSE,
    zero.margins = TRUE
)

```

Arguments

<code>x</code>	A graph. It supports networks stored as <code>igraph</code> , <code>network</code> , and <code>matrices</code> objects (see details).
<code>layout</code>	Numeric two-column matrix with the graph layout in x/y positions of the vertices.
<code>vertex.size</code>	Numeric vector of length <code>vcount(x)</code> . Absolute size of the vertex from 0 to 1. Can also be a one-sided formula (e.g. <code>~ degree</code>) naming a numeric vertex attribute to map sizes from (see "Mapping attributes with formulas").
<code>bg.col</code>	Color of the background.
<code>vertex.nsides</code>	Numeric vector of length <code>vcount(x)</code> . Number of sides of the vertex. E.g. three is a triangle, and 100 approximates a circle. Can also be a one-sided formula (e.g. <code>~ group</code>) naming a vertex attribute; each unique value is then mapped to a distinct shape (see "Mapping attributes with formulas").
<code>vertex.color</code>	Vector of length <code>vcount(x)</code> . Vertex HEX or built in colors. Can also be a one-sided formula (e.g. <code>~ group</code>) naming a vertex attribute to color vertices by (see "Mapping attributes with formulas").
<code>vertex.size.range</code>	Numeric vector of length 2 or 3, or <code>NULL</code> . Relative size for the minimum and maximum of the plot, and curvature of the scale. The third number is used as <code>size^rel[3]</code> . If <code>NULL</code> , scaling is suppressed and <code>vertex.size</code> is used as is.
<code>vertex.frame.color</code>	Vector of length <code>vcount(x)</code> . Border of vertex in HEX or built in colors.
<code>vertex.rot</code>	Vector of length <code>vcount(x)</code> in Radians. Passed to npolygon , elevation degree from which the polygon is drawn.
<code>vertex.frame.prop</code>	Vector of length <code>vcount(x)</code> . What proportion of the vertex does the frame occupy (values between 0 and 1).
<code>edge.width</code>	Numeric vector of length <code>ecount(x)</code> . Relative edge widths. Values are normalized and then mapped to the range specified by <code>edge.width.range</code> , unless <code>edge.width.range</code> is <code>NULL</code> . For <code>nplot.igraph</code> and <code>nplot.network</code> , defaults to the "weight" edge attribute if present; otherwise all edges use width 1. Can also be a one-sided formula (e.g. <code>~ weight</code>) naming a numeric edge attribute (see "Mapping attributes with formulas").
<code>edge.width.range</code>	Numeric vector of length 2, or <code>NULL</code> . The minimum and maximum line widths (in points) to use when mapping <code>edge.width</code> values. For example, <code>c(1, 4)</code>

	maps the smallest edge weight to 1pt and the largest to 4pt. If NULL, scaling is suppressed and <code>edge.width</code> is used as is.
<code>edge.arrow.size</code>	Vector of length <code>ecount(x)</code> from 0 to 1.
<code>edge.color</code>	A vector of length <code>ecount(x)</code> . In HEX or built in colors. Can be NULL in which case the color is picked as a mixture between <code>ego</code> and <code>alters'</code> <code>vertex.color</code> values.
<code>edge.color.mix</code>	Proportion of the mixing.
<code>edge.color.alpha</code>	Either a vector of length 1 or 2, or a matrix of size <code>ecount(x)*2</code> with values in <code>[0, 1]</code> . Alpha (transparency) levels (see details)
<code>edge.curvature</code>	Numeric vector of length <code>ecount(x)</code> . Curvature of edges in terms of radians.
<code>edge.line.lty</code>	Vector of length <code>ecount(x)</code> . Line types in R (e.g.- 1 = Solid, 2 = Dashed, etc).
<code>edge.line.breaks</code>	Vector of length <code>ecount(x)</code> . Number of vertices to draw (approximate) the arc (edge).
<code>sample.edges</code>	Numeric scalar between 0 and 1. Proportion of edges to sample.
<code>skip.vertex</code> , <code>skip.edges</code> , <code>skip.arrows</code>	Logical scalar. When TRUE the object is not plotted.
<code>add</code>	Logical scalar.
<code>zero.margins</code>	Logical scalar.

Value

`nplot_base` returns a list with the following components:

- `vertex.coords` A list of length `N` where each element describes the geomtry of each vertex.
- `vertex.color` A vector of colors
- `vertex.frame.coords` Similar to `vertex.coords`, but for the frame.
- `vertex.frame.color` Similar to `vertex.color`, but for the frame.
- `edge.color` Vector of functions used to compute the edge colors.
- `edge.coords` Similar to `vertex.coords`, the points that describe each edge.
- `edge.arrow.coords` A list of matrices describing the geometry of the tip of the edges.
- `edge.width` A numeric vector with edges' widths.
- `xlim, ylim` Limits of the plot area.

See Also

[nplot](#)

Examples

