Package ‘autograph’

August 22, 2025

Title Automatic Plotting of Many Graphs

Version 0.2.0

Date 2025-08-22

Description Visual exploration and presentation of networks should not be difficult.
This package includes functions for plotting networks and network-
related metrics with sensible and pretty defaults.
It includes 'ggplot2'-based plot methods for many popular network package classes.
It also includes some novel layout algorithms, and options for straightforward, consistent themes.

URL `https://stocnet.github.io/autograph/`

BugReports `https://github.com/stocnet/autograph/issues`

License MIT + file LICENSE

Language en-GB

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.6.0), manynet

Imports cli, dplyr (>= 1.1.0), ggdendro, ggraph, ggplot2, igraph,
tidygraph, tidyr

Suggests BiocManager, concaveman, gganimate, ggforce, gifski,
graphlayouts, methods, patchwork, testthat (>= 3.0.0)

Enhances Rgraphviz, RSiena

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown

Config/testthat/parallel true

Config/testthat/edition 3

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID:
<`https://orcid.org/0000-0002-8361-9647`>),
Henrique Sposito [ctb] (ORCID: <`https://orcid.org/0000-0003-3420-6085`>)

Maintainer James Hollway <james.hollway@graduateinstitute.ch>

Repository CRAN

Date/Publication 2025-08-22 19:50:02 UTC

Contents

`+.ggplot` *Add ggplot objects together*

Description

This function allows you to add ggplot objects together using the + operator. This is made possible by the `{patchwork}` package. This is useful for combining multiple plots into a single figure.

Usage

```
## S3 method for class 'ggplot'
e1 + e2, ...
```

Arguments

`e1, e2`	ggplot objects
`...`	Other arguments passed to `patchwork::wrap_plots()`.

ag_call *Consistent palette calls*

Description

These functions assist in calling particular parts of a theme's palette. For example, `ag_base()` will return the current theme's base or background color, and `ag_highlight()` will return the color used in that theme to highlight one or more nodes, lines, or such.

Usage

```
ag_base()

ag_highlight()

ag_positive()

ag_negative()

ag_qualitative(number)

ag_sequential(number)

ag_divergent(number)
```

Arguments

`number`	Integer of how many category colours to return.

Value

One or more hexcodes as strings.

`layout_configuration` *Layout algorithms based on configurational positions*

Description

Configurational layouts locate nodes at symmetric coordinates to help illustrate particular configurations. Currently "triad" and "quad" layouts are available. The "configuration" layout will choose the appropriate configurational layout automatically.

Usage

```
layout_configuration(.data, circular = FALSE, times = 1000)

layout_tbl_graph_configuration(.data, circular = FALSE, times = 1000)

layout_dyad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_dyad(.data, circular = FALSE, times = 1000)

layout_triad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_triad(.data, circular = FALSE, times = 1000)

layout_tetrad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_tetrad(.data, circular = FALSE, times = 1000)

layout_pentad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_pentad(.data, circular = FALSE, times = 1000)

layout_hexad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_hexad(.data, circular = FALSE, times = 1000)
```

Arguments

`.data`	Some `{manynet}` compatible network data.
`circular`	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
`times`	Maximum number of iterations, where appropriate

See Also

Other mapping: `layout_partition`, `map_graphr`, `map_graphs`, `map_grapht`

`layout_layered` *Layered layout*

Description

Layered layout

Usage

```
layout_tbl_graph_layered(.data, center = NULL, circular = FALSE, times = 4)
```

Arguments

`.data`	Some `{manynet}` compatible network data.
`center`, `circular`	Extra parameters required for `{tidygraph}` compatibility.
`times`	Integer of sweeps that the algorithm will pass through. By default 4.

Value

Returns a table of coordinates.

