Package 'graph4lg'

July 22, 2020

Type Package

Title Build Graphs for Landscape Genetics Analysis

Version 1.0.0

Author Paul Savary

Maintainer Paul Savary <`savarypaul660@gmail.com`>

Description Build graphs for landscape genetics analysis. This set of
functions can be used to import and convert spatial and genetic data
initially in different formats, import landscape graphs created with
'GRAPHAB' software (Foltete et al., 2012) <doi:10.1016/j.envsoft.2012.07.002>,
make diagnosis plots of isolation by distance relationships in order to
choose how to build genetic graphs, create graphs with a large range of
pruning methods, weight their links with several genetic distances, plot
and analyse graphs,compare them with other graphs. It uses functions from
other packages such as 'adegenet'
(Jombart, 2008) <doi:10.1093/bioinformatics/btn129> and 'igraph' (Csardi
et Nepusz, 2006) <https://bit.ly/35a3V3H>. It also implements methods
commonly used in landscape genetics to create graphs, described by Dyer et
Nason (2004) <doi:10.1111/j.1365-294X.2004.02177.x> and Greenbaum et
Fefferman (2017) <doi:10.1111/mec.14059>, and to analyse distance data
(van Strien et al., 2015) <doi:10.1038/hdy.2014.62>.

Depends R(>= 3.1.0)

License GPL-2

Encoding UTF-8

LazyData true

Imports stringr, adegenet, stats, spatstat, Matrix, vegan, utils,
methods, pegas, MASS, igraph, ggplot2, tidyr, sp, sf,
diveRsity, rappdirs, gdistance, raster, foreign, ecodist

RdMacros Rdpack

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, Rdpack

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-07-22 07:10:02 UTC

R topics documented:

`add_nodes_attr` *Add attributes to the nodes of a graph*

Description

The function adds attributes to the nodes of a graph from either an object of class `data.frame` or from a shapefile layer. The node IDs in the input objects must be the same as in the graph object.

Usage

```
add_nodes_attr(
  graph,
  input = "df",
  data,
  dir_path = NULL,
  layer = NULL,
  index = "Id",
  include = "all"
)
```

Arguments

`graph`	A graph object of class `igraph`.
`input`	A character string indicating the nature of the input data from which come the attributes to add to the nodes. • If 'input = "shp"', then attributes come from the attribute table of a shapefile layer of type point. • If 'input = "df"', then attributes come from an object of class `data.frame` In both cases, input attribute table or dataframe must have a column with the exact same values as the node IDs.
`data`	(only if 'input = "df"') The name of the object of class `data.frame` with the attributes to add to the nodes.
`dir_path`	(only if 'input = "shp"') The path (character string) to the directory containing the shapefile layer of type point whose attribute table contains the attributes to add to the nodes.

`layer`	(only if 'input = "shp"') The name (character string) of the shapefile layer of type point (without extension, ex.: "nodes" refers to "nodes.shp" layer) whose attribute table contains the attributes to add to the nodes.
`index`	The name (character string) of the column with the nodes names in the input data (column of the attribute table or of the dataframe).
`include`	A character string (vector) indicating which columns of the input data will be added as nodes' attributes. By default, 'include = "all"', i.e. every column of the input data is added. Alternatively, 'include' can be a vector with the names of the columns to add (ex.: "c('x', 'y', 'pop_name')").

Details

The graph can be created with the function `graphab_to_igraph` by importing output from Graphab projects. Values of the metrics computed at the node level with Graphab can then be added to such a graph with this function.

Value

A graph object of class `igraph`

Author(s)

P. Savary

Examples

```
data("data_tuto")
graph <- data_tuto[[3]]
df_nodes <- data.frame(Id = igraph::V(graph)$name,
                       Area = runif(50, min = 10, max = 60))
graph <- add_nodes_attr(graph,
                        data = df_nodes,
                        input = "df",
                        index = "Id",
                        include = "Area")
```

`compute_graph_modul` *Compute modules from a graph by maximising modularity*

Description

The function computes modules from a graph by maximising modularity.

Usage

```
compute_graph_modul(
  graph,
  algo = "fast_greedy",
  node_inter = NULL,
  nb_modul = NULL
)
```

Arguments

`graph`	An object of class `igraph`. Its nodes must have names.
`algo`	A character string indicating the algorithm used to create the modules with **igraph**. • If `algo = 'fast_greedy'` (default), function `cluster_fast_greedy` from **igraph** is used (Clauset et al., 2004). • If `algo = 'walktrap'`, function `cluster_walktrap` from **igraph** is used (Pons et Latapy, 2006) with 4 steps (default options). • If `algo = 'louvain'`, function `cluster_louvain` from **igraph** is used (Blondel et al., 2008). In that case, the number of modules created in each graph is imposed. • If `algo = 'optimal'`, function `cluster_optimal` from **igraph** is used (Brandes et al., 2008) (can be very long). In that case, the number of modules created in each graph is imposed.
`node_inter`	(optional, default = NULL) A character string indicating whether the links of the graph are weighted by distances or by similarity indices. It is only used to compute the modularity index. It can be: • 'distance': Link weights correspond to distances. Nodes that are close to each other will more likely be in the same module. • 'similarity': Link weights correspond to similarity indices. Nodes that are similar to each other will more likely be in the same module. Inverse link weights are then used to compute the modularity index.
`nb_modul`	(optional , default = NULL) A numeric or integer value indicating the number of modules in the graph. When this number is not specified, the optimal value is retained.

Value

A `data.frame` with the node names and the corresponding module ID.

Author(s)

P. Savary

Examples

```
data("data_tuto")
mat_gen <- data_tuto[[1]]
```

```
graph <- gen_graph_thr(mat_w = mat_gen, mat_thr = mat_gen,
                            thr = 0.8)
res_mod <- compute_graph_modul(graph = graph,
                                algo = "fast_greedy",
                                node_inter = "distance")
```

---

compute_node_metric *Compute graph-theoretic metrics from a graph at the node level*

Description

The function computes graph-theoretic metric values at the node level.

Usage

```
compute_node_metric(
  graph,
  metrics = c("deg", "close", "btw", "str", "siw", "miw"),
  weight = TRUE
)
```

Arguments

`graph`	An object of class `igraph`. Its nodes must have names.
`metrics`	Character vector specifying the graph-theoretic metrics computed at the node-level in the graphs Graph-theoretic metrics can be: • Degree (`metrics = c("deg",...)`) • Closeness centrality index (`metrics = c("close",...)`) • Betweenness centrality index (`metrics = c("btw",...)`) • Strength (sum of the weights of the links connected to a node) (`metrics = c("str",...)`) • Sum of the inverse weights of the links connected to a node (`metrics = c("siw",...)`, default) • Mean of the inverse weights of the links connected to a node (`metrics = c("miw",...)`) By default, the vector `metrics` includes all these metrics.
`weight`	Logical which indicates whether the links are weighted during the calculation of the centrality indices betweenness and closeness. (default: `weight = TRUE`). Link weights are interpreted as distances when computing the shortest paths. They should then be inversely proportional to the strength of the relationship between nodes (e.g. to fluxes).

Value

A `data.frame` with the node names and the metrics computed.

Author(s)

P. Savary

Examples

```
data(data_ex_genind)
mat_gen <- mat_gen_dist(x = data_ex_genind, dist = "DPS")
graph <- gen_graph_thr(mat_w = mat_gen, mat_thr = mat_gen,
                            thr = 0.8)
res_met <- compute_node_metric(graph)
```

---

convert_cd — *Fit a model to convert cost-distances into Euclidean distances*

Description

The function fits a model to convert cost-distances into Euclidean distances as implemented in Graphab software.

Usage

```
convert_cd(
  mat_euc,
  mat_ld,
  to_convert,
  method = "log-log",
  fig = TRUE,
  line_col = "black",
  pts_col = "#999999"
)
```

Arguments

`mat_euc`	A symmetric `matrix` or `dist` object with pairwise geographical Euclidean distances between populations or sample sites. It will be the explanatory variable, and only values from the off diagonal lower triangle will be used.
`mat_ld`	A symmetric `matrix` or `dist` object with pairwise landscape distances between populations or sample sites. These distances can be cost-distances or resistance distances, among others. It will be the explained variable, and only values from the off diagonal lower triangle will be used.
`to_convert`	A numeric value or numeric vector with Euclidean distances to convert into cost-distances.
`method`	A character string indicating the method used to fit the model. • If 'method = "log-log"' (default), then the model takes the following form : log(ld) ~ A + B * log(euc)

- If 'method = "lm"', then the model takes the following form : ld ~ A + B * euc

`fig`	Logical (default = TRUE) indicating whether a figure is plotted
`line_col`	(if 'fig = TRUE') Character string indicating the color used to plot the line (default: "blue"). It must be a hexadecimal color code or a color used by default in R.
`pts_col`	(if 'fig = TRUE') Character string indicating the color used to plot the points (default: "#999999"). It must be a hexadecimal color code or a color used by default in R.

Details

IDs in 'mat_euc' and 'mat_ld' must be the same and refer to the same sampling site or populations, and both matrices must be ordered in the same way. Matrix of Euclidean distance 'mat_euc' can be computed using the function `mat_geo_dist`. Matrix of landscape distance 'mat_ld' can be computed using the function `mat_cost_dist`. Before the log calculation, 0 distance values are converted into 1, so that they are 0 after this calculation.

Value

A list of output (converted values, estimated parameters, R2) and optionally a ggplot2 object to plot

Author(s)

P. Savary

References

FoltÃªte J, Clauzel C, Vuidel G (2012). "A software tool dedicated to the modelling of landscape networks." *Environmental Modelling \& Software*, **38**, 316–327.

Examples

```
data("data_tuto")
mat_ld <- data_tuto[[2]][1:10, 1:10] * 1000
mat_euc <- data_tuto[[1]][1:10, 1:10] * 50000
to_convert <- c(30000, 40000)
res <- convert_cd(mat_euc = mat_euc,
                  mat_ld = mat_ld,
                  to_convert = to_convert, fig = FALSE)
```

`data_ex_genind` *data_ex_genind genetic dataset*

Description

Genetic dataset from genetic simulation on CDPOP 200 individuals, 10 populations 20 microsatellite loci (3 digits coding) 100 generations simulated

Usage

```
data_ex_genind
```

Format

An object of type 'genind'

Details

The simulation was made with CDPOP during 100 generations. Dispersal was possible between the 10 populations. Its probability depended on the cost distance between populations, calculated on a simulated resistance surface (raster). Mutations were not possible. There were initially 600 alleles in total (many disappeared because of drift). Population stayed constant with a sex-ratio of 1. Generations did not overlap. This simulation includes a part of stochasticity and these data result from only 1 simulation run.

References

Landguth EL, Cushman S (2010). "CDPOP: a spatially explicit cost distance population genetics program." *Molecular Ecology Resources*, **10**(1), 156–161.

Examples

```
data("data_ex_genind")
length(unique(data_ex_genind@pop))
```

`data_ex_gstud` *data_ex_gstud genetic dataset*

Description

Genetic dataset from genetic simulation on CDPOP 200 individuals, 10 populations 20 microsatellite loci (3 digits coding) 100 generations simulated

Usage

```
data_ex_gstud
```

Format

A 'data.frame' with columns:

ID Individual ID

POP Population name

LOCI-1 to LOCI-20 20 loci columns with microsatellite data with 3 digits coding, alleles separated by ":", and blank missing data (class 'locus' from **gstudio**)

Examples

```
data("data_ex_gstud")
str(data_ex_gstud)
length(unique(data_ex_gstud$POP))
```

`data_ex_loci` *data_ex_loci genetic dataset*

Description

Genetic dataset from genetic simulation on CDPOP 200 individuals, 10 populations 20 microsatellite loci (3 digits coding) 100 generations simulated

Usage

```
data_ex_loci
```

Format

An object of class 'loci' and 'data.frame' with the columns :

population Population name

Other columns 20 loci columns with microsatellite data with 3 digits coding, alleles separated by "/", and missing data noted "NA/NA"

Row names correspond to individuals' ID

Examples

```
data("data_ex_loci")
length(unique(data_ex_loci$population))
```

`data_simul_genind` *data_simul_genind genetic dataset*

Description

Genetic dataset from genetic simulation on CDPOP 1500 individuals, 50 populations 20 microsatellite loci (3 digits coding) 50 generations simulated

Usage

```
data_simul_genind
```

Format

An object of type 'genind'

Details

The simulation was made with CDPOP during 50 generations. Dispersal was possible between the 50 populations. Its probability depended on the cost distance between populations, calculated on a simulated resistance surface (raster). Mutations were not possible. There were initially 600 alleles in total (many disappeared because of drift). Population stayed constant with a sex-ratio of 1. Generations did not overlap. This simulation includes a part of stochasticity and these data result from only 1 simulation run.

