

Package: MPSEM (via r-universe)

September 23, 2026

Type Package
Encoding UTF-8
Title Modelling Phylogenetic Signals using Eigenvector Maps
Version 0.6-1
Date 2025-03-26
Description Computational tools to represent phylogenetic signals using adapted eigenvector maps.
Depends R (>= 3.0.0), ape, magrittr, sf
Imports MASS, expm
Suggests knitr, caper, xfun
License GPL-3
LazyLoad yes
NeedsCompilation yes
VignetteBuilder knitr
RoxygenNote 7.3.1
Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev
Repository <https://guenardg.r-universe.dev>
Date/Publication 2025-04-07 19:34:47 UTC
RemoteUrl <https://github.com/guenardg/mpsem>
RemoteRef HEAD
RemoteSha 6c44d72d81ba7c73b5c3acaa1fa4e6d9322d951f

Contents

MPSEM-package	2
concatenate	3
dstGraph	4
graph-class	7

graph-functions	13
graph-purge	15
graph-utils	17
graphModplot	21
harmonicMean	22
InflMat-class	24
invDistWeighting	26
lm-utils	27
molEvolSim	29
MPSEM-generics	36
PEM-class	38
PEM-functions	45
PEM-utils	46
penlm-class	49
randomGraph	52
show.sequence	54
traitEvolSim	55
write.fasta	62

Index 64

MPSEM-package	<i>Modelling Phylogenetic Signals using Eigenvector Maps</i>
---------------	--

Description

Computational tools to represent phylogenetic signals using adapted eigenvector maps.

Details

Phylogenetic eigenvector maps (PEM) is a method for using phylogeny to model features of organism, most notably quantitative traits. It consists in calculating sets of explanatory variables (eigenvectors) that are meant to represent different patterns in trait values that are likely to have been induced by evolution. These patterns are used to model the data, using a linear model for instance.

If one is interested in a ‘target’ species (i.e. a species for which the trait value is unknown), and provided that we know the phylogenetic relationships between that species and those of the model, the method allows us to obtain the scores of that new species on the phylogenetic eigenfunctions underlying a PEM. These scores are used to make empirical predictions of trait values for the target species on the basis of those observed for the species used in the model.

The package provides low-level utility functions for performing operations on graphs (see [graph-functions](#)), calculate influence matrix ([InflMat](#)), and so on.

A phylogenetic modelling tutorial using MPSEM is available as a package vignette. See example below.

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

See Also

Makarenskov, V., Legendre, P. & Desdevisse, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89-96

Blanchet, F. G., Legendre, P. & Borcard, D. 2008. Modelling directional spatial processes in ecological data. *Ecological Modelling* 215: 325-336

Examples

```
## To view MPSEM tutorial
vignette("MPSEM", package="MPSEM")
```

concatenate	<i>Sequence Concatenation</i>
-------------	-------------------------------

Description

A function to transform a [raw](#) matrix, used to store sets of single nucleotides, into a vector of strings, optionally removing specific value(s).

Usage

```
concatenate(x, discard = NULL)
```

Arguments

x	A raw vector storing the molecular traits (single nucleotides, gaps, and so on).
discard	A vector of characters to exclude from the results (default: NULL, meaning nothing is excluded).

Details

That function concatenates the rows of a [raw](#) matrix, which is used for storing the molecular trait values, into a vector of strings. If argument discard is given a set of values, they will be excluded from the resulting strings.

Value

A vector of character strings.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

Examples

```
## Define a raw vector for storing nuceotide values:
c(Sequence_1 = "ATCG-TTTCG--CCCCA--TTA--TTAA-GTAA-GTAATCTTTCA",
  Sequence_2 = "TTGGCTTCC--TC--CAT-TTC--TTCA-GT-ACG-ATTCTTTTA",
  Sequence_3 = "TTCG-TACC-T-T---A-ATAA--T-AA-GTCTTGTAAATCGTTCA") %>%
  sapply(charToRaw) %>%
  t -> sqn

## Display the raw sequence:
sqn

## Transforming the sequence to character strings
concatenate(sqn)

## Transforming the sequence to character strings without the gaps:
concatenate(sqn, discard="-")

## Clean-up:
rm(sqn)
```

dstGraph	<i>Distance-Based Directed Graph</i>
----------	--------------------------------------

Description

Calculates a distance-based directed graph from a dissimilarity matrix, a threshold value, and an origin (or root) vertex.