Examples

```
ties <- data.frame(
  from = c("A", "A", "B", "C", "D", "F", "F", "E"),
  to   = c("B", "C", "D", "E", "E", "E", "G", "G"),
  stringsAsFactors = FALSE)

coords <- layout_tbl_graph_layered(ties, times = 6)
coords
```

`layout_matching` *Matching layout*

Description

This layout works to position nodes opposite their matching nodes. See `manynet::to_matching()` for more details on the matching procedure.

Usage

```
layout_tbl_graph_matching(.data, center = NULL, circular = FALSE, times = 1)
```

Arguments

`.data`	Some `{manynet}` compatible network data.
`center`, `circular`, `times`	Extra parameters required for `{tidygraph}` compatibility.

Value

Returns a table of nodes' x and y coordinates.

`layout_partition` *Layout algorithms based on bi- or other partitions*

Description

These algorithms layout networks based on two or more partitions, and are recommended for use with `graphr()` or `{ggraph}`. Note that these layout algorithms use `{Rgraphviz}`, a package that is only available on Bioconductor. It will first need to be downloaded using `BiocManager::install("Rgraphviz")`. If it has not already been installed, there is a prompt the first time these functions are used though.

The "hierarchy" layout layers the first node set along the bottom, and the second node set along the top, sequenced and spaced as necessary to minimise edge overlap. The "alluvial" layout is similar to "hierarchy", but places successive layers horizontally rather than vertically. The "railway" layout is similar to "hierarchy", but nodes are aligned across the layers. The "ladder" layout is similar to "railway", but places successive layers horizontally rather than vertically. The "concentric" layout places a "hierarchy" layout around a circle, with successive layers appearing as concentric circles. The "multilevel" layout places successive layers as multiple levels. The "lineage" layout ranks nodes in Y axis according to values.

Usage

```
layout_concentric(
  .data,
  membership,
  radius = NULL,
  order.by = NULL,
  circular = FALSE,
  times = 1000
)

layout_tbl_graph_concentric(
  .data,
  membership,
  radius = NULL,
  order.by = NULL,
  circular = FALSE,
  times = 1000
)
```

```
layout_multilevel(.data, level, circular = FALSE)

layout_tbl_graph_multilevel(.data, level, circular = FALSE)

layout_lineage(.data, rank, circular = FALSE)

layout_tbl_graph_lineage(.data, rank, circular = FALSE)

layout_hierarchy(.data, center = NULL, circular = FALSE, times = 1000)

layout_tbl_graph_hierarchy(
  .data,
  center = NULL,
  circular = FALSE,
  times = 1000
)

layout_alluvial(.data, circular = FALSE, times = 1000)

layout_tbl_graph_alluvial(.data, circular = FALSE, times = 1000)

layout_railway(.data, circular = FALSE, times = 1000)

layout_tbl_graph_railway(.data, circular = FALSE, times = 1000)

layout_ladder(.data, circular = FALSE, times = 1000)

layout_tbl_graph_ladder(.data, circular = FALSE, times = 1000)
```

Arguments

`.data`	Some `{manynet}` compatible network data.
`membership`	A node attribute or a vector to draw concentric circles for "concentric" layout.
`radius`	A vector of radii at which the concentric circles should be located for "concentric" layout. By default this is equal placement around an empty centre, unless one (the core) is a single node, in which case this node occupies the centre of the graph.
`order.by`	An attribute label indicating the (decreasing) order for the nodes around the circles for "concentric" layout. By default ordering is given by a bipartite placement that reduces the number of edge crossings.
`circular`	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
`times`	Maximum number of iterations, where appropriate
`level`	A node attribute or a vector to hierarchically order levels for "multilevel" layout.
`rank`	A numerical node attribute to place nodes in Y axis according to values for "lineage" layout.

`center`	Further split "hierarchical" layouts by declaring the "center" argument as the "events", "actors", or by declaring a node name in hierarchy layout. Defaults to NULL.

Source

Diego Diez, Andrew P. Hutchins and Diego Miranda-Saavedra. 2014. "Systematic identification of transcriptional regulatory modules from protein-protein interaction networks". *Nucleic Acids Research*, 42 (1) e6.