References

Landguth EL, Cushman S (2010). "CDPOP: a spatially explicit cost distance population genetics program." *Molecular Ecology Resources*, **10**(1), 156–161.

Examples

```
data("data_simul_genind")
length(unique(data_simul_genind@pop))
```

`data_tuto` *data_tuto : data used to generate the vignette*

Description

Data used to generate the vignette

Data used to generate the vignette

Usage

```
data_tuto

data_tuto
```

Format

Several outputs or inputs to show how the package works in a list

mat_dps Genetic distance matrix example

mat_pg Second genetic distance matrix example

graph_ci Genetic independence graph example

dmc Output of the function 'dist_max_corr'

land_graph Landscape graph example

mat_ld Landscape distance matrix example

Several outputs or inputs to show how the package works in a list

dmc Output of the function 'dist_max_corr'

graph_ci Genetic independence graph example

mat_dps Genetic distance matrix example

mat_pg Second genetic distance matrix example

Examples

```
data("data_tuto")
mat_dps <- data_tuto[[1]]
str(mat_dps)
data("data_tuto")
mat_dps <- data_tuto[[1]]
str(mat_dps)
```

---

df_to_pw_mat *Convert an edge-list data.frame into a pairwise matrix*

Description

The function converts an edge-list data.frame into a symmetric pairwise matrix

Usage

```
df_to_pw_mat(data, from, to, value)
```

Arguments

`data`	An object of class `data.frame`
`from`	A character string indicating the name of the column with the ID of the origins
`to`	A character string indicating the name of the column with the ID of the arrivals
`value`	A character string indicating the name of the column with the values corresponding to each pair

Details

The matrix is a symmetric matrix. Be careful, you shall not provide a data.frame with different values corresponding to the pair 1-2 and 2-1 as an example. Ideally, for a complete matrix, data should have n(n-1)/2 rows if values are computed between n objects.

Value

A pairwise matrix

Author(s)

P. Savary

Examples

```
data(pts_pop_simul)
suppressWarnings(mat_geo <- mat_geo_dist(pts_pop_simul,
                  ID = "ID",
                  x = "x",
                 y = "y"))
g <- gen_graph_topo(mat_w = mat_geo,
                    mat_topo = mat_geo,
                    topo = "comp")
df <- data.frame(igraph::as_edgelist(g))
df$w <- igraph::E(g)$weight
df_to_pw_mat(df, from = "X1", to = "X2", value = "w")
```

`dist_max_corr` *Compute the distance at which the correlation between genetic distance and landscape distance is maximal*

Description

The function enables to compute the distance at which the correlation between genetic distance and landscape distance is maximal, using a method similar to that employed by van Strien et al. (2015). Iteratively, distance threshold values are tested. For each value, all the population pairs separated by a landscape distance larger than the threshold are removed before the Mantel correlation coefficient between genetic distance and landscape distance is computed. The distance threshold at which the correlation is the strongest is then identified. A figure showing the evolution of the correlation coefficients when landscape distance threshold increases is plotted.

Usage

```
dist_max_corr(
  mat_gd,
  mat_ld,
  interv,
  from = NULL,
  to = NULL,
  fig = TRUE,
  thr_gd = NULL,
  line_col = "black",
  pts_col = "#999999"
)
```

Arguments

`mat_gd`	A symmetric `matrix` or `dist` object with pairwise genetic distances between populations or sample sites.
`mat_ld`	A symmetric `matrix` or `dist` object with pairwise landscape distances between populations or sample sites. These distances can be Euclidean distances, cost-distances or resistance distances, among others.
`interv`	A numeric or integer value indicating the interval between the different distance thresholds for which the correlation coefficients are computed.
`from`	(optional) The minimum distance threshold value at which the correlation coefficient is computed.
`to`	(optional) The maximum distance threshold value at which the correlation coefficient is computed.
`fig`	Logical (default = TRUE) indicating whether a figure is plotted.
`thr_gd`	(optional) A numeric or integer value used to remove genetic distance values from the data before the calculation. All genetic distances values above 'thr_gd' are removed from the data. This parameter can be used especially when there are outliers.
`line_col`	(optional, if fig = TRUE) A character string indicating the color used to plot the line (default: "blue"). It must be a hexadecimal color code or a color used by default in R.
`pts_col`	(optional, if fig = TRUE) A character string indicating the color used to plot the points (default: "#999999"). It must be a hexadecimal color code or a color used by default in R.

Details

IDs in 'mat_gd' and 'mat_ld' must be the same and refer to the same sampling sites or populations, and both matrices must be ordered in the same way. The correlation coefficient between genetic distance and landscape distance computed is a Mantel correlation coefficient. If there are less than 50 pairwise values, the correlation is not computed, as in van Strien et al. (2015). Such a method can be subject to criticism from a strict statistical point of view given correlation coefficients computed from samples of different size are compared. The matrix of genetic distance 'mat_gd' can be computed using `mat_gen_dist`. The matrix of landscape distance 'mat_ld' can be computed using `mat_geo_dist` when the landscape distance needed is a Euclidean geographical distance. Mantel correlation coefficients are computed using the function `mantel`.

Value

A list of objects:

- The distance at which the correlation is the highest.
- The vector of correlation coefficients at the different distance thresholds
- The vector of the different distance thresholds
- A ggplot2 object to plot

Author(s)

P. Savary

References

Van Strien MJ, Holderegger R, Van Heck HJ (2015). "Isolation-by-distance in landscapes: considerations for landscape genetics." *Heredity*, **114**(1), 27.

Examples

```
data("data_tuto")
mat_gen <- data_tuto[[1]]
mat_dist <- data_tuto[[2]]*1000
res_dmc <- dist_max_corr(mat_gd = mat_gen,
                         mat_ld = mat_dist,
                         from = 32000, to = 42000,
                         interv = 5000,
                         fig = FALSE)
```

`genepop_to_genind` *Convert a GENEPOP file into a genind object*

Description

The function converts a text file in the format used by GENEPOP software into a genind object

Usage

```
genepop_to_genind(path, n.loci, pop_names = NULL, allele.digit.coding = 3)
```

Arguments

`path`	A character string with the path leading to the GENEPOP file in format .txt, or alternatively the name of this file in the working directory.
`n.loci`	The number of loci in the GENEPOP file (integer or numeric).
`pop_names`	(optional) Populations' names in the same order as in the GENEPOP file. Vector object (class character) of the same length as the number of populations. Without this parameter, populations are numbered from 1 to the number of populations.
`allele.digit.coding`	Number indicating whether alleles are coded with 3 (default) or 2 digits.

Details

This function uses functions from **pegas** package. GENEPOP file should can include microsatellites loci or SNPs with allele names of length 2 or 3 (noted as 01, 02, 03 or 04 for SNPs). The loci line(s) must not start with a spacing.

Value

An object of type `genind`.

Author(s)

P. Savary

References

Raymond M (1995). "GENEPOP: Population genetics software for exact tests and ecumenism. Vers. 1.2." *Journal of Heredity*, **86**, 248–249.

See Also

For more details about GENEPOP file formatting : `http://genepop.curtin.edu.au/help_input.html#Input` For the opposite conversion, see `genind_to_genepop`. The output file can be used to compute pairwise FST matrix with `mat_pw_fst`

Examples

```
path_in <- system.file('extdata', 'gpop_simul_10_g100_04_20.txt',
                       package = 'graph4lg')
file_n <- file.path(tempdir(), "gpop_simul_10_g100_04_20.txt")
file.copy(path_in, file_n, overwrite = TRUE)
genepop_to_genind(path = file_n, n.loci = 20,
                  pop_names = as.character(order(as.character(1:10))))
file.remove(file_n)
```

`genind_to_genepop` *Convert a genind object into a GENEPOP file*

Description

The function converts an object of class `genind` into a GENEPOP file. It then allows to use the functionalities of the GENEPOP software and its derived package **GENEPOP** on R, as well as some functions from other packages (differentiation test, F-stats calculations, HWE test,...). It is designed to be used with diploid microsatellite data with alleles coded with 2 or 3 digits or SNPs genind objects.

Usage

```
genind_to_genepop(x, output = "data.frame")
```

Arguments

`x`	An object of class `genind` from package **adegenet**.
`output`	A character string indicating the option used to select what the function will return: • If `output = "data.frame"`(default), then the function will return an object 'x' of class `data.frame` ready to be saved as a text file with the following command: `write.table(x,file = "file_name.txt",quote=FALSE,row.names=FALSE,col.nam` • If `output = "path_to_file/file_name.txt"`, then the function will write a text file named 'file_name.txt' in the directory corresponding to 'path_to_file'. Without 'path_to_file', the text file is written in the current working directory. The text file has the format required by GENEPOP software.

Value

An object of type `data.frame` if `ouput = "data.frame"`. If `output` is the path and/or the file name of a text file, then nothing is returned in R environment but a text file is created with the specified file name, either in the current working directory or in the specified folder.

Warning

Confusion: Do not confound this function with `genind2genpop` from **adegenet**. The latter converts an object of class `genind` into an object of class `genpop`, whereas `genind_to_genepop` converts an object of class `genind` into a text file compatible with GENEPOP software (Rousset, 2008).

Allele coding: This function can handle genetic data with different allele coding: 2 or 3 digit coding for microsatellite data or 2 digit coding for SNPs (A,C,T,G become respectively 01, 02, 03, 04).

Individuals order: When individuals in input data are not ordered by populations, individuals from the same population can be separated by individuals from other populations. It can be problematic when calculating then pairwise distance matrices. Therefore, in such a case, individuals are ordered by populations and populations ordered in alphabetic order.

Author(s)

P. Savary

References

Raymond M (1995). "GENEPOP: Population genetics software for exact tests and ecumenism. Vers. 1.2." *Journal of Heredity*, **86**, 248–249.

See Also

For more details about GENEPOP file formatting : http://genepop.curtin.edu.au/help_input.html#Input. For the opposite conversion, see genepop_to_genind. The output file can be used to compute pairwise FST matrix with mat_pw_fst

Examples

```
data(data_ex_genind)
x <- data_ex_genind
df_genepop <- suppressWarnings(genind_to_genepop(x,
                                            output = "data.frame"))
```

---

gen_graph_indep   *Create an independence graph of genetic differentiation from genetic data of class genind*

Description

The function allows to create genetic graphs from genetic data by applying the conditional independence principle. Populations whose allelic frequencies covary significantly once the covariance with the other populations has been taken into account are linked on the graphs.

Usage

```
gen_graph_indep(
  x,
  dist = "basic",
  cov = "sq",
  pcor = "magwene",
  alpha = 0.05,
  test = "EED",
  adj = "none",
  output = "igraph"
)
```

Arguments

`x`	An object of class `genind` that contains the multilocus genotype (format 'locus') of the individuals as well as their population and their geographical coordinates.
`dist`	A character string indicating the method used to compute the multilocus genetic distance between populations • If 'dist = 'basic" (default), then the multilocus genetic distance is computed using a Euclidean genetic distance formula (Excoffier et al., 1992) • If 'dist = 'weight", then the multilocus genetic distance is computed as in Fortuna et al. (2009). It is a Euclidean genetic distance giving more weight to rare alleles • If 'dist = 'PG", then the multilocus genetic distance is computed as in popgraph::popgraph function, following several steps of PCA and SVD (Dyer et Nason, 2004). • If 'dist = 'PCA", then the genetic distance is computed following a PCA of the matrix of allelic frequencies by population. It is a Euclidean genetic distance between populations in the multidimensional space defined by all the independent principal components.
`cov`	A character string indicating the formula used to compute the covariance matrix from the distance matrix • If 'cov = 'sq" (default), then the covariance matrix is calculated from the matrix of squared distances as in Everitt et Hothorn (2011) • If 'cov = 'dist", then the covariance matrix is calculated from the matrix of distances as in Dyer et Nason (2004) and popgraph function
`pcor`	A character string indicating the way the partial correlation matrix is computed from the covariance matrix. • If 'pcor = 'magwene", the steps followed are the same as in Magwene (2001) and in popgraph::popgraph function. It is the recommended option as it meets mathematical requirements. • If 'pcor = 'other", the steps followed are the same as used by Fortuna et al. (2009). They are not consistent with the approach of Magwene (2001).
`alpha`	A numeric value corresponding to the statistical tolerance threshold used to test the difference from 0 of the partial correlation coefficients. By default, 'alpha=0.05'.
`test`	A character string indicating the method used to test the significance of the partial correlation coefficients. • If 'test = 'EED" (default), then the Edge Exclusion Deviance criterion is used (Whittaker, 2009). Although other methods exist, this is the most common and thus the only one implemented here.
`adj`	A character string indicating the way of adjusting p-values to assess the significance of the p-values • If 'adj = 'none" (default), there is no p-value adjustment correction • If 'adj = 'holm", p-values are adjusted using the sequential Bonferroni correction (Holm, 1979)

- If 'adj = 'bonferroni'', p-values are adjusted using the classic Bonferroni correction
- If 'adj = 'BH'', p-values are adjusted using Benjamini et Hochberg (1995) correction controlling false discovery rate

`output` A character string indicating the matrices included in the output list.