Usage

```
dstGraph(d, th, origin, stretch)
```

Arguments

- d A dissimilarity matrix such as the one obtained from [dist](#) or [dist.dna](#).
- th Numeric. A threshold value for dissimilarity. Vertices are considered as connected whenever their pairwise dissimilarity value is smaller or equal to that value.

origin	Integer. Index of the origin vertex from which the edges are directed.
stretch	Numeric (optional). When a vertex is unreachable, stretch the threshold value for the shortest edge connecting it to the rest of the graph up to that value.

Details

The algorithm

Beginning on a user-defined origin vertex, the algorithm proceeds by connecting all vertices within a given dissimilarity value from the ones that have already been connected, until all the vertices that can be reached has been reached. Optionally, the dissimilarity threshold value can be stretched for the vertices that are unreachable. Vertices that cannot be reached in any way are reported by the function.

Value

A `graph-class` object.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

- Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131
- Makarenkov, V., Legendre, L. & Desdise, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89-96

Examples

```
## We set the seed to obtain a consistent example, but you are welcome to
## experiment with other graphs.
set.seed(7653401)

## Here, the dissimilarity matrix is generated from the Euclidean distance of
## a two-dimensional plot for the sake of simplicity. In practice, the matrix
## will come from DNA data using a dissimilarity method such as those
## implemented by ape packages's function dist.dna().

N <- 100
coords <- cbind(x=runif(N,-1,1), y=runif(N,-1,1))
rownames(coords) <- sprintf("N%d",1:N)
dst <- dist(coords)

## Calculate the distance-based graph:
gr <- dstGraph(d=dst, th=0.25, origin=15)

## This graph have unconnected vertices.
```

```

## Plotting the graph with colors indicating the order of the edges:
plot(coords, type="n", asp=1)
col <- head(rainbow(max(gr$order) + 1), max(gr$order))
arrows(
  x0 = coords[edge(gr)[[1]],1],
  x1 = coords[edge(gr)[[2]],1],
  y0 = coords[edge(gr)[[1]],2],
  y1 = coords[edge(gr)[[2]],2],
  length = 0.05,
  col = col[gr$order[edge(gr)[[2]]]]
)
points(coords, pch=21, bg="black", cex=0.25)

## Try again raising the threshold to help in connecting all the vertices:
gr <- dstGraph(d=dst, th=0.28, origin=15)

## It helped, but does not entirely solve the matter.

plot(coords, type="n", asp=1)
col <- head(rainbow(max(gr$order) + 1), max(gr$order))
arrows(
  x0 = coords[edge(gr)[[1]],1],
  x1 = coords[edge(gr)[[2]],1],
  y0 = coords[edge(gr)[[1]],2],
  y1 = coords[edge(gr)[[2]],2],
  length = 0.05,
  col = col[gr$order[edge(gr)[[2]]]]
)
points(coords, pch=21, bg="black", cex=0.25)

## Try again while stretching the threshold for the unconnected vertices:
gr <- dstGraph(d=dst, th=0.28, origin=15, stretch=0.5)

## All the vertices are now connected.

plot(coords, type="n", asp=1)
col <- head(rainbow(max(gr$order) + 1), max(gr$order))
arrows(
  x0 = coords[edge(gr)[[1]],1],
  x1 = coords[edge(gr)[[2]],1],
  y0 = coords[edge(gr)[[1]],2],
  y1 = coords[edge(gr)[[2]],2],
  length = 0.05,
  col = col[gr$order[edge(gr)[[2]]]]
)
points(coords, pch=21, bg="black", cex=0.25)

## This is the influence matrix of that directed graph:
tmp <- InflMat(gr)

## An image plot of this influence matrix:
image(t(tmp[nrow(tmp):1L,]), col=gray(c(1,0)), asp=1)

```

Description

Class and methods to handle MPSEM graphs.

Usage

```
## S3 method for class 'graph'
print(x, ...)

## S3 method for class 'graph'
nedge(x)

## S3 method for class 'graph'
geometry(x)

## S3 replacement method for class 'graph'
geometry(x) <- value

## S3 replacement method for class 'graph'
x$name <- value

## S3 method for class 'graph'
as.phylo(x, ...)

## S3 method for class 'phylo'
as.graph(x, ...)

## S3 method for class 'graph'
plot(
  x,
  y,
  ylim,
  use.geometry = TRUE,
  pch = 21L,
  bg = "white",
  cex.min = 2,
  cex.max = cex.min,
  cex.lab = cex.min/3,
  axes = FALSE,
  xlab = "",
  ylab = "",
```

```

    edge.color = list("black", "red"),
    length = 0.05,
    code = 2L,
    show.vertex.labels = TRUE,
    direction = c("rightwards", "leftwards", "upwards", "downwards"),
    ...
)

## S3 method for class 'graph'
locate(x, target, ...)