See Also

Other mapping: `layout_configuration()`, `map_graphr`, `map_graphs`, `map_grapht`

Examples

```
#graphr(ison_southern_women, layout = "concentric", membership = "type",
#           node_color = "type", node_size = 3)
#graphr(ison_lotr, layout = "multilevel",
#           node_color = "Race", level = "Race", node_size = 3)
# ison_adolescents %>%
#   mutate(year = rep(c(1985, 1990, 1995, 2000), times = 2),
#          cut = node_is_cutpoint(ison_adolescents)) %>%
#   graphr(layout = "lineage", rank = "year", node_color = "cut",
#              node_size = migraph::node_degree(ison_adolescents)*10)
#graphr(ison_southern_women, layout = "hierarchy", center = "events",
#           node_color = "type", node_size = 3)
#graphr(ison_southern_women, layout = "alluvial")
```

`layout_valence` *Valence-based layout*

Description

Valence-based layout

Usage

```
layout_valence(
  .data,
  times = 500,
  center = NULL,
  circular = FALSE,
  repulsion_coef = 1,
  attraction_coef = 0.05
)

layout_tbl_graph_valence(
  .data,
  times = 500,
  center = NULL,
  circular = FALSE,
  repulsion_coef = 1,
  attraction_coef = 0.05
)
```

Arguments

`.data`	Some `{manynet}` compatible network data.
`times`	Integer of sweeps that the algorithm will pass through. By default 4.
`center, circular`	Extra parameters required for `{tidygraph}` compatibility.
`repulsion_coef`	Coefficient for global repulsion force. Default is 1.
`attraction_coef`	Coefficient for edge-based attraction/repulsion force. Default is 0.05.

Examples

```
edges <- data.frame(
  from = c("A", "B", "C", "D"),
  to   = c("B", "C", "D", "A"),
  weight = c(2, 3, 1, 4),
  sign = c(1, -1, 1, -1)  # 1 = positive, -1 = negative
  )
graphr(as_igraph(edges), layout="valence")
```

`made_earlier` *Precooked results for demonstrating plotting*

Description

These are all pre-cooked results objects, saved here to save time in testing and demonstrating how autograph plots look.

Usage

```
data(res_migraph_reg)

data(res_migraph_test)

data(res_migraph_diff)

data(res_manynet_diff)

data(res_siena_gof)
data(res_siena_influence)

data(res_siena_selection)

data(res_monan_traces)

data(res_monan_gof)
```

Format

An object of class `netlm` of length 15.

An object of class `network_test` of length 9.

An object of class `diffs_model` (inherits from `data.frame`) with 20 rows and 11 columns.

An object of class `diff_model` (inherits from `tbl_df`, `tbl`, `data.frame`) with 4 rows and 10 columns.

An object of class `sienaGOF` of length 1.

An object of class `influenceTable` (inherits from `data.frame`) with 25 rows and 4 columns.

An object of class `selectionTable` (inherits from `data.frame`) with 25 rows and 4 columns.

An object of class `traces.monan` of length 3.

An object of class `gof.stats.monan` of length 2.

`map_graphr` *Easily graph networks with sensible defaults*

Description

This function provides users with an easy way to graph (m)any network data for exploration, investigation, inspiration, and communication.

It builds upon `{ggplot2}` and `{ggraph}` to offer pretty and extensible graphing solutions. However, compared to those solutions, `graphr()` contains various algorithms to provide better looking graphs by default. This means that just passing the function some network data will often be sufficient to return a reasonable-looking graph.

The function also makes it easy to modify many of the most commonly adapted aspects of a graph, including node and edge size, colour, and shape, as arguments rather than additional functions that you need to remember. These can be defined outright, e.g. `node_size = 8`, or in reference to an attribute of the network, e.g. `node_size = "wealth"`.