- If 'output = 'all'' (default), then D (distance matrix), C (covariance matrix), Rho (partial correlation matrix), M (graph incidence matrix) and S (strength matrix) are included
- If 'output = 'dist_graph'', then the distance matrix D is returned only with the values corresponding to the graph edges
- If 'output = 'str_graph'', then the strength values matrix S is returned only with the values corresponding to the graph edges
- If 'output = 'inc'', then the binary adjacency matrix M is returned
- If 'output = 'igraph'', then a graph of class `igraph` is returned

Details

The function allows to vary many parameters such as the genetic distance used, the formula used to compute the covariance, the statistical tolerance threshold, the p-values adjustment, among others.

Value

A `list` of objects of class `matrix`, an object of class `matrix` or a graph object of class `igraph`

Author(s)

P. Savary

References

Dyer RJ, Nason JD (2004). "Population graphs: the graph theoretic shape of genetic structure." *Molecular ecology*, **13**(7), 1713–1727. Benjamini Y, Hochberg Y (1995). "Controlling the false discovery rate: a practical and powerful approach to multiple testing." *Journal of the royal statistical society. Series B (Methodological)*, 289–300. Bowcock AM, Ruiz-Linares A, Tomfohrde J, Minch E, Kidd JR, Cavalli-Sforza LL (1994). "High resolution of human evolutionary trees with polymorphic microsatellites." *nature*, **368**(6470), 455–457. Everitt B, Hothorn T (2011). *An introduction to applied multivariate analysis with R*. Springer Science \& Business Media. Excoffier L, Smouse PE, Quattro JM (1992). "Analysis of molecular variance inferred from metric distances among DNA haplotypes: application to human mitochondrial DNA restriction data." *Genetics*, **131**(2), 479–491. Fortuna MA, Albaladejo RG, FernÃ¡ndez L, Aparicio A, Bascompte J (2009). "Networks of spatial genetic variation across species." *Proceedings of the National Academy of Sciences*, **106**(45), 19044–19049. Holm S (1979). "A simple sequentially rejective multiple test procedure." *Scandinavian journal of statistics*, 65–70. Magwene PM (2001). "New tools for studying integration and modularity." *Evolution*, **55**(9), 1734–1745. Wermuth N, Scheidt E (1977). "Algorithm AS 105: fitting a covariance selection model to a matrix." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **26**(1), 88–92. Whittaker J (2009). *Graphical models in applied multivariate statistics*. Wiley Publishing.

Examples

```
data(data_ex_genind)
dist_graph_test <- gen_graph_indep(x = data_ex_genind, dist = "basic",
                              cov = "sq", pcor = "magwene",
                              alpha = 0.05, test = "EED",
                              adj = "none", output = "igraph")
```

---

gen_graph_thr *Create a graph of genetic differentiation using a link weight threshold*

Description

The function allows to construct a genetic graph whose links' weights are larger or lower than a specific threshold

Usage

```
gen_graph_thr(mat_w, mat_thr = NULL, thr, mode = "larger")
```

Arguments

`mat_w`	A symmetric (pairwise) `matrix` or a `dist` object whose elements will be the links' weights
`mat_thr`	(optional) A symmetric (pairwise) distance `matrix` or a `dist` object whose values will be used for the pruning based on the threshold value.
`thr`	The threshold value (logically between min(mat_thr) and max(mat_thr))(integer or numeric)
`mode`	• If 'mode = 'larger'' (default), all the links whose weight is larger than 'thr' are removed. • If 'mode = 'lower'', all the links whose weight is lower than 'thr' are removed.

Details

If 'mat_thr' is not defined, 'mat_w' is used for the pruning. Matrices 'mat_w' and 'mat_thr' must have the same dimensions and the same rows' and columns' names. Values in 'mat_thr' matrix must be positive. Negative values from 'mat_w' are transformed into zeros. The function works only for undirected graphs. If dist objects are specified, it is assumed that colnames and row.names of mat_w and mat_thr refer to the same populations/locations.

Value

A graph object of class `igraph`

Author(s)

P. Savary

Examples

```
mat_w <- mat_gen_dist(x = data_ex_genind, dist = 'DPS')
suppressWarnings(mat_thr <- mat_geo_dist(pts_pop_ex,
                ID = "ID",
                x = "x",
               y = "y"))
mat_thr <- mat_thr[row.names(mat_w), colnames(mat_w)]
graph <- gen_graph_thr(mat_w, mat_thr, thr = 6000, mode = "larger")
```

`gen_graph_topo` *Create a graph of genetic differentiation with a specific topology*

Description

The function constructs a genetic graph with a specific topology from genetic and/or geographical distance matrices

Usage

```
gen_graph_topo(mat_w, mat_topo = NULL, topo = "gabriel", k = NULL)
```

Arguments

`mat_w`	A symmetric (pairwise) `matrix` or a `dist` object whose elements will be the links' weights
`mat_topo`	(optional) A symmetric (pairwise) distance `matrix` or a `dist` object whose values will be used for the pruning method.
`topo`	Which topology does the created graph have?

- If 'topo = 'gabriel" (default), the resulting graph will be a Gabriel graph (Gabriel et al., 1969). It means that there is a link between nodes x and y if and only if $d_{xy}^2 \leq \min(\sqrt{d_{xz}^2 + d_{yz}^2})$, with z any other node of the graph.
- If 'topo = 'mst", the resulting graph will have the topology of a minimum spanning tree. It means that the graph will not include any cycle (tree) and it will be the subgraph with a tree topology with the minimum total links' weight (based on 'mat_topo' values).
- If 'topo = 'percol", if the link of the resulting graph with the minimum weight is removed, then the graph breaks into two components.
- If 'topo = 'comp", a complete graph whose links are weighted with values from 'mat_w' is created.
- If 'topo = 'knn", a k-nearest neighbor graph whose links are weighted with values from 'mat_w' is created. If the distance between node i and node j is among the k-th smallest distances between node i and the other nodes according to distances in matrix 'mat_topo', then there is a link between i and j in the resulting graph. Therefore, a node can be connected to more than two nodes because the nearest node to node j is not necessarily among

the k nearest neighbors to node i. Let d1 be the smallest distance from node i to other nodes, if there are k nodes or more at this distance from node i, they are all connected to i. If there are less than k nodes connected to i at a distance d1, then we consider nodes at a distance d2 from i. In the latter case, all the nodes at a distance d2 from i are connected to i.

`k`	(if 'topo = 'knn") An integer which indicates the number of nearest neighbors considered to create the K-nearest neighbor graph. k must be lower than the total number of nodes minus 1.

Details

If 'mat_topo' is not defined, 'mat_w' is used for the pruning. Matrices 'mat_w' and 'mat_topo' must have the same dimensions and the same rows' and columns' names. Values in 'mat_topo' matrix must be positive. Negative values from 'mat_w' are transformed into zeros. The function works only for undirected graphs. Note that the topology 'knn' works best when 'mat_topo' contains distance values from a continuous value range, thereby avoiding equal distances between a node and the others. are more than k nodes located at distances in the k-th smallest distances If dist objects are specified, it is assumed that colnames and row.names of mat_w and mat_topo refer to the same populations/locations.

Value

A graph object of class `igraph`

Author(s)

P. Savary

References

Gabriel KR, Sokal RR (1969). "A new statistical approach to geographic variation analysis." *Systematic zoology*, **18**(3), 259–278.

Examples

```
mat_w <- mat_gen_dist(x = data_ex_genind, dist = 'DPS')
suppressWarnings(mat_topo <- mat_geo_dist(pts_pop_ex,
                 ID = "ID",
                 x = "x",
                y = "y"))
mat_topo <- mat_topo[row.names(mat_w), colnames(mat_w)]
graph <- gen_graph_topo(mat_w, mat_topo, topo = "mst")
```

`get_graphab` *Download Graphab if not present on the user's machine*

Description

The function checks for the presence of Graphab (.jar) on the user's machine and downloads it if absent. It also checks that users have installed java on their machine.

Usage

```
get_graphab(res = TRUE, return = FALSE)
```

Arguments

`res`	Logical indicating whether a message says if Graphab has been downloaded or not.
`return`	Logical indicating whether the function returns a 1 or a 0 to indicate if Graphab has been downloaded or not.

Details

If the download does not work, you can create a directory named 'graph4lg_jar' in the directory `rappdirs::user_data_dir()` and copy Graphab software downloaded from `https://thema.univ-fcomte.fr/productions/download.php?name=graphab&version=2.4&username=Graph4lg&institution=R`

Value

If res = TRUE, the function displays a message indicating to users what has been done. If return = TRUE, it returns a 0 if Graphab is already on the machine and a 1 if it has been downloaded.

Author(s)

P. Savary

Examples

```
## Not run:
get_graphab()

## End(Not run)
```

`get_graphab_linkset` *Get linkset computed in the Graphab project*

Description

The function gets a linkset computed in the Graphab project

Usage

```
get_graphab_linkset(proj_name, linkset, proj_path = NULL)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is.
`linkset`	A character string indicating the name of the link set whose properties are imported. The link set has been created with Graphab or using `graphab_link` function.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.

Details

See more information in Graphab 2.4 manual: `https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf`. This function works if `link{get_graphab}` function works correctly.

Value

A data.frame with the link properties (from, to, cost-distance, Euclidean distance)

Author(s)

P. Savary

Examples

```
## Not run:
get_graphab_linkset(proj_name = "grphb_ex",
               linkset = "lkst1")

## End(Not run)
```

`get_graphab_metric` *Get metrics computed at the node in the Graphab project*

Description

The function gets the metrics computed at the node-level in the Graphab project

Usage

```
get_graphab_metric(proj_name, proj_path = NULL)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.

Details

The imported metrics describe the patches and have been computed from the different graphs created in the Graphab project. See more information in Graphab 2.4 manual: https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf

Value

A data.frame with metrics computed at the patch level.

Author(s)

P. Savary

Examples

```
## Not run:
get_graphab_metric(proj_name = "grphb_ex")

## End(Not run)
```

graphab_graph *Create a graph in the Graphab project*

Description

The function creates a graph from a link set in a Graphab project

Usage

```
graphab_graph(
  proj_name,
  linkset = NULL,
  name = NULL,
  thr = NULL,
  cost_conv = FALSE,
  proj_path = NULL,
  alloc_ram = NULL
)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is. It can be created with `graphab_project`
`linkset`	(optional, default=NULL) A character string indicating the name of the link set used to create the graph. If `linkset=NULL`, every link set present in the project will be used to create a graph. Link sets can be created with `graphab_link`.
`name`	(optional, default=NULL) A character string indicating the name of the graph created. If `name=NULL`, a name will be created. If both `linkset=NULL` and `name=NULL`, then a graph will be created for every link set present in the project and a name will be created every time. In the latter case, a unique name cannot be specified. Link sets can be created with `graphab_link`.
`thr`	(optional, default=NULL) An integer or numeric value indicating the maximum distance associated with the links of the created graph. It allows users to create a pruned graph based on a distance threshold. Note that when the link set used has a planar topology, the graph is necessarily a pruned graph (not complete) and adding this threshold parameter can remove other links. When the link set has been created with cost-distances, the parameter is expressed in cost-distance units whereas when the link set is based upon Euclidean distances, the parameter is expressed in meters.
`cost_conv`	FALSE (default) or TRUE. Logical indicating whether numeric `thr` values are converted from cost-distance into Euclidean distance using a log-log linear regression. See also `convert_cd` function.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the

	current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.

Details

By default, intra-patch distances are considered for metric calculation. See more information in Graphab 2.4 manual: https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf

Author(s)

P. Savary

Examples

```
## Not run:
graphab_graph(proj_name = "grphb_ex",
              linkset = "lcp",
              name = "graph")

## End(Not run)
```

`graphab_link` *Create a link set in the Graphab project*

Description

The function creates a link set between habitat patches in the Graphab project.

Usage

```
graphab_link(
  proj_name,
  distance = "cost",
  name,
  cost = NULL,
  topo = "planar",
  proj_path = NULL,
  alloc_ram = NULL
)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is. It can be created with `graphab_project`
`distance`	A character string indicating whether links between patches are computed based on: • Shortest cost distances: `distance='cost'` (default) • Straight Euclidean distances: `distance='euclid'` In the resulting link set, each link will be associated with its corresponding cost-distance and the length of the least-cost path in meters (if `distance='cost'`) or with its length in Euclidean distance (if `distance='euclid'`)
`name`	A character string indicating the name of the created linkset.
`cost`	A `data.frame` indicating the cost values associated to each raster cell value. These values refer to the raster used to create the project with `graphab_project`. The data.frame must have two columns: • 'code': raster cell values • 'cost': corresponding cost values
`topo`	A character string indicating the topology of the created link set. It can be: • Planar (`topo='planar'` (default)): a planar set of links is created. It speeds up the computation but will prevent from creating complete graphs with `graphab_graph`. • Complete (`topo='complete'`): a complete set of links is created. A link is computed between every pair of patches.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.