```

Arguments

<code>x</code>	A graph-class object.
<code>...</code>	Additional parameters to be internally passed to other functions or methods.
<code>value</code>	A vector or data.frame containing the values to be given to the graph-class object.
<code>name</code>	A literal character string or a name .
<code>y</code>	An optional set of numeric value available at the vertices to be shown using marker of different sizes or background colors.
<code>ylim</code>	A length 2 numeric vector giving minimum and maximum bounds for <code>y</code> . It is only pertinent when <code>y</code> is given. When omitted, it is estimated from the range of <code>y</code> values.
<code>use.geometry</code>	A logical value specifying whether to plot the graph using its specified geometry, if available (default: TRUE). When no geometry is available or if <code>use.geometry = FALSE</code> , the graph is plotted using a back bone tree (obtained using the <code>as.phylo</code> method) with added supplementary edges.
<code>pch</code>	An integer. Graphical parameter <code>pch</code> used internally by function points (default: 21, a round marker).
<code>bg</code>	Either a single character string specifying a single background color or a vector of colors used as a color scale to display the values of <code>y</code> (default: "white").
<code>cex.min</code>	A numeric value giving the marker size used for the smallest value of <code>y</code> or for all markers when <code>y</code> is NULL (default: 2).
<code>cex.max</code>	A numeric value. When <code>y</code> is not NULL, the marker size used for the largest value of <code>y</code> (default: <code>cex.min</code>).
<code>cex.lab</code>	A numeric value specifying the size of the vertex labels (default: <code>cex.min/3</code>).
<code>axes</code>	A logical; whether to show the axis graduation of the plot (default: FALSE).
<code>xlab</code>	A character string; the title of the abscissa of the plot (default: "").
<code>ylab</code>	A character string; the title of the ordinates of the plot (default: "").
<code>edge.color</code>	A list of two colors or two color vectors (mode: character) specifying the color (or colors) of the edges. The first elements is used for the backbone tree when not using the geometry or all edges when using the geometry. The second element is only used to specify the color (or colors) of the supplementary edges when not using the geometry. The default is <code>list("black", "red")</code> .

length	A numeric value giving the size of the arrow heads (in inches, Default: 0.05).
code	An integer determining kind of arrows to be drawn (see arrows for the details, default: 2).
show.vertex.labels	A logical specifying whether to plot the vertex labels (default: TRUE).
direction	One of character strings "rightwards" (the default), "leftwards", "upwards", or "downwards", or any unambiguous thereof, specifying the direction of the plot, when not using the geometry.
target	An integer or character vector containing the indices or names to be specified as target vertices.

Format

Minimally, a graph-class object contains a [data.frame](#) object that contains vertex properties (or a zero-column data.frame when there is no vertex property), to which it adds a edge attribute. The edge attribute is an other [data.frame](#) object with its first two integer columns called from and to that contain the indices of the origin and destination vertices, respectively, for each of the edges in addition to any number of edge properties.

A vertex property commonly found in a graph-class object is a [logical](#) property called 'species' that identifies which vertex has trait values. Also, an edge property commonly found in a graph-class object is a numeric property called 'distance' that gives the evolutionary distance (or length) associated with the edges. These two additional properties are mandatory for calculating a [PEM](#), but they may bear different names than the ones previously mentioned.

Details

Prints user-relevant information about the graph: number of edges and vertices, edge and vertex labels, addition edge properties and vertex properties.

Function `as.graph()` uses the MPSEM graph functions to convert a phylogenetic tree of class 'phylo' (see [ape-package](#)) to a [graph-class](#) object. It recycles tip labels. It also creates default node labels if they were absent from the 'phylo' object, and uses them as vertex labels. The resulting graph can then be edited to represent cases that do not strictly conform to a tree topology.

The method `as.phylo()` convert a graph-class object into a class 'phylo' objects as defined in package `ape`. The function ensures that all vertices have only a single incoming edge. When multiple incoming edges are found, the one originating from the vertex that is the closer from the origin (or root) of the graph is kept and any other is discarded. The discarded edges are gathered into a data frame which is appended to the returned class 'phylo' object as an attribute called `discarded`. Another attribute called `subst` is also appended to the returned object; it contains the indices of the graph vertices as they appear in the returned class 'phylo' object.

The plot method follows two possible approaches. The graph may be assigned a geometry (see [sf-package](#)), in which case the latter is used to plot the graph provided that argument `use.geometry = TRUE