Lastly, `graphr()` uses `{ggplot2}`-related theme information, so it is easy to make colour palette and fonts institution-specific and consistent. See e.g. `theme_iheid()` for more.

To learn more about what can be done visually, try `run_tute("Visualisation")`.

Usage

```
graphr(
  .data,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  node_group,
  edge_color,
  edge_size,
  snap = FALSE,
  ...,
  node_colour,
  edge_colour
)
```

Arguments

`.data`	A manynet-consistent object.
`layout`	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress" for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.
`labels`	Logical, whether to print node names as labels if present.
`node_color`, `node_colour`	Node variable to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be colored by declaring a color instead.
`node_shape`	Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.
`node_size`	Node variable to be used for sizing the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all node-related statistics prior

	to using this function. Nodes can also be sized by declaring a numeric size or vector instead.
`node_group`	Node variable to be used for grouping the nodes. It is easiest if this is added as a hull over groups before plotting. Group variables should have a minimum of 3 nodes, if less, number groups will be reduced by merging categories with lower counts into one called "other".
`edge_color`, `edge_colour`	Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.
`edge_size`	Tie variable to be used for sizing the edges. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.
`snap`	Logical scalar, whether the layout should be snapped to a grid.
`...`	Extra arguments to pass on to the layout algorithm, if necessary.

Value

A `ggplot2::ggplot()` object. The last plot can be saved to the file system using `ggplot2::ggsave()`.

See Also

Other mapping: `layout_configuration()`, `layout_partition`, `map_graphs`, `map_grapht`

Examples

```
graphr(ison_adolescents)
ison_adolescents %>%
  mutate(color = rep(c("introvert","extrovert"), times = 4),
         size = ifelse(node_is_cutpoint(ison_adolescents), 6, 3)) %>%
  mutate_ties(ecolor = rep(c("friends", "acquaintances"), times = 5)) %>%
  graphr(node_color = "color", node_size = "size",
         edge_size = 1.5, edge_color = "ecolor")
```

`map_graphs` *Easily graph a set of networks with sensible defaults*

Description

This function provides users with an easy way to graph lists of network data for comparison.

It builds upon this package's `graphr()` function, and inherits all the same features and arguments. See `graphr()` for more. However, it uses the `{patchwork}` package to plot the graphs side by side and, if necessary, in successive rows. This is useful for lists of networks that represent, for example, ego or component subgraphs of a network, or a list of a network's different types of tie or across time. By default just the first and last network will be plotted, but this can be overridden by the "waves" parameter.

Where the graphs are of the same network (same nodes), the graphs may share a layout to facilitate comparison. By default, successive graphs will use the layout calculated for the "first" network, but other options include the "last" layout, or a mix, "both", of them.

Usage

```
graphs(netlist, waves, based_on = c("first", "last", "both"), ...)
```

Arguments

`netlist`	A list of manynet-compatible networks.
`waves`	Numeric, the number of plots to be displayed side-by-side. If missing, the number of plots will be reduced to the first and last when there are more than four plots. This argument can also be passed a vector selecting the waves to plot.
`based_on`	Whether the layout of the joint plots should be based on the "first" or the "last" network, or "both".
`...`	Additional arguments passed to `graphr()`.

Value

Multiple `ggplot2::ggplot()` objects displayed side-by-side.

See Also

Other mapping: `layout_configuration()`, `layout_partition`, `map_graphr`, `map_grapht`

Examples

```
#graphs(to_egos(ison_adolescents))
#graphs(to_egos(ison_adolescents), waves = 8)
#graphs(to_egos(ison_adolescents), waves = c(2, 4, 6))
#graphs(play_diffusion(ison_adolescents))
```

`map_grapht` *Easily animate dynamic networks with sensible defaults*

Description

This function provides users with an easy way to graph dynamic network data for exploration and presentation.