Details

By default, links crossing patches are not ignored nor broken into two links. For example, a link from patches A to C crossing patch B is created. It takes into account the distance inside patch B. It can be a problem when computing BC index. See more information in Graphab 2.4 manual: `https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf`

Author(s)

P. Savary

Examples

```
## Not run:
df_cost <- data.frame(code = 1:5,
                      cost = c(1, 10, 100, 1000, 1))
graphab_link(proj_name = "grphb_ex",
             distance = "cost",
             name = "lcp",
             cost = df_cost,
             topo = "complete")

## End(Not run)
```

`graphab_metric` *Compute connectivity metrics from a graph in the Graphab project*

Description

The function computes connectivity metrics on a graph from a link set in a Graphab project

Usage

```
graphab_metric(
  proj_name,
  graph,
  metric,
  dist = NULL,
  prob = 0.05,
  beta = 1,
  cost_conv = FALSE,
  return_val = TRUE,
  proj_path = NULL,
  alloc_ram = NULL
)
```

Arguments

`proj_name` A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is.

`graph` A character string indicating the name of the graph on which the metric is computed. This graph has been created with Graphab or using `graphab_graph` function and is associated with a link set. Only the links present in the graph and their corresponding weights will be used in the computation, together with patch areas.

`metric` A character string indicating the metric which will be computed on the graph. This metric can be:

- A global metric:

- Probability of Connectivity (`metric = 'PC'`): Sum of products of area of all pairs of patches weighted by their interaction probability, divided by the square of the area of the study zone. This ratio is the equivalent to the probability that two points randomly placed in the study area are connected.
- Integral Index of Connectivity (`metric = 'IIC'`): For the entire graph: product of patch areas divided by the number of links between them, the sum is divided by the square of the area of the study zone. IIC is built like the PC index but using the inverse of a topological distance rather than a negative exponential function of the distance based on the link weight.

- A local metric:
  - Flux (`metric = 'F'`): For the focal patch i : sum of area of patches other than i and weighted according to their minimum distance to the focal patch through the graph. This sum is an indicator of the potential dispersion from the patch i or, conversely to the patch i
  - Betweenness Centrality index (`metric = 'BC'`): Sum of the shortest paths through the focal patch i, each path is weighted by the product of the areas of the patches connected and of their interaction probability. All possible paths between every pair of patches is considered in this computation.
  - Interaction Flux (`metric = 'IF'`): Sum of products of the focal patch area with all the other patches, weighted by their interaction probability.
  - Degree (`metric = 'Dg'`): Number of edges connected to the node i i.e. number of patches connected directly to the patch i.
  - Closeness Centrality index (`metric = 'CCe'`): Mean distance from the patch i to all other patches of its component k.
  - Current Flux (`metric = 'CF'`): Sum of currents passing through the patch i. c_i^j represents the current through the patch i when currents are sent from all patches (except j) to the patch j. The patch j is connected to the ground.
- A delta metric:
  - delta Probability of Connectivity (`metric = 'dPC'`): Rate of variation between the value of PC index and the value of PC' corresponding to the removal of the patch i. The value of `dPC` is decomposed into three parts:
    * dPC_{area} is the variation induced by the area lost after removal;
    * dPC_{flux} is the variation induced by the loss of interaction between the patch i and other patches;
    * $dPC_{connector}$ is the variation induced by the modification of paths connecting other patches and initially routed through i.

For most metrics, the interaction probability is computed for each pair of patches from the path that minimizes the distance d (or the cost) between them. It then maximizes $e^{-\alpha d_{ij}}$ for patches i and j. To use patch capacity values different from the patch area, please use directly Graphab software.

`dist`	A numeric or integer value specifying the distance at which dispersal probability is equal to `prob`. This argument is mandatory for weighted metrics (PC, F, IF, BC, dPC, CCe, CF) but not used for others. It is used to set α for computing dispersal probabilities associated with all inter-patch distances such that dispersal probability between patches i and j is $p_{ij} = e^{-\alpha d_{ij}}$.
`prob`	A numeric or integer value specifying the dispersal probability at distance `dist`. By default, `code=0.05`. It is used to set α (see param `dist` above).
`beta`	A numeric or integer value between 0 and 1 specifying the exponent associated with patch areas in the computation of metrics weighted by patch area. By default, `beta=1`. When `beta=0`, patch areas do not have any influence in the computation.
`cost_conv`	FALSE (default) or TRUE. Logical indicating whether numeric `dist` values are converted from cost-distance into Euclidean distance using a log-log linear regression. See also `convert_cd` function.
`return_val`	Logical (default = TRUE) indicating whether metric values are returned in R (TRUE) or only stored in the patch attribute layer (FALSE)
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.

Details

The metrics are described in Graphab 2.4 manual: https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf Graphab software makes possible the computation of other metrics.

Value

If `return_val=TRUE`, the function returns a `data.frame` with the computed metric values and the corresponding patch ID when the metric is local or delta metric, or the numeric value of the global metric.

Author(s)

P. Savary

Examples

```
## Not run:
graphab_metric(proj_name = "grphb_ex",
               graph = "graph",
               metric = "PC",
               dist = 1000,
               prob = 0.05,
                beta = 1)

## End(Not run)
```

---

graphab_modul *Create modules from a graph in the Graphab project*

Description

The function creates modules from a graph by maximising modularity

Usage

```
graphab_modul(
  proj_name,
  graph,
  dist,
  prob = 0.05,
  beta = 1,
  nb = NULL,
  return = TRUE,
  proj_path = NULL,
  alloc_ram = NULL
)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is.
`graph`	A character string indicating the name of the graph on which the modularity index is computed. This graph has been created with Graphab or using `graphab_graph` function and is associated with a link set. Only the links present in the graph and their corresponding weights will be used in the computation, together with patch areas.
`dist`	A numeric or integer value specifying the distance at which dispersal probability is equal to `prob`. This argument is mandatory for weighted metrics (PC, F, IF, BC, dPC, CCe, CF) but not used for others. It is used to set α for computing dispersal probabilities associated with all inter-patch distances such that dispersal probability between patches i and j is $p_{ij} = e^{-\alpha d_{ij}}$.
`prob`	A numeric or integer value specifying the dispersal probability at distance `dist`. By default, `code=0.05`. It is used to set α (see param `dist` above).
`beta`	A numeric or integer value between 0 and 1 specifying the exponent associated with patch areas in the computation of metrics weighted by patch area. By default, `beta=1`. When `beta=0`, patch areas do not have any influence in the computation.

`nb`	(optional, default=NULL) An integer or numeric value indicating the number of modules to be created. By default, it is the number that maximises the modularity index.
`return`	Logical (default=TRUE) indicating whether results are returned to user.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.

Details

This function maximises a modularity index by searching for the node partition involves a large number of links within modules and a small number of inter-module links. Each link is given a weight in the computation, such as the weight w_{ij} of the link between patches i and j is:

$$w_{ij} = (a_i a_j)^\beta e^{-\alpha d_{ij}}$$

. This function does not allow users to convert automatically Euclidean distances into cost-distances. See more information in Graphab 2.4 manual: https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf

Value

If `return=TRUE`, the function returns a message indicating whether the partition has been done. New options are being developed.

Author(s)

P. Savary

Examples

```
## Not run:
graphab_modul(proj_name = "grphb_ex",
              graph = "graph",
              dist = 1000,
              prob = 0.05,
              beta = 1)

## End(Not run)
```

`graphab_pointset` *Add a point set to the Graphab project*

Description

The function adds a spatial point set to the Graphab project, allowing users to identify closest habitat patch from each point and get corresponding connectivity metrics.

Usage

```
graphab_pointset(
  proj_name,
  linkset,
  pointset,
  return_val = TRUE,
  proj_path = NULL,
  alloc_ram = NULL
)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml is.
`linkset`	A character string indicating the name of the link set used. The link set is here used to get the defined cost values and compute the distance from the point to the patches. Link sets can be created with `graphab_link`.
`pointset`	Can be either; • A character string indicating the path (absolute or relative) to a shapefile point layer • A character string indicating the path to a .csv file with three columns: ID, x and y, respectively indicating the point ID, longitude and latitude • A data.frame with three columns: ID, x and y, respectively indicating the point ID, longitude and latitude. • A SpatialPointsDataFrame
`return_val`	Logical (default=TRUE) indicating whether the metrics associated with closest habitat patches from the points are returned to users.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.

Details

Point coordinates must be in the same coordinate reference system as the habitat patches (and initial raster layer). See more information in Graphab 2.4 manual: `https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf`

Value

If `return_val=TRUE`, the function returns a `data.frame` with the properties of the nearest patch to every point in the point set, as well as the distance from each point to the nearest patch.

Author(s)

P. Savary

Examples

```
## Not run:
graphab_pointset(proj_name = "grphb_ex",
                 graph = "graph",
                 pointset = "pts.shp")

## End(Not run)
```

`graphab_project` *Create a Graphab project*

Description

The function creates a Graphab project from a raster file on which habitat patches can be delimited.

Usage

```
graphab_project(
  proj_name,
  raster,
  habitat,
  minarea = 0,
  nodata = NULL,
  alloc_ram = NULL,
  proj_path = NULL
)
```

Arguments

`proj_name`	A character string indicating the Graphab project name. The project name is also the name of the project directory in which the file proj_name.xml will be created.

`raster`	A character string indicating the name of the .tif raster file or of its path. If the path is not specified, the raster must present in the current working directory. Raster cell values must be in INT2S encoding.
`habitat`	An integer or numeric value or vector indicating the code.s (cell value.s) of the habitat cells in the raster file.
`minarea`	(optional, default=0) An integer or numeric value specifiying the minimum area in hectares for a habitat patch size to become a graph node.
`nodata`	(optional, default=NULL) An integer or numeric value specifying the code in the raster file associated with nodata value (often corresponding to peripheric cells)
`alloc_ram`	(optional, default = NULL) Integer or numeric value indicating RAM gigabytes allocated to the java process. Increasing this value can speed up the computations. Too large values may not be compatible with your machine settings.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory. It should be used when the project directory is not in the current working directory. Default is NULL. When 'proj_path = NULL', the project directory is equal to `getwd()`.