```
# Same example as in nplot
library(igraph)
library(netplot)
set.seed(1)
x <- sample_smallworld(1, 200, 5, 0.03)

nplot_base(x) # ala netplot (using base)
```

nplot_legend	<i>Add legend to a netplot object</i>
--------------	---------------------------------------

Description

Legends in [grid](#) graphics is a bit more complicated than in base graphics. The function `nplot_legend` is a wrapper of `grid::legendGrob()` that makes the process easier. Besides labels, the main visual arguments for the figure are passed through the `gp` argument (see examples).

Usage

```
nplot_legend(
  g,
  labels,
  pch,
  gp = grid::gpar(),
  ...,
  packgrob.args = list(side = "left")
)

## S3 method for class 'netplot_legend'
print(x, y = NULL, newpage = TRUE, ...)
```

Arguments

<code>g</code>	An object of class netplot .
<code>labels</code>	Character vector of labels.
<code>pch</code>	See graphics::points() .
<code>gp</code>	An object of class grid::gpar()
<code>...</code>	Further arguments passed to grid::legendGrob() .
<code>packgrob.args</code>	List of arguments passed to grid::packGrob() .
<code>x</code>	An object of class <code>netplot_legend</code> .
<code>y</code>	Ignored.
<code>newpage</code>	Logical scalar. When TRUE it calls grid::grid.newpage() .

Value

A frame grob.

Examples

```
library(igraph)
library(netplot)
set.seed(1)
x <- sample_smallworld(1, 200, 5, 0.03)
V(x)$nsides <- sample(c(10, 4), 200, replace = TRUE)

g <- nplot(
  x,
  vertex.nsides = V(x)$nsides,
  vertex.color = ifelse(V(x)$nsides == 4, "red", "steelblue"),
  edge.line.breaks = 5
)

nplot_legend(
  g,
  labels = c("circle", "diamond", "edge"),
  pch = c(21, 23, NA),
  gp = gpar(
    fill = c("steelblue", "red", NA),
    lwd = c(NA, NA, 1),
    col = c(NA, NA, "purple")
  )
)
grid.text("Legend to the left (default)", y = unit(.95, "npc"), just = "bottom")

nplot_legend(
  g,
  labels = c("circle", "diamond", "edge"),
  pch = c(21, 23, NA),
  gp = gpar(
    fill = c("steelblue", "red", NA),
    lwd = c(NA, NA, 1),
    col = c(NA, NA, "purple")
  ),
  # These two extra options set the legend to the bottom
  packgrob.args = list(side = "bottom"),
  ncol = 3
)
grid.text("Legend bottom", y = unit(.95, "npc"), just = "bottom")
```

Description

n-sided polygons Calculate the coordinates for an nsided polygon

Usage

```
npolygon(x = 0, y = 0, n = 6L, r = 1, d = 2 * pi/(n)/2)
```

Arguments

x, y	Numeric scalar. Origin of the polygon.
n	Integer scalar. Number of sides.
r	Numeric scalar. Radius of the polygon.
d	Numeric scalar. Starting degree in radians.

Value

A two column matrix with the coordinates to draw a n sided polygon.

Examples

```
graphics.off()
oldpar <- par(no.readonly = TRUE)

par(xpd = NA, mfrow = c(3, 3), mai = rep(0, 4))
for (n in c(2, 3, 4, 5, 6, 8, 12, 20, 50)) {

  plot.new()
  plot.window(c(-1.25, 1.25), c(-1.25, 1.25))

  for (i in seq(1, .0005, length.out = 200)) {
    col <- adjustcolor("tomato", alpha.f = i)
    polygon(npolygon(x=(i-1)/4, y = (i-1)/4, r = i, d = i-1, n = n),
            col = NA, border=col)
  }

  mtext(sprintf("n = %i", n), side = 1, line = -3)
}

par(oldpar)
```

piechart

A flexible piechart.

Description

While similar to `graphics::pie()`, this function is much more flexible as it allows providing different parameters for each slice of the pie. Furthermore, it allows adding the plot to the current device, making it possible to create compound piecharts.

Usage