It builds upon this package's `graphr()` function, and inherits all the same features and arguments. See `graphr()` for more. However, it uses the `{gganimate}` package to animate the changes between successive iterations of a network. This is useful for networks in which the ties and/or the node or tie attributes are changing.

A progress bar is shown if it takes some time to encoding all the .png files into a .gif.

Usage

```
grapht(
  tlist,
  keep_isolates = TRUE,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  edge_color,
  edge_size,
  ...,
  node_colour,
  edge_colour
)
```

Arguments

`tlist`	The same migraph-compatible network listed according to a time attribute, waves, or slices.
`keep_isolates`	Logical, whether to keep isolate nodes in the graph. TRUE by default. If FALSE, removes nodes from each frame they are isolated in.
`layout`	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress" for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.
`labels`	Logical, whether to print node names as labels if present.
`node_color, node_colour`	Node variable to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be colored by declaring a color instead.
`node_shape`	Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.
`node_size`	Node variable to be used for sizing the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an

existing variable, it is recommended to calculate all node-related statistics prior to using this function. Nodes can also be sized by declaring a numeric size or vector instead.

`edge_color`, `edge_colour`
: Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.

`edge_size`
: Tie variable to be used for sizing the edges. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.

`...`
: Extra arguments to pass on to the layout algorithm, if necessary.

Value

Shows a .gif image. Assigning the result of the function saves the gif to a temporary folder and the object holds the path to this file.

Source

https://blog.schochastics.net/posts/2021-09-15_animating-network-evolutions-with-gganimate/

See Also

Other mapping: `layout_configuration()`, `layout_partition`, `map_graphr`, `map_graphs`

Examples

```
#ison_adolescents %>%
#  mutate_ties(year = sample(1995:1998, 10, replace = TRUE)) %>%
#  to_waves(attribute = "year", cumulative = TRUE) %>%
#  grapht()
#ison_adolescents %>%
#  mutate(gender = rep(c("male", "female"), times = 4),
#         hair = rep(c("black", "brown"), times = 4),
#         age = sample(11:16, 8, replace = TRUE)) %>%
#  mutate_ties(year = sample(1995:1998, 10, replace = TRUE),
#              links = sample(c("friends", "not_friends"), 10, replace = TRUE),
#              weekly_meetings = sample(c(3, 5, 7), 10, replace = TRUE)) %>%
#  to_waves(attribute = "year") %>%
#  grapht(layout = "concentric", membership = "gender",
#             node_shape = "gender", node_color = "hair",
#             node_size =  "age", edge_color = "links",
#             edge_size = "weekly_meetings")
#grapht(play_diffusion(ison_adolescents, seeds = 5))
```

`map_measure` *Plotting logical marks Plotting numeric measures*

Description

These functions plot distributions for node, tie, and network measures, as defined in the `{manynet}` package.

Usage

```
## S3 method for class 'node_measure'
plot(x, type = c("h", "d"), ...)

## S3 method for class 'tie_measure'
plot(x, type = c("h", "d"), ...)

## S3 method for class 'network_measures'
plot(x, ...)
```

Arguments

`x`	An object of "node_measure", "tie_measure", or "network_measures" class.
`type`	For node and tie measures, whether the plot should be "h" a histogram or "d" a density plot. By default "h".
`...`	Other arguments to be passed on.

Value

`plot.node_measure()` and `plot.tie_measure()` returns a histogram and/or density plot of the distribution of the measure.

`plot.network_measures()` returns a plot of the measure traced over time.

Examples

```
plot(manynet::node_deg(ison_karateka))
plot(manynet::tie_betweenness(ison_karateka))
```

map_member *Plotting categorical memberships*

Description

This plotting method operates on "node_member" class objects from the {manynet} package, plotting the dendrogram of their membership.

Usage

```
## S3 method for class 'node_member'
plot(x, ...)

## S3 method for class 'matrix'
plot(x, ..., membership = NULL)
```

Arguments

x	An object of "node_member" class, for example as a result of running manynet::node_in_community().
...	Other arguments to be passed on.
membership	A "node_member" membership vector.