Details

A habitat patch consists of the central pixel with its eight neighbors if they are of the same value (8-connexity) and the path geometry is not simplified. See more information in Graphab 2.4 manual: `https://sourcesup.renater.fr/www/graphab/download/manual-2.4-en.pdf`

Author(s)

P. Savary

Examples

```
## Not run:
proj_name <- "grphb_ex"
raster <- "rast_ex.tif"
habitat <- 5
graphab_project(proj_name = proj_name,
                raster = raster,
                habitat = habitat)

## End(Not run)
```

`graphab_to_igraph` *Create landscape graphs from Graphab link set*

Description

The function creates a landscape graph from a link set created with Graphab software or different functions of this package and converts it into a graph object of class `igraph`. The graph has weighted links and is undirected. Nodes attributes present in the Graphab project are included, including connectivity metrics when computed

Usage

```
graphab_to_igraph(
  proj_name,
  linkset,
  nodes = "patches",
  weight = "cost",
  proj_path = NULL,
  fig = FALSE,
  crds = FALSE
)
```

Arguments

`proj_name`	A character string indicating the project name. It is also the name of the directory in which proj_name.xml file is found. By default, 'proj_name' is searched into the current working directory
`linkset`	A character string indicating the name of the linkset used to create the graph links. The linkset must have been created previously (see the function `graphab_link`). It can be complete or planar. The graph is given the topology of the selected link set.
`nodes`	A character string indicating whether the nodes of the created graph are given all the attributes or metrics computed in Graphab or only those specific to a given graph previously created with `graphab_graph` It can be: • `nodes = "patches"`(default): all the attributes and metrics of the habitat patches are included as node attributes in `igraph` object. • `nodes = "graph_name"`(default): only the metrics of the habitat patches computed from the graph 'graph_name' created with `graphab_graph` are included as node attributes in `igraph` object, along with some basic patch attributes.
`weight`	A character string ("euclid" or "cost") indicating whether to weight the links with Euclidean distance or cost-distance (default) values.
`proj_path`	(optional) A character string indicating the path to the directory that contains the project directory ('proj_name'). By default, 'proj_name' is searched into the current working directory
`fig`	Logical (default = FALSE) indicating whether to plot a figure of the resulting spatial graph. The figure is plotted using function `plot_graph_lg`. The plotting can be long if the graph has many nodes and links.
`crds`	Logical (default = FALSE) indicating whether to create an object of class `data.frame` with the node centroid spatial coordinates. Such a `data.frame` has 3 columns: 'ID', 'x', 'y'.

Value

A graph object of class `igraph` (if crds = FALSE) or a list of objects: a graph object of class `igraph` and a `data.frame` with the nodes spatial coordinates (if crds = TRUE).

Author(s)

P. Savary

References

FoltÃªte J, Clauzel C, Vuidel G (2012). "A software tool dedicated to the modelling of landscape networks." *Environmental Modelling \& Software*, **38**, 316–327.

Examples

```
## Not run:
proj_path <- system.file('extdata',package='graph4lg')
proj_name <- "grphb_ex"
linkset <- "lkst1"
nodes <- "graph"
graph <- graphab_to_igraph(proj_name = proj_name,
                           linkset = "lkst1",
                           nodes = "graph",
                           links = links,
                           weights = "cost",
                           proj_path = proj_path,
                           crds = FALSE,
                           fig = FALSE)

## End(Not run)
```

`graph_modul_compar` *Compare the partition into modules of two graphs*

Description

The function computes the Adjusted Rand Index (ARI) to compare two graphs' partitions into modules or clusters more generally. Both graphs must have the same number of nodes, but not necessarily the same number of links. They must also have the same node names and in the same order.

Usage

```
graph_modul_compar(
  x,
  y,
  mode = "graph",
  nb_modul = NULL,
  algo = "fast_greedy",
  node_inter = "distance",
  data = NULL
)
```

Arguments

`x` The first graph object

- If `mode = 'graph'` (default), `x` is a graph object of class `igraph`. Then, its nodes must have the same names as in graph `y`.
- If `mode = 'data.frame'`, `x` refers to a column of the `data.frame` 'data'. Then `x` must be a character string indicating the name of the column of 'data' with the modules' labels of the nodes in the first graph. In that case, the column can be of class `numeric`, `character` or `factor` but will be converted into a `numeric` vector in any case.
- If `mode = 'vector'`, `x` is a vector of class `character`, `factor` or `numeric`. In that case, it must have the same length as vector `y` and will be converted into a `numeric` vector.

`y` The second graph object Same classes possible as for `x`. Must be of the same format as `x`

`mode` A character string indicating whether `x` and `y` are igraph objects, vectors or columns from a data.frame. `mode` can be 'graph', 'data.frame' or 'vector'.

`nb_modul` (if `x` and `y` are igraph objects) A numeric or integer value or a numeric vector with 2 elements indicating the number of modules to create in both graphs.

- If `nb_modul` is a numeric value, then the same number of modules are created in both graphs.
- If `nb_modul` is a numeric vector of length 2, then the numbers of modules created in graphs `x` and `y` are the first and second elements of `nb_modul`, respectively.

`algo` (if `x` and `y` are igraph objects) A character string indicating the algorithm used to create the modules with **igraph**.

- If `algo = 'fast_greedy'` (default), function `cluster_fast_greedy` from **igraph** is used (Clauset et al., 2004).
- If `algo = 'walktrap'` (default), function `cluster_walktrap` from **igraph** is used (Pons et Latapy, 2006) with 4 steps (default options).
- If `algo = 'louvain'`, function `cluster_louvain` from **igraph** is used (Blondel et al., 2008). In that case, the number of modules created in each graph is imposed.
- If `algo = 'optimal'`, function `cluster_optimal` from **igraph** is used (Brandes et al., 2008) (can be very long). In that case, the number of modules created in each graph is imposed.

`node_inter` (optional, if `x` and `y` are igraph objects, default is 'none') A character string indicating whether the links of the graph are weighted by distances or by similarity indices. It is only used to compute the modularity index. It can be:

- 'distance': Link weights correspond to distances. Nodes that are close to each other will more likely be in the same module.

- 'similarity': Link weights correspond to similarity indices. Nodes that are similar to each other will more likely be in the same module. Inverse link weights are then used to compute the modularity index.
- 'none': Links are not weighted for the computation, which is only based on graph topology.

Two different weightings can be used to create the modules of the two graphs.

- If `node_inter` is a character string, then the same link weighting is used for both graphs.
- If `node_inter` is a character vector of length 2, then the link weighting used by the algorithm to create the modules of graphs `x` and `y` is determined by the first and second elements of `node_inter`, respectively.

`data` (if `x` and `y` are columns from a data.frame) An object of class data.frame with at least two columns and as many rows as there are nodes in the graphs compared. The columns indicate the modules of each node in 2 different classifications.

Details

This index takes values between -1 and 1. It measures how often pairs of nodes pertaining to the same module in one graph also pertain to the same module in the other graph. Therefore, large values indicate that both partitions are similar. The Rand Index can be defined as the frequency of agreement between two classifications into discrete classes. It is the number of times a pair of elements are classified into the same class or in two different classes in both compared classifications, divided by the total number of possible pairs of elements. The Rand Index is between 0 and 1 but its maximum value depends on the number of elements. Thus, another 'adjusted' index was created, the Adjusted Rand Index. According to the Hubert et Arabie's formula, the ARI is computed as follows: $ARI = \frac{Index - Expectedindex}{Maximumindex - Expectedindex}$ where the values of Index, Expected index and Maximum index are computed from a contingency table. This function uses `adjustedRandIndex` from package **mclust** which applies the Hubert and Arabie's formula for the ARI. This function works for undirected graphs only.

Value

The value of the ARI

Author(s)

P. Savary

References

Dyer RJ, Nason JD (2004). "Population graphs: the graph theoretic shape of genetic structure." *Molecular ecology*, **13**(7), 1713–1727. Hubert L, Arabie P (1985). "Comparing partitions." *Journal of classification*, **2**(1), 193–218. Clauset A, Newman ME, Moore C (2004). "Finding community structure in very large networks." *Physical review E*, **70**(6). Blondel VD, Guillaume J, Lambiotte R, Lefebvre E (2008). "Fast unfolding of communities in large networks." *Journal of Statistical Mechanics - Theory and Experiment*, **10**. Brandes U, Delling D, Gaertler M, Gorke R, Hoefer M, Nikoloski Z, Wagner D (2008). "On modularity clustering." *IEEE transactions on knowledge and data engineering*, **20**(2), 172–188. Pons P, Latapy M (2006). "Computing communities in large networks using random walks." *J. Graph Algorithms Appl.*, **10**(2), 191–218.

Examples

```
data(data_ex_genind)
data(pts_pop_ex)
mat_dist <- suppressWarnings(graph4lg::mat_geo_dist(data=pts_pop_ex,
      ID = "ID",
      x = "x",
      y = "y"))
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                      order(as.character(colnames(mat_dist)))]
graph_obs <- gen_graph_thr(mat_w = mat_dist, mat_thr = mat_dist,
                            thr = 24000, mode = "larger")
mat_gen <- mat_gen_dist(x = data_ex_genind, dist = "DPS")
graph_pred <- gen_graph_topo(mat_w = mat_gen, mat_topo = mat_dist,
                            topo = "gabriel")
ARI <- graph_modul_compar(x = graph_obs, y = graph_pred)
```

---

graph_node_compar *Compare the local properties of the nodes from two graphs*

Description

The function computes a correlation coefficient between the graph-theoretic metric values computed at the node-level in two graphs sharing the same nodes. It allows to assess whether the connectivity properties of the nodes in one graph are similar to that of the same nodes in the other graph. Alternatively, the correlation is computed between a graph-theoretic metric values and the values of an attribute associated to the nodes of a graph.

Usage

```
graph_node_compar(
  x,
  y,
  metrics = c("siw", "siw"),
  method = "spearman",
  weight = TRUE,
  test = TRUE
)
```

Arguments

x	An object of class `igraph`. Its nodes must have the same names as in graph `y`.
y	An object of class `igraph`. Its nodes must have the same names as in graph `x`.
metrics	Two-element character vector specifying the graph-theoretic metrics computed at the node-level in the graphs or the node attribute values to be correlated to these metrics. Graph-theoretic metrics can be: • Degree (`metrics = c("deg",...)`) • Closeness centrality index (`metrics = c("close",...)`)

- Betweenness centrality index (`metrics = c("btw",...)`)
- Strength (sum of the weights of the links connected to a node) (`metrics = c("str",...)`)
- Sum of the inverse weights of the links connected to a node (`metrics = c("siw",...)`, default)
- Mean of the inverse weights of the links connected to a node (`metrics = c("miw",...)`)

Node attributes must have the same names as in the `igraph` object, and must refer to an attribute with numerical values. The vector `metrics` is composed of two character values. When a node attribute has the same name as a metric computable from the graph, node attributes are given priority.

`method`	A character string indicating which correlation coefficient is to be computed (`"pearson"`, `"kendall"` or `"spearman"` (default)).
`weight`	Logical which indicates whether the links are weighted during the calculation of the centrality indices betweenness and closeness. (default: `weight = TRUE`). Link weights are interpreted as distances when computing the shortest paths. They should then be inversely proportional to the strength of the relationship between nodes (e.g. to fluxes).
`test`	Logical. Should significance testing be performed? (default = TRUE)

Details

The correlation coefficients between the metrics can be computed in different ways, as initial assumptions (e.g. linear relationship) are rarely verified. Pearson's r, Spearman's rho and Kendall's tau can be computed (from function `cor`). When `x` is similar to `y`, then the correlation is computed between two metrics characterizing the nodes of the same graph.

Value

A `list` summarizing the correlation analysis.

Author(s)

P. Savary

Examples

```
data(data_ex_genind)
data(pts_pop_ex)
mat_dist <- suppressWarnings(graph4lg::mat_geo_dist(data = pts_pop_ex,
      ID = "ID",
      x = "x",
      y = "y"))
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                      order(as.character(colnames(mat_dist)))]
graph_obs <- gen_graph_thr(mat_w = mat_dist, mat_thr = mat_dist,
                           thr = 9500, mode = "larger")
mat_gen <- mat_gen_dist(x = data_ex_genind, dist = "DPS")
graph_pred <- gen_graph_topo(mat_w = mat_gen, mat_topo = mat_dist,
                            topo = "gabriel")
res_cor <- graph_node_compar(x = graph_obs, y = graph_pred,
                             metrics = c("siw", "siw"), method = "spearman",
                             test = TRUE, weight = TRUE)
```

---

graph_plan *Create a graph with a minimum planar graph topology*

Description

The function constructs a graph with a minimum planar graph topology

Usage

```
graph_plan(crds, ID = NULL, x = NULL, y = NULL, weight = TRUE)
```

Arguments

`crds`	A `data.frame` with the spatial coordinates of the point set (the graph nodes). It must have three columns: • ID: A character string indicating the name of the points(graph nodes). • x: A numeric or integer indicating the longitude of the graph nodes. • y: A numeric or integer indicating the latitude of the graph nodes.
`ID`	A character string indicating the name of the column of `crds` with the point IDs
`x`	A character string indicating the name of the column of `crds` with the point longitude
`y`	A character string indicating the name of the column of `crds` with the point latitude
`weight`	A character string indicating whether the links of the graph are weighted by Euclidean distances (TRUE)(default) or not (FALSE). When the graph links do not have weights in Euclidean distances, each link is given a weight of 1.

Details

A delaunay triangulation is performed in order to get the planar graph.