```
piechart(
  x,
  labels = names(x),
  radius = 1,
  doughnut = 0,
  origin = c(0, 0),
  edges = 200,
  slice.off = 0,
  init.angle = 0,
  last.angle = 360,
  tick.len = 0.1,
  text.args = list(),
  segments.args = list(),
  skip.plot.slices = FALSE,
  add = FALSE,
  rescale = TRUE,
  ...
)
```

Arguments

<code>x</code>	Numeric vector. Values that specify the area of the slices.
<code>labels</code>	Character vector of length <code>length(x)</code> . Passed to <code>graphics::text()</code> .
<code>radius</code>	Numeric vector. Radius of each slice (can be a scalar).
<code>doughnut</code>	Numeric scalar. Radius of each inner circle (doughnut) (can be a scalar).
<code>origin</code>	Numeric vector of length 2. Coordinates of the origin.
<code>edges</code>	Numeric scalar. Smoothness of the slices curve (can be a vector).
<code>slice.off</code>	Numeric vector. When <code>!=0</code> , specifies how much to move the slice away from the origin. When scalar is recycled.
<code>init.angle</code>	Numeric scalar. Angle from where to start drawing in degrees.
<code>last.angle</code>	Numeric scalar. Angle where to finish drawing in degrees.
<code>tick.len</code>	Numeric scalar. Size of the tick marks as proportion of the radius.
<code>text.args</code>	List. Further arguments passed to <code>graphics::text()</code> .
<code>segments.args</code>	List. Further arguments passed to <code>graphics::segments()</code> when drawing the tickmarks.
<code>skip.plot.slices</code>	Logical scalar. When <code>FALSE</code> , slices are not drawn. This can be useful if, for example, the user only wants to draw the labels.
<code>add</code>	Logical scalar. When <code>TRUE</code> it is added to the current device.
<code>rescale</code>	Logical scalar. When <code>TRUE</code> (default), the y-coordinates of the polygons (slices), text and tickmarks will be rescaled such that the aspect ratio is preserved, i.e. looks like a circle.
<code>...</code>	Further arguments passed to <code>graphics::polygon()</code> (see details).

Details

The function is a wrapper of `graphics::polygon()`, so all parameters such as color, density, border, etc. are passed directly by `mapply()` so that are specified one per slice. The coordinates of the slices are computed internally.

Value

A list with the following elements:

<code>slices</code>	A list of length <code>length(x)</code> with the coordinates of each slice.
<code>textcoords</code>	A numeric matrix of size <code>length(x)*2</code> with coordinates where the labels can be put at.
<code>alpha0</code>	A numeric vector of size <code>length(x)</code> with the starting degree in radians of the slice.
<code>alpha1</code>	A numeric vector of size <code>length(x)</code> with the ending degree in radians of the slice.

See Also

<https://commons.wikimedia.org/wiki/File:Nightingale-mortality.jpg>

Examples

```
# Example 1 -----
# A set of 3 nested rings rings starting at 315 deg. and ending at 270 deg.

# Values to plot
vals <- c(1,2,3,10)

# Outer (includes labels)
piechart(vals, col=grDevices::blues9[5:8], border=NA, doughnut = .5,
  radius=.75, labels=vals, init.angle = 315, last.angle = 270)

# Middle
piechart(vals, col=grDevices::blues9[3:6], border=NA, doughnut = .3,
  radius=.5, add=TRUE, init.angle = 315, last.angle = 270)

# Inner
piechart(vals, col=grDevices::blues9[1:4], border="gray", doughnut = .1,
  radius=.3, add=TRUE, init.angle = 315, last.angle = 270)

# Example 2 -----
# Passing values to polygon and playing with the radius and slice.off

piechart(1:10, density=(1:10)^2/2, slice.off = (1:10)/30, doughnut = .5,
  radius = sqrt(10:1),
  # Here we are setting random labels...
  labels=sapply(1:10, function(x) paste(sample(letters, x, TRUE), collapse=""))
)
```

segments_gradient	<i>Draw segments colored by gradients</i>
-------------------	---

Description

Draw segments colored by gradients

Usage