Value

plot.node_member() returns a dendrogram, with labels colored to indicate the different clusters, and with the optimal cutpoint shown by a dashed highlight line.

plot.matrix() returns a plot of an adjacency or incidency matrix, potentially with the rows and columns reordered to illustrate an additional membership vector.

Examples

```
plot(manynet::node_in_walktrap(ison_southern_women, "e"))
plot(as_matrix(ison_adolescents),
  membership = node_in_walktrap(ison_adolescents, "e"))
plot(as_matrix(ison_southern_women),
  membership = node_in_walktrap(ison_southern_women, "e"))
```

`map_motifs` *Plotting tabular motifs*

Description

These functions will plot graphs of the motifs used in a vector of results of e.g. a triad census.

Usage

```
## S3 method for class 'node_motif'
plot(x, ...)

## S3 method for class 'network_motif'
plot(x, ...)
```

Arguments

x	An object of "node_motif" class, e.g. resulting from a call to `manynet::node_by_triad()`.
...	Other arguments to be passed on.

Value

`plot.node_motif()` returns a set of graphs that illustrate the motifs mentioned in the results from a node_motif function in `{manynet}`.

`plot.network_motif()` returns a set of graphs that illustrate the motifs mentioned in the results from a net_motif function in `{manynet}`.

`map_scales` *Many scales*

Description

These functions enable to add color scales to be graphs.

Usage

```
scale_fill_iheid(direction = 1, ...)

scale_colour_iheid(direction = 1, ...)

scale_color_iheid(direction = 1, ...)

scale_edge_colour_iheid(direction = 1, ...)

scale_edge_color_iheid(direction = 1, ...)
```

```
scale_fill_centres(direction = 1, ...)

scale_colour_centres(direction = 1, ...)

scale_color_centres(direction = 1, ...)

scale_edge_colour_centres(direction = 1, ...)

scale_edge_color_centres(direction = 1, ...)

scale_fill_sdgs(direction = 1, ...)

scale_colour_sdgs(direction = 1, ...)

scale_color_sdgs(direction = 1, ...)

scale_edge_colour_sdgs(direction = 1, ...)

scale_edge_color_sdgs(direction = 1, ...)

scale_fill_ethz(direction = 1, ...)

scale_colour_ethz(direction = 1, ...)

scale_color_ethz(direction = 1, ...)

scale_edge_colour_ethz(direction = 1, ...)

scale_edge_color_ethz(direction = 1, ...)

scale_fill_uzh(direction = 1, ...)

scale_colour_uzh(direction = 1, ...)

scale_color_uzh(direction = 1, ...)

scale_edge_colour_uzh(direction = 1, ...)

scale_edge_color_uzh(direction = 1, ...)

scale_fill_rug(direction = 1, ...)

scale_colour_rug(direction = 1, ...)

scale_color_rug(direction = 1, ...)

scale_edge_colour_rug(direction = 1, ...)
```

```
scale_edge_color_rug(direction = 1, ...)
```

Arguments

`direction`	Direction for using palette colors.
`...`	Extra arguments passed to `ggplot2::discrete_scale()`.

Examples

```
#ison_brandes %>%
#mutate(core = migraph::node_is_core(ison_brandes)) %>%
#graphr(node_color = "core") +
#scale_color_iheid()
#graphr(ison_physicians[[1]], edge_color = "type") +
#scale_edge_color_ethz()
```

match_color *Matching colors across palettes*

Description

Sometimes particular colours are coded in certain ways to facilitate interpretation. For example, perhaps primary colours or traffic light colours are used to represent some discrete options. Yet institutional palettes vary in terms of which colours they have available. This function uses the Euclidean distance of colours in CIELAB space to those of a target palette to find the closes corresponding colours.