Value

A planar graph of class `igraph`

Author(s)

P. Savary

Examples

```
data(pts_pop_ex)
g_plan <- graph_plan(crds = pts_pop_ex,
             ID = "ID",
             x = "x",
             y = "y")
```

`graph_plot_compar` *Visualize the topological differences between two spatial graphs on a map*

Description

The function enables to compare two spatial graphs by plotting them highlighting the topological similarities and differences between them. Both graphs should share the same nodes and cannot be directed graphs.

Usage

```
graph_plot_compar(x, y, crds)
```

Arguments

`x`	A graph object of class `igraph`. Its nodes must have the same names as in graph `y`.
`y`	A graph object of class `igraph`. Its nodes must have the same names as in graph `x`.
`crds`	A `data.frame` with the spatial coordinates of the graph nodes (both `x` and `y`). It must have three columns: • ID: Name of the graph nodes (character string). The names must be the same as the node names of the graphs of class `igraph` (`igraph::V(graph)$name`) • x: Longitude of the graph nodes (numeric or integer). • y: Latitude of the graph nodes (numeric or integer).

Details

The graphs `x` and `y` of class `igraph` must have node names (not necessarily in the same order as IDs in crds, given a merging is done).

Value

A ggplot2 object to plot

Author(s)

P. Savary

Examples

```
data(pts_pop_ex)
data(data_ex_genind)
mat_w <- mat_gen_dist(data_ex_genind, dist = "DPS")
mat_dist <- mat_geo_dist(data = pts_pop_ex,
                         ID = "ID",
                         x = "x",
                         y = "y")
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                   order(as.character(colnames(mat_dist)))]
g1 <- gen_graph_topo(mat_w = mat_w, topo = "mst")
g2 <- gen_graph_topo(mat_w = mat_w, mat_topo = mat_dist, topo = "gabriel")
g <- graph_plot_compar(x = g1, y = g2,
                       crds = pts_pop_ex)
```

`graph_topo_compar` *Compute an index comparing graph topologies*

Description

The function computes several indices in order to compare two graph topologies. One of the graph has the "true" topology the other is supposed to reproduce. The indices are then a way to assess the reliability of the latter graph. Both graphs must have the same number of nodes, but not necessarily the same number of links. They must also have the same node names and in the same order.

Usage

```
graph_topo_compar(obs_graph, pred_graph, mode = "mcc", directed = FALSE)
```

Arguments

`obs_graph`	A graph object of class `igraph` with `n` nodes. It is the observed graph that `pred_graph` is supposed to approach.
`pred_graph`	A graph object of class `igraph` with `n` nodes. It is the predicted graph that is supposed to be akin to `obs_graph`.
`mode`	A character string specifying which index to compute in order to compare the topologies of the graphs. • If 'mode = 'mcc" (default), the Matthews Correlation Coefficient (MCC) is computed. • If 'mode = 'kappa", the Kappa index is computed. • If 'mode = 'fdr", the False Discovery Rate (FDR) is computed. • If 'mode = 'acc", the Accuracy is computed. • If 'mode = 'sens", the Sensitivity is computed. • If 'mode = 'spec", the Specificity is computed. • If 'mode = 'prec", the Precision is computed.
`directed`	Logical (TRUE or FALSE) specifying whether both graphs are directed or not.

Details

The indices are calculated from a confusion matrix counting the number of links that are in the "observed" graph ("true") and also in the "predicted" graph (true positives : TP), that are in the "observed" graph but not in the "predicted" graph (false negatives : FN), that are not in the "observed" graph but in the "predicted" graph (false positives : FP) and that are not in the "observed" graph and not in the "predicted" graph neither (true negatives: TN). K is the total number of links in the graphs. K is equal to $n \times (n-1)$ if the graphs are directed and to $\frac{n \times (n-1)}{2}$ if they are not directed, with n the number of nodes. OP = TP + FN, ON = TN + FP, PP = TP + FP and PN = FN + TN.

The Matthews Correlation Coefficient (MCC) is computed as follows: $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

The Kappa index is computed as follows: $Kappa = \frac{K \times (TP+TN) - (ON \times PN) - (OP \times PP)}{K^2 - (ON \times PN) - (OP \times PP)}$

The False Discovery Rate (FDR) is calculated as follows: $FDR = \frac{FP}{TP+FP}$

The Accuracy is calculated as follows: $Acc = \frac{TP+TN}{K}$

The Sensitivity is calculated as follows: $Sens = \frac{TP}{TP+FN}$

The Specificity is calculated as follows: $Spec = \frac{TN}{TN+FP}$

The Precision is calculated as follows: $Prec = \frac{TP}{TP+FP}$

Self loops are not taken into account.

Value

The value of the index computed

Author(s)

P. Savary

References

Dyer RJ, Nason JD (2004). “Population graphs: the graph theoretic shape of genetic structure.” *Molecular ecology*, **13**(7), 1713–1727. Baldi P, Brunak S, Chauvin Y, Andersen CA, Nielsen H (2000). “Assessing the accuracy of prediction algorithms for classification: an overview.” *Bioinformatics*, **16**(5), 412–424. Matthews BW (1975). “Comparison of the predicted and observed secondary structure of T4 phage lysozyme.” *Biochimica et Biophysica Acta (BBA)-Protein Structure*, **405**(2), 442–451.

Examples

```
data(data_ex_genind)
data(pts_pop_ex)
mat_dist <- suppressWarnings(graph4lg::mat_geo_dist(data=pts_pop_ex,
      ID = "ID",
      x = "x",
      y = "y"))
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                     order(as.character(colnames(mat_dist)))]
graph_obs <- gen_graph_thr(mat_w = mat_dist, mat_thr = mat_dist,
                            thr = 15000, mode = "larger")
mat_gen <- mat_gen_dist(x = data_ex_genind, dist = "DPS")
graph_pred <- gen_graph_topo(mat_w = mat_gen, mat_topo = mat_dist,
                            topo = "gabriel")
graph_topo_compar(obs_graph = graph_obs,
                  pred_graph = graph_pred,
                  mode = "mcc",
                  directed = FALSE)
```

graph_to_df *Convert a graph into a edge list data.frame*

Description

The function converts a graph into a edge list data.frame

Usage

```
graph_to_df(graph, weight = TRUE)
```

Arguments

graph	A graph object of class igraph
weight	Logical. If TRUE (default), then the column 'link' of the output data.frame contains the weights of the links. If FALSE, it contains only 0 and 1.

Details

The 'graph' nodes must have names. Links must have weights if 'weight = TRUE'.

Value

An object of class `data.frame` with a link ID, the origin nodes ('from') and arrival nodes ('to') and the link value ('link')(weighted or binary)

Author(s)

P. Savary

Examples

```
data(pts_pop_ex)
suppressWarnings(mat_geo <- mat_geo_dist(pts_pop_ex,
                 ID = "ID",
                 x = "x",
                y = "y"))
g1 <- gen_graph_thr(mat_w = mat_geo,
                    mat_thr = mat_geo,
                    thr = 20000)
```

```
g1_df <- graph_to_df(g1,
                     weight = TRUE)
```

graph_to_shp *Export a spatial graph to shapefile layers*

Description

The function enables to export a spatial graph to shapefile layers.

Usage

```
graph_to_shp(
  graph,
  crds,
  mode = "both",
  crds_crs,
  layer,
  dir_path,
  metrics = FALSE
)
```

Arguments

`graph`	A graph object of class `igraph`
`crds`	(if 'mode = 'spatial") A `data.frame` with the spatial coordinates of the graph nodes. It must have three columns: • ID: Name of the graph nodes (will be converted into character string). The names must the same as the node names of the graph object of class `igraph` (`igraph::V(graph)$name`) • x: Longitude (numeric or integer) of the graph nodes in the coordinates reference system indicated with the argument crds_crs. • y: Latitude (numeric or integer) of the graph nodes in the coordinates reference system indicated with the argument crds_crs.
`mode`	Indicates which shapefile layers will be created • If 'mode = 'both" (default), then two shapefile layers are created, one for the nodes and another for the links. • If 'mode = 'node", a shapefile layer is created for the nodes only. • If 'mode = 'link", a shapefile layer is created for the links only.
`crds_crs`	An integer indicating the EPSG code of the coordinates reference system to use. The projection and datum are given in the PROJ.4 format.
`layer`	A character string indicating the suffix of the name of the layers to be created.
`dir_path`	A character string corresponding to the path to the directory in which the shapefile layers will be exported. If `dir_path = "wd"`, then the layers are created in the current working directory.
`metrics`	(not considered if 'mode = 'link") Logical. Should graph node attributes integrated in the attribute table of the node shapefile layer? (default: FALSE)

Value

Create shapefile layers in the directory specified with the parameter 'dir_path'.

Author(s)

P. Savary

Examples

```
data(data_tuto)
mat_w <- data_tuto[[1]]
gp <- gen_graph_topo(mat_w = mat_w, topo = "gabriel")
crds_crs <- 2154
crds <- pts_pop_simul
layer <- "graph_dps_gab"
graph_to_shp(graph = gp, crds = pts_pop_simul, mode = "both",
             crds_crs = crds_crs,
             layer = "test_fonct",
             dir_path = tempdir(),
             metrics = FALSE)
```

`gstud_to_genind` *Convert a file from* **gstudio** *or* **popgraph** *into a genind object*

Description

The function converts a file formatted to use **gstudio** or **popgraph** package into a genind object (**adegenet** package)

Usage

```
gstud_to_genind(x, pop_col, ind_col = NULL)
```

Arguments

`x`	An object of class `data.frame` with loci columns in format `locus` (defined in package **gstudio**) with as many rows as individuals and as many columns in format `locus` as there are loci and additional columns
`pop_col`	A character string indicating the name of the column with populations' names in `x`
`ind_col`	(optional) A character string indicating the name of the column with individuals' ID in `x`

Details

This function uses functions from **pegas** package. It can handle genetic data where alleles codings do not have same length, (99:101, for example). If the names of the loci include '.' characters, they will be replaced by '_'.

Value

An object of class `genind`.

Author(s)

P. Savary

Examples

```
data("data_ex_gstud")
x <- data_ex_gstud
pop_col <- "POP"
ind_col <- "ID"
data_genind <- gstud_to_genind(x, pop_col, ind_col)
```

g_percol *Prune a graph using the 'percolation threshold' method*

Description

The function allows to prune a graph by removing the links with the largest weights until the graph breaks into two components. The returned graph is the last graph with only one component.

Usage

```
g_percol(x, val_step = 20)
```

Arguments

`x`	A symmetric `matrix` or a `dist` object with pairwise distances between nodes
`val_step`	The number of classes to create to search for the threshold value without testing all the possibilities. By default, 'val_step = 20'.

Value

A graph object of type `igraph`

Author(s)

P. Savary

Examples

```
data(data_ex_genind)
suppressWarnings(mat_w <- graph4lg::mat_geo_dist(data = pts_pop_ex,
                            ID = "ID",
                            x = "x",
                            y = "y"))
g_percol(x = mat_w)
```

kernel_param *Compute dispersal kernel parameters*

Description

The function computes the constant parameters of a dispersal kernel with a negative exponential distribution

Usage

```
kernel_param(p, d_disp, mode = "A")
```

Arguments

p	A numeric value indicating the dispersal probability at a distance equal to 'd_disp' under a negative exponential distribution.
d_disp	A numeric value indicating the distance to which dispersal probability is equal to 'p' under a negative exponential distribution.
mode	A character string indicating the value to return: • If 'mode = 'A" (default), the returned value 'alpha' is such that exp(-alpha * d_disp) = p • If 'mode = 'B", the returned value 'alpha' is such that 10(-alpha * d_disp) = p

Details

If the resulting parameter when mode = "A" is a and the resulting parameter when mode = "B" is b, then we have: p = exp(-a.d_disp) = 10^(-b.d_disp) and a = b.ln(10)

Value

A numeric value

Author(s)

P. Savary

Examples

```
p <- 0.5
d_disp <- 3000
alpha <- kernel_param(p, d_disp, mode = "A")
```

`loci_to_genind` *Convert a loci object into a genind object*

Description

This function is exactly the same as `loci2genind` from **pegas** package

Usage

```
loci_to_genind(x, ploidy = 2, na.alleles = c("NA"))
```

Arguments

`x`	An object of class `loci` to convert
`ploidy`	An integer indicating the ploidy level (by default, 'ploidy = 2')
`na.alleles`	A character vector indicating the coding of the alleles to be treated as missing data (by default, 'na.alleles = c("NA")')

Value

An object of class `genind`

Author(s)

P. Savary

Examples

```
data("data_ex_loci")
genind <- loci_to_genind(data_ex_loci, ploidy = 2, na.alleles = "NA")
```

`mat_cost_dist` *Compute cost distances between points on a raster*

Description

The function computes cost-distances associated to least cost paths between point pairs on a raster with specified cost values.