```
segments_gradient(
  x,
  y = NULL,
  col = colorRamp2(c("transparent", "black"), TRUE),
  lend = 1,
  ...
)
```

Arguments

x, y	Coordinates passed to grDevices::xy.coords .
col	Color ramp function (see grDevices::colorRamp).
lend	Passed to graphics::segments .
...	Further arguments passed to segments.

Value

See [graphics::segments](#).

Examples

```
set.seed(1)
x <- cbind(cumsum(rnorm(1e3, sd=.1)), cumsum(rnorm(1e3, sd=.4)))
plot(x, type="n")
segments_gradient(x)
```

set_gpar	<i>Set/retrieve graphical parameters of a netplot object</i>
----------	--

Description

Set/retrieve graphical parameters of a netplot object

Usage

```

set_gpar(x, type, element, idx, ...)

set_edge_gpar(x, element, idx, ...)

set_vertex_gpar(x, element, idx, ...)

get_vertex_gpar(x, element, ..., idx)

get_edge_gpar(x, element, ..., idx)

get_gpar(x, type, element, ..., idx, simplify = TRUE)

```

Arguments

x	An object of class <code>netplot</code> .
type	Character. Either "edge" or "vertex".
element	Character. If "edge", then it can be either "line" or "arrow", otherwise it can be either "core" or "frame".
idx	(optional) Integer vector. Indices of the elements to be modified. When missing, all elements are modified.
...	Parameters to be modified/retrieved. This is passed to grid::editGrob via grid::gpar .
simplify	Logical. When TRUE it tries to simplify the result. Otherwise it returns a nested list.

Details

`set_edge_gpar` and `set_vertex_gpar` are shorthands for `set_gpar(type = "edge", ...)` and `set_gpar(type = "vertex", ...)` respectively.

`get_edge_gpar` and `get_vertex_gpar` are shorthands for `get_gpar(type = "edge", ...)` and `get_gpar(type = "vertex", ...)` respectively.

Value

An object of class `netplot` with modified parameters.

Examples

```

library(igraph)
library(netplot)

x <- make_ring(5)

g <- nplot(x)

# Updating edge color
g <- set_edge_gpar(g, col = "gray80")

```

```
# Retrieving the color of the vertices (core)
get_vertex_gpar(g, element = "core", "fill", "lwd")
```

Index

`alter (netplot-formulae)`, 7

`color_formula (netplot-formulae)`, 7

`colorkey`, 2

`colorRamp2`, 4

`ego (netplot-formulae)`, 7

`get_edge_gpar (set_gpar)`, 24

`get_gpar (set_gpar)`, 24

`get_vertex_gpar (set_gpar)`, 24

`graphics::pie()`, 21

`graphics::points()`, 19

`graphics::polygon()`, 22, 23

`graphics::segments`, 24

`graphics::segments()`, 22

`graphics::text`, 3

`graphics::text()`, 22

`grDevices::colorRamp`, 4, 24

`grDevices::xy.coords`, 24

`grid`, 19

`grid::editGrob`, 25

`grid::gpar`, 13, 25

`grid::gpar()`, 19

`grid::grid.locator()`, 5

`grid::grid.newpage`, 14

`grid::grid.newpage()`, 19

`grid::grob`, 15

`grid::legendGrob()`, 19

`grid::packGrob()`, 19

`igraph::graph_from_adjacency_matrix()`, 14

`locate_vertex`, 5

`make_colors`, 6

`make_edges_colors (make_colors)`, 6

`make_vertex_colors (make_colors)`, 6

`mapply()`, 23

`matrix`, 14

`netplot`, 7, 19

`netplot (nplot)`, 9

`netplot-formulae`, 7, 15

`nplot`, 6, 9, 18

`nplot()`, 7, 8

`nplot_base`, 15, 16

`nplot_legend`, 19

`nplot_legend()`, 14

`npolygon`, 13, 17, 20

`piechart`, 21

`print.netplot (nplot)`, 9

`print.netplot_legend (nplot_legend)`, 19

`segments_gradient`, 24

`set_edge_gpar (set_gpar)`, 24

`set_edge_gpar()`, 7, 8, 14

`set_gpar`, 6, 24

`set_vertex_gpar (set_gpar)`, 24

`set_vertex_gpar()`, 14, 15