Usage

```
match_color(colors, pal)
```

Arguments

`colors`	One or more hexcodes to match with colors from the palette.
`pal`	Optionally, a vector of hexcodes representing a palette in which to find matches. By default, the current theme's qualitative palette is used.

Value

A vector of hexcodes the length of the first argument.

Examples

```
match_color("#4575b4")
```

`model_mrqap` *Plotting methods for MRQAP models*

Description

These plotting methods are for results obtained by fitting an MRQAP model. The S3 classes are "netlm" or "netlogit", and so are compatible with the results from either the `{sna}` or `{migraph}` packages.

Usage

```
## S3 method for class 'netlm'
plot(x, ...)

## S3 method for class 'netlogit'
plot(x, ...)
```

Arguments

`x`	An object obtained by fitting an MRQAP model to some data. For example, `migraph::net_regression()`.
`...`	Further arguments to be passed on to plot.

Value

A plot showing the location of observed statistics compared to the distribution of statistics from permuted networks.

Examples

```
# Here's something I cooked up with migraph earlier:
plot(res_migraph_reg)
```

`plot.diffusion` *Plotting diffusion models*

Description

Plotting diffusion models

Usage

```
## S3 method for class 'diff_model'
plot(x, ..., all_steps = TRUE)

## S3 method for class 'diffs_model'
plot(x, ...)

## S3 method for class 'learn_model'
plot(x, ...)
```

Arguments

`x`	A "diff_model" of "diffs_model" class of object. E.g. as a result from `manynet::play_diffusion()`.
`...`	Other arguments to be passed.
`all_steps`	Whether all steps should be plotted or just those where there is change in the distributions.

Value

`plot.diff_model()` returns a bar chart of the number of new infected nodes at each time point, as well as an overlay line plot of the total of infected

Examples

```
plot(res_manynet_diff)
plot(res_migraph_diff)
plot(play_learning(ison_networkers, beliefs = runif(net_nodes(ison_networkers))))
```

`plot.influenceTable` *Plotting influence tables*

Description

These are functions for constructing and presenting influence tables for the interpretation of results for network and behavior dynamics obtained with the RSiena or multiSiena packages.

Usage

```
## S3 method for class 'influenceTable'
plot(x, separation = 0, ...)
```

Arguments

`x`	An object of class "influenceTable", created using `RSiena::influenceTable()`.
`separation`	This can be used to make the curves visually distinguishable if they overlap too much without it. An advisable value then is, e.g., 0.01.
`...`	Other arguments to be passed.

Value

A plot showing how the influence evaluation function changes based on ego's value and alter's value of some covariate.

Author(s)

Tom Snijders

References

Consult also the RSiena manual, Sections 13.2 and 13.4. Gratitude to Steffen Triebel and Rene Veenstra for corrections.

Examples

```
plot(res_siena_influence)
```

`plot.network_test` *Plotting methods for CUG and QAP tests*

Description

These plotting methods are for results obtained by testing some statistic against those produced in a reference distribution of conditional uniform graphs or as a quadratic assignment procedure. The S3 class is "network_test".

Usage

```
## S3 method for class 'network_test'
plot(x, ..., threshold = 0.95, tails = c("two", "one"))
```

Arguments

`x`	An object obtained from a conditional uniform graph or quadratic assignment procedure test. For example, `migraph::test_permutation()`.
`...`	Other arguments to be passed on.
`threshold`	The empirical threshold to shade in the plot.
`tails`	By default "two" indicating a two-tailed test, but "one" for a one-tailed test is also available.

Value

A distribution of the simulated or permuted statistics, with 2.5% shaded at each end, and a line highlighting where the observed statistic lies on this distribution.

Examples

```
# Here's something I cooked up with migraph earlier:
plot(res_migraph_test)
```

`plot.selectionTable` *Plotting selection tables*

Description

These are functions for constructing and presenting selection tables for the interpretation of results for network dynamics obtained with the RSiena package.