Usage

```
mat_cost_dist(
  raster,
  pts,
  cost,
  method = "gdistance",
  return = "mat",
  direction = 8,
  parallel.java = 1
)
```

Arguments

`raster` A parameter indicating the raster file on which cost distances are computed. It can be:

- A character string indicating the path to a raster file in format .tif or .asc.
- A `RasterLayer` object already loaded in R environment

All the raster cell values must be present in the column 'code' from `cost` argument.

`pts` A parameter indicating the points between which cost distances are computed. It can be either:

- A character string indicating the path to a .csv file. It must have three columns:
  - ID: The ID of the points.
  - x: A numeric or integer indicating the longitude of the points.
  - y: A numeric or integer indicating the latitude of the points.
- A `data.frame` with the spatial coordinates of the points. It must have three columns:
  - ID: The ID of the points.
  - x: A numeric or integer indicating the longitude of the points.
  - y: A numeric or integer indicating the latitude of the points.
- A `SpatialPointsDataFrame` with at least an attribute column named "ID" with the point IDs.

The point coordinates must be in the same spatial coordinate reference system as the raster file.

`cost` A `data.frame` indicating the cost values associated to each raster value. It must have two columns:

- 'code': raster cell values
- 'cost': corresponding cost values

`method` A character string indicating the method used to compute the cost distances. It must be:

- 'gdistance': uses the functions from the package **gdistance** assuming that movement is possible in 8 directions from each cell, that a geo-correction is applied to correct for diagonal movement lengths and that raster cell values correspond to resistance (and not conductance).

- 'java': uses a .jar file which is downloaded on the user's machine if necessary and if java is installed. This option substantially reduces computation times and makes possible the parallelisation.

`return`	A character string indicating whether the returned object is a `data.frame` (`return="df"`) or a pairwise `matrix` (`return="mat"`).
`direction`	An integer (4, 8, 16) indicating the directions in which movement can take place from a cell. Only used when `method="gdistance"`. By default, `direction=8`.
`parallel.java`	An integer indicating how many computer cores are used to run the .jar file. By default, `parallel.java=1`.

Value

The function returns:

- If `return="mat"`, a pairwise `matrix` with cost-distance values between points.
- If `return="df"`, an object of type `data.frame` with three columns:
  - from: A character string indicating the ID of the point of origin.
  - to: A character string indicating the ID of the point of destination.
  - cost_dist: A numeric indicating the accumulated cost-distance along the least-cost path between point ID1 and point ID2.

Author(s)

P. Savary

Examples

```
x <- raster::raster(ncol=10, nrow=10, xmn=0, xmx=100, ymn=0, ymx=100)
raster::values(x) <- sample(c(1,2,3,4), size = 100, replace = TRUE)
pts <- data.frame(ID = 1:4,
                  x = c(10, 90, 10, 90),
                  y = c(90, 10, 10, 90))
cost <- data.frame(code = 1:4,
                   cost = c(1, 10, 100, 1000))
mat_cost_dist(raster = x,
              pts = pts, cost = cost,
              method = "gdistance")
```

`mat_gen_dist` *Compute a pairwise matrix of genetic distances between populations*

Description

The function computes a pairwise matrix of genetic distances between populations and allows to implement several formula.

Usage

```
mat_gen_dist(x, dist = "basic", null_val = FALSE)
```

Arguments

`x`	An object of class `genind` that contains the multilocus genotypes (format 'locus') of the individuals as well as their populations.
`dist`	A character string indicating the method used to compute the multilocus genetic distance between populations • If 'dist = 'basic" (default), then the multilocus genetic distance is computed using a formula of Euclidean genetic distance (Excoffier et al., 1992) • If 'dist = 'weight", then the multilocus genetic distance is computed as in Fortuna et al. (2009). It is a Euclidean genetic distance giving more weight to rare alleles • If 'dist = 'PG", then the multilocus genetic distance is computed as in popgraph::popgraph function, following several steps of PCA and SVD (Dyer et Nason, 2004). • If 'dist = 'DPS", then the genetic distance used is equal to 1 - the proportion of shared alleles (Bowcock, 1994) • If 'dist = 'FST", then the genetic distance used is the pairwise FST (Weir et Cockerham, 1984) • If 'dist = 'FST_lin", then the genetic distance used is the linearised pairwise FST (Weir et Cockerham, 1984)(FST_lin = FST/(1-FST)) • If 'dist = 'PCA", then the genetic distance is computed following a PCA of the matrix of allelic frequencies by population. It is a Euclidean genetic distance between populations in the multidimensional space defined by all the independent principal components. • If 'dist = 'GST", then the genetic distance used is the G'ST (Hedrick, 2005) • If 'dist = 'D", then the genetic distance used is Jost's D (Jost, 2008)
`null_val`	(optional) Logical. Should negative and null FST, FST_lin, GST or D values be replaced by half the minimum positive value? This option allows to compute Gabriel graphs from these "distances". Default is null_val = FALSE. This option only works if 'dist = 'FST" or 'FST_lin' or 'GST' or 'D'

Details

Negative values are converted into 0. Euclidean genetic distance d_{ij} between population i and j is computed as follows:

$$d_{ij}^2 = \sum_{k=1}^{n} (x_{ki} - x_{kj})^2$$

where x_{ki} is the allelic frequency of allele k in population i and n is the total number of alleles. Note that when 'dist = 'weight", the formula becomes

$$d_{ij}^2 = \sum_{k=1}^{n} (1/(K * p_k))(x_{ki} - x_{kj})^2$$

where K is the number of alleles at the locus of the allele k and p_k is the frequency of the allele k in all populations. Note that when 'dist = 'PCA", n is the number of conserved independent principal components and x_{ki} is the value taken by the principal component k in population i.

Value

An object of class `matrix`

Author(s)

P. Savary

References

Bowcock AM, Ruiz-Linares A, Tomfohrde J, Minch E, Kidd JR, Cavalli-Sforza LL (1994). "High resolution of human evolutionary trees with polymorphic microsatellites." *nature*, **368**(6470), 455–457. Excoffier L, Smouse PE, Quattro JM (1992). "Analysis of molecular variance inferred from metric distances among DNA haplotypes: application to human mitochondrial DNA restriction data." *Genetics*, **131**(2), 479–491. Dyer RJ, Nason JD (2004). "Population graphs: the graph theoretic shape of genetic structure." *Molecular ecology*, **13**(7), 1713–1727. Fortuna MA, Albaladejo RG, FernÃ¡ndez L, Aparicio A, Bascompte J (2009). "Networks of spatial genetic variation across species." *Proceedings of the National Academy of Sciences*, **106**(45), 19044–19049. Weir BS, Cockerham CC (1984). "Estimating F-statistics for the analysis of population structure." *evolution*, **38**(6), 1358–1370. Hedrick PW (2005). "A standardized genetic differentiation measure." *Evolution*, **59**(8), 1633–1638. Jost L (2008). "GST and its relatives do not measure differentiation." *Molecular ecology*, **17**(18), 4015–4026.

Examples

```
data(data_ex_genind)
x <- data_ex_genind
D <- mat_gen_dist(x = x, dist = "basic")
```

`mat_geo_dist` *Compute Euclidean geographic distances between points*

Description

The function computes Euclidean geographic distance between points given their spatial coordinates either in a metric projected Coordinate Reference System or in a polar coordinates system.

Usage

```
mat_geo_dist(
  data,
  ID = NULL,
  x = NULL,
  y = NULL,
  crds_type = "proj",
  gc_formula = "vicenty"
)
```

Arguments

`data`	An object of class : • `data.frame` with 3 columns: 2 columns with the point spatial coordinates and another column with point IDs • `SpatialPointsDataFrame`
`ID`	(if `data` is of class `data.frame`) A character string indicating the name of the column of `data` with the point IDs
`x`	(if `data` is of class `data.frame`) A character string indicating the name of the column of `data` with the point longitude
`y`	(if `data` is of class `data.frame`) A character string indicating the name of the column of `data` with the point latitude
`crds_type`	A character string indicating the type of coordinate reference system: • 'proj' (default): a projected coordinate reference system • 'polar': a polar coordinate reference system, such as WGS84
`gc_formula`	A character string indicating the formula used to compute the Great Circle distance: • 'vicenty'(default): Vincenty inverse formula for ellipsoids • 'slc': Spherical Law of Cosines • 'hvs': Harversine formula

Details

When a projected coordinate reference system is used, it calculates classical Euclidean geographic distance between two points using Pythagora's theorem. When a polar coordinate reference system is used, it calculates the Great circle distance between points using different methods. Unless `method = "polar"`, when `data` is a `data.frame`, it assumes projected coordinates by default.

Value

A pairwise matrix of geographic distances between points in meters

Author(s)

P. Savary

Examples

```
# Projected CRS
data(pts_pop_simul)
mat_dist <- mat_geo_dist(data=pts_pop_simul,
             ID = "ID",
             x = "x",
             y = "y")

#Polar CRS
city_us <- data.frame(name = c("New York City", "Chicago",
                               "Los Angeles", "Atlanta"),
                      lat  = c(40.75170,  41.87440,
                               34.05420,  33.75280),
                      lon  = c(-73.99420, -87.63940,
                              -118.24100, -84.39360))
mat_geo_us <- mat_geo_dist(data = city_us,
                           ID = "name", x = "lon", y = "lat",
                           crds_type = "polar")
```

`plot_graph_lg` *Plot graphs*

Description

The function enables to plot graphs, whether spatial or not.

Usage

```
plot_graph_lg(
  graph,
  crds = NULL,
  mode = "aspatial",
  node_inter = NULL,
  link_width = NULL,
  node_size = NULL,
  module = NULL,
  pts_col = NULL
)
```

Arguments

`graph`	A graph object of class `igraph`
`crds`	(optional, default = NULL) If 'mode = 'spatial'', it is a `data.frame` with the spatial coordinates of the graph nodes. It must have three columns : • ID: A character string indicating the name of the graph nodes. The names must be the same as the node names of the graph of class `igraph` (`igraph::V(graph)$name`) • x: A numeric or integer indicating the longitude of the graph nodes. • y: A numeric or integer indicating the latitude of the graph nodes. This argument is not used when 'mode = 'aspatial'' and mandatory when 'mode = 'spatial''.
`mode`	A character string indicating whether the graph is spatial ('mode = 'spatial'') or not ('mode = 'aspatial'' (default))

`node_inter` (optional, default = NULL) A character string indicating whether the links of the graph are weighted by distances or by similarity indices. It is only used when 'mode = 'aspatial'' to compute the node positions with Fruchterman and Reingold algorithm. It can be equal to:

- 'distance': Link weights correspond to distances. Nodes that are close to each other will be close on the figure.
- 'similarity': Link weights correspond to similarity indices. Nodes that are similar to each other will be close on the figure.

`link_width` (optional, default = NULL) A character string indicating how the width of the link is set on the figure. Their width can be:

- inversely proportional to link weights ("inv_w", convenient with distances, default)
- proportional to link weights ("w")

`node_size` (optional, default = NULL) A character string indicating the graph node attribute used to set the node size on the figure. It must be the name of a numeric or integer node attribute from the graph.

`module` (optional, default = NULL) A character string indicating the graph node modules used to set the node color on the figure. It must be the name of a node attribute from the graph with discrete values.

`pts_col` (optional, default = NULL) A character string indicating the color used to plot the nodes (default: "#F2B950"). It must be a hexadecimal color code or a color used by default in R. It cannot be used if 'module' is specified.

Details

When the graph is not spatial ('mode = 'aspatial''), the nodes coordinates are calculated with Fruchterman et Reingold algorithm. The graph object `graph` of class `igraph` must have node names (not necessarily in the same order as IDs in crds, given a merging is done).

Value

A ggplot2 object to plot

Author(s)

P. Savary

References

Fruchterman TM, Reingold EM (1991). "Graph drawing by force-directed placement." *Software: Practice and experience*, **21**(11), 1129–1164.

Examples

```
data(pts_pop_ex)
data(data_ex_genind)
mat_w <- mat_gen_dist(data_ex_genind, dist = "DPS")
gp <- gen_graph_topo(mat_w = mat_w, topo = "mst")
```

```
g <- plot_graph_lg(graph = gp,
                             crds = pts_pop_ex,
                             mode = "spatial",
                             link_width = "inv_w")
```

plot_w_hist *Plot histograms of link weights*

Description

The function enables to plot histogram to visualize the distribution of the link weights

Usage

```
plot_w_hist(graph, fill = "#396D35", class_width = NULL)
```

Arguments

`graph`	A graph object of class `igraph` whose links are weighted
`fill`	A character string indicating the color used to fill the bars (default: "#396D35"). It must be a hexadecimal color code or a color used by default in R.
`class_width`	(default values: NULL) A numeric or an integer specifying the width of the classes displayed on the histogram. When it is not specified, the width is equal to the difference between the minimum and maximum values divided by 80.

Value

A ggplot2 object to plot

Author(s)

P. Savary

Examples

```
data(data_ex_genind)
mat_w <- mat_gen_dist(data_ex_genind, dist = "DPS")
gp <- gen_graph_topo(mat_w = mat_w, topo = "gabriel")
hist <- plot_w_hist(gp)
```

pop_gen_index *Compute population-level genetic indices*

Description

The function computes population-level genetic indices from an object of class `genind`.

Usage

```
pop_gen_index(x, pop_names = NULL, indices = c("Nb_ind", "A", "He", "Ho"))
```

Arguments

`x`	An object of class `genind` from package **adegenet**.
`pop_names`	(optional) A character vector indicating population names. It is of the same length as the number of populations. Without this argument, populations are given the names they have initially in the 'genind' object (which is sometimes only a number). The order of the population names must match with their order in the 'genind' object. The function does not reorder them. Users must be careful.
`indices`	(optional) A character vector indicating the population-level indices to compute. These indices can be: • Mean allelic richness by locus by population (`indices = c("A",...)`) • Mean expected heterozygosity by locus by population (`indices = c("He",...)`) • Mean observed heterozygosity by locus by population (`indices = c("Ho",...)`) • Number of individuals by population (`indices = c("Nb_ind",...)`) • Total allelic richness by population (`indices = c("A_tot",...)`) By default, `indices = c("Nb_ind","A","He","Ho")`.

Value

An object of class `data.frame` whose rows correspond to populations and columns to population attributes (ID, size, genetic indices). By default, the first column corresponds to the population names (ID). The order of the columns depends on the vector 'indices'.

Author(s)

P. Savary

Examples

```
data(data_ex_genind)
x <- data_ex_genind
pop_names <- levels(x@pop)
df_pop_indices <- pop_gen_index(x = x,
                  pop_names = pop_names,
                  indices = c("Nb_ind", "A"))
```

`pts_pop_ex` *pts_pop_ex : details on simulated populations*

Description

Simulation dataset 10 populations located on a simulated landscape

Usage

```
pts_pop_ex
```

Format

An object of class 'data.frame' with the following columns :

ID Population ID of the 10 populations

x Site longitude (RGF93)

y Site latitude (RGF93)

References

Landguth EL, Cushman S (2010). "CDPOP: a spatially explicit cost distance population genetics program." *Molecular Ecology Resources*, **10**(1), 156–161. There are as many rows as there are sampled populations.

Examples

```
data("pts_pop_ex")
str(pts_pop_ex)
```

`pts_pop_simul` *pts_pop_simul : details on simulated populations*

Description

Simulation dataset 50 populations located on a simulated landscape

Usage

```
pts_pop_simul
```

Format

An object of class 'data.frame' with the following columns :

ID Population ID of the 50 populations

x Site longitude (RGF93)

y Site latitude (RGF93)

References

Landguth EL, Cushman S (2010). "CDPOP: a spatially explicit cost distance population genetics program." *Molecular Ecology Resources*, **10**(1), 156–161. There are as many rows as there are sampled populations.

Examples

```
data("pts_pop_simul")
str(pts_pop_simul)
```

`pw_mat_to_df` *Convert a pairwise matrix into an edge-list data.frame*

Description

The function converts a pairwise matrix into an edge-list data.frame

Usage

```
pw_mat_to_df(pw_mat)
```

Arguments

`pw_mat`	A pairwise matrix which can be: • An object of class `matrix`. It must have the same row names and column names. If values represent distances, diagonal elements should be equal to 0. • An object of class `dist`. In that, its column numbers are used to create IDs in the resulting data.frame.

Value

An object of class `data.frame`

Author(s)

P. Savary

Examples

```
data(data_tuto)
pw_mat <- data_tuto[[1]]
df <- pw_mat_to_df(pw_mat)
```

`reorder_mat` *Reorder the rows and columns of a symmetric matrix*

Description

The function reorders the rows and columns of a symmetric matrix according to a specified order.

Usage

```
reorder_mat(mat, order)
```

Arguments

`mat`	An object of class `matrix`
`order`	A character vector with the rows and columns names of the matrix in the order in which they will be ordered by the function. All its elements must be rows and columns names of the matrix `mat`.

Details

The matrix `mat` must be symmetric and have rows and columns names. Its values are not modified.

Value

A reordered symmetric matrix

Author(s)

P. Savary

Examples

```
mat <- matrix(rnorm(36), 6)
mat[lower.tri(mat)] <- t(mat)[lower.tri(mat)]
row.names(mat) <- colnames(mat) <- c("A", "C", "E", "B", "D", "F")
order <- c("A", "B", "C", "D", "E", "F")
mat <- reorder_mat(mat = mat, order = order)
```

`scatter_dist` *Plot scatterplots of genetic distance vs landscape distance*

Description

The function enables to plot scatterplots to visualize the relationship between genetic distance (or differentiation) and landscape distance (Euclidean distance, cost-distance, etc.)between populations or sample sites.

Usage

```
scatter_dist(
  mat_gd,
  mat_ld,
  method = "loess",
  thr_gd = NULL,
  thr_ld = NULL,
  se = TRUE,
  smooth_col = "black",
  pts_col = "#999999"
)
```

Arguments

`mat_gd`	A symmetric `matrix` or `dist` object with pairwise genetic distances between populations or sample sites.
`mat_ld`	A symmetric `matrix` or `dist` object with pairwise landscape distances between populations or sample sites. These distances can be Euclidean distances, cost-distances or resistance distances, among others.
`method`	A character string indicating the smoothing method used to fit a line on the scatterplot. Possible values are the same as with function 'geom_smooth()' from **ggplot2** : 'lm', 'glm', 'gam', 'loess' (default).
`thr_gd`	(optional) A numeric or integer value used to remove values from the data before to plot. All genetic distances values above `thr_gd` are removed from the data.
`thr_ld`	(optional) A numeric or integer value used to remove values from the data before to plot. All landscape distances values above `thr_ld` are removed from the data.
`se`	Logical (optional, default = TRUE) indicating whether the confidence interval around the smooth line is displayed.
`smooth_col`	(optional) A character string indicating the color used to plot the smoothing line (default: "blue"). It must be a hexadecimal color code or a color used by default in R.
`pts_col`	(optional) Character string indicating the color used to plot the points (default: "#999999"). It must be a hexadecimal color code or a color used by default in R.

Details

IDs in `mat_gd` and `mat_ld` must be the same and refer to the same sampling sites or populations, and both matrices must be ordered in the same way. Matrix of genetic distance `mat_gd` can be computed using `mat_gen_dist`. Matrix of landscape distance `mat_ld` can be computed using `mat_geo_dist` when the landscape distance needed is a Euclidean geographical distance.

Value

A ggplot2 object to plot

Author(s)

P. Savary

Examples

```
data(data_tuto)
mat_dps <- data_tuto[[1]]
mat_dist <- suppressWarnings(mat_geo_dist(data = pts_pop_simul,
      ID = "ID",
      x = "x",
      y = "y"))
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                      order(as.character(colnames(mat_dist)))]
scatterplot_ex <- scatter_dist(mat_gd = mat_dps,
                              mat_ld = mat_dist)
```

`scatter_dist_g` *Plot scatterplots of distances to visualize the graph pruning intensity*

Description

The function enables to plot scatterplots of the relationship between two distances (often a genetic distance and a landscape distance between populations or sample sites), while highlighting the population pairs between which a link was conserved during the creation of a graph whose nodes are populations (or sample sites). It thereby allows to visualize the graph pruning intensity.

Usage

```
scatter_dist_g(
  mat_y,
  mat_x,
  graph,
  thr_y = NULL,
  thr_x = NULL,
  pts_col_1 = "#999999",
  pts_col_2 = "black"
)
```

Arguments

mat_y	A symmetric (complete) matrix or dist object with pairwise (genetic or landscape) distances between populations or sample sites. These values will be the point coordinates on the y axis. mat_y is the matrix used to weight the links of the graph x, whose nodes correspond to row and column names of mat_y.
mat_x	A symmetric (complete) matrix or dist object with pairwise (genetic or landscape) distances between populations or sample sites. These values will be the point coordinates on the x axis. mat_x and mat_y must have the same row and column names, ordered in the same way.
graph	A graph object of class igraph. Its nodes must have the same names as the row and column of mat_y and mat_x matrices. x must have weighted links. Link weights have to be values from mat_y matrix. graph must be an undirected graph.
thr_y	(optional) A numeric or integer value used to remove values from the data before to plot. All values from mat_y above thr_y are removed from the data.
thr_x	(optional) A numeric or integer value used to remove values from the data before to plot. All values from mat_x above thr_x are removed from the data.
pts_col_1	(optional) A character string indicating the color used to plot the points associated to all populations or sample sites pairs (default: "#999999"). It must be a hexadecimal color code or a color used by default in R.
pts_col_2	(optional) A character string indicating the color used to plot the points associated to populations or sample sites pairs connected on the graph (default: "black"). It must be a hexadecimal color code or a color used by default in R.

Details

IDs in mat_y and mat_x must be the same and refer to the same sampling sites or populations, and both matrices must be ordered in the same way. Matrices of genetic distance can be computed using mat_gen_dist. Matrices of landscape distance can be computed using mat_geo_dist when the landscape distance needed is a Euclidean geographical distance. This function is based upon scatter_dist function.

Value

A ggplot2 object to plot

Author(s)

P. Savary

Examples

```
data(data_tuto)
mat_gen <- data_tuto[[1]]
mat_dist <- suppressWarnings(mat_geo_dist(data=pts_pop_simul,
      ID = "ID",
      x = "x",
      y = "y"))
mat_dist <- mat_dist[order(as.character(row.names(mat_dist))),
                     order(as.character(colnames(mat_dist)))]
x <- gen_graph_topo(mat_w = mat_gen, mat_topo = mat_dist, topo = "gabriel")
scat <- scatter_dist_g(mat_y = mat_gen, mat_x = mat_dist,
                       graph = x)
```

`structure_to_genind` *Convert a file in STRUCTURE format into a genind object*

Description

The function converts a text file in STRUCTURE format into a genind object to use in R

Usage

```
structure_to_genind(
  path,
  pop_names = NULL,
  loci_names = NULL,
  ind_names = NULL
)
```

Arguments

`path`	A character string indicating the path to the STRUCTURE file in format .txt, or alternatively the name of the file in the working directory. The STRUCTURE file must only have : • A first column with the IDs of the individuals (can be a simple number) • A second column with the IDs of the populations (can be a simple number) • Some loci columns : as many columns as loci in the data The row for loci names is optional but recommended. Each individual is displayed on 2 rows.
`pop_names`	(optional) A character vector indicating the population names in the same order as in the STRUCTURE file. It is of the same length as the number of populations. Without this argument, populations are numbered from 1 to the total number of individuals.
`loci_names`	A character vector with the names of the loci if not specified in the file first row. This argument is mandatory if the STRUCTURE file does not include the names of the loci in the first row. In other cases, the names of the loci is extracted from the file first row
`ind_names`	(optional) A character vector indicating the individual names in the same order as in the STRUCTURE file. It is of the same length as the number of individuals. Without this argument, individuals are numbered from 1 to the total number of individuals.

Details

The column order of the resulting object can be different from that of objects returned by `gstud_to_genind` and `genepop_to_genind`, depending on allele and loci coding This function uses functions from **pegas** package. For details about STRUCTURE file format : STRUCTURE user manual

Value

An object of type `genind`.

Author(s)

P. Savary

Examples

```
data("data_ex_genind")
loci_names <- levels(data_ex_genind@loc.fac)
pop_names <- levels(data_ex_genind@pop)
ind_names <- row.names(data_ex_genind@tab)
path_in <- system.file('extdata', 'data_ex_str.txt',
                       package = 'graph4lg')
file_n <- file.path(tempdir(), "data_ex_str.txt")
file.copy(path_in, file_n, overwrite = TRUE)
str <- structure_to_genind(path = file_n, loci_names = loci_names,
                           pop_names = pop_names, ind_names = ind_names)
file.remove(file_n)
```

Index