Usage

```
## S3 method for class 'selectionTable'
plot(x, quad = TRUE, separation = 0, ...)
```

Arguments

`x`	An object of class "selectionTable", created using `RSiena::selectionTable()`.
`quad`	When TRUE (the default), a quadratic function (average and total alter) is plotted. Use `quad = FALSE` for similarity effects.
`separation`	This can be used to make the curves visually distinguishable if they overlap too much without it. An advisable value then is, e.g., 0.01.
`...`	Other arguments to be passed.

Value

A plot showing how the selection evaluation function changes based on ego's value and alter's value of some covariate.

Author(s)

Tom Snijders

References

Consult also the RSiena manual, Sections 13.1 and 13.3.

Examples

```
plot(res_siena_selection)
```

`plot.sienaGOF` *SIENA Goodness of Fit*

Description

This function plots goodness of fit objects created using RSiena. Unlike the plot method included in the `{RSiena}` package, this function utilises `{ggplot2}` and not `{lattice}`, which makes the output more compatible and themeable.

Usage

```
## S3 method for class 'sienaGOF'
plot(x, ...)
```

Arguments

`x`	A sienaGOF object, as returned by `RSiena::sienaGOF()`.
`...`	Other parameters to be passed to the plotting funciton, for example `main = "Title"` for a different title than the default.

Value

A violin plot showing the distribution of statistics from the simulations and a line highlighting the observed statistics.

Examples

```
plot(res_siena_gof)
```

`plot_monan_gof` *plot.gof.stats.monan*

Description

plot.gof.stats.monan

Usage

```
## S3 method for class 'gof.stats.monan'
plot(x, lvls, ...)
```

Arguments

`x`	An object of class "gof.stats.monan".
`lvls`	The values for which the gofFunction should be calculated/plotted.
`...`	Additional plotting parameters, use discouraged.

Value

The function `plot.gof.stats.monan` returns violin plots of the gof tests with observed values superimposed in red.

Examples

```
plot(res_monan_gof, lvls = 1:15)
```

`plot_monan_trace` *plot.traces.monan*

Description

plot.traces.monan

Usage

```
## S3 method for class 'traces.monan'
plot(x, ...)
```

Arguments

`x`	An object of class "traces.monan".
`...`	Additional plotting parameters, use not recommended.

Value

The function `plot.traces.monan` shows a scatter plot of the statistics of simulated networks from phase three of the esimtation.

Examples

```
plot(res_monan_traces)
```

theme_scales *Many themes*

Description

These functions enable graphs to be easily and quickly themed, e.g. changing the default colour of the graph's vertices and edges.

Usage

```
theme_iheid(base_size = 12, base_family = "serif")

theme_ethz(base_size = 12, base_family = "sans")

theme_uzh(base_size = 12, base_family = "sans")

theme_rug(base_size = 12, base_family = "mono")
```

Arguments

`base_size`	Font size, by default 12.
`base_family`	Font family, by default "sans".

Examples

```
to_mentoring(ison_brandes) %>%
  mutate(color = c(rep(c(1,2,3), 3), 3)) %>%
  graphr(node_color = "color") +
  labs(title = "Who leads and who follows?") +
  scale_color_iheid() +
  theme_iheid()
```

theme_set *Many themes*

Description

This function enables all plots to be quickly, easily and consistently themed. This is achieved by setting a theme option that enables the appropriate palette to be used for all autograph-consistent plotting methods.

The following themes are currently available: default, bw, iheid, ethz, uzh, rug, unibe, crisp, neon, rainbow.

Usage

```
stocnet_theme(theme = NULL)
```

Arguments

`theme`	String naming a theme. By default "default". This string can be capitalised or not.

Value

This function sets the theme and palette(s) to be used across all stocnet packages. The palettes are written to options and held there.

Examples

```
stocnet_theme("default")
plot(manynet::node_degree(ison_karateka))
stocnet_theme("rug")
plot(manynet::node_degree(ison_karateka))
```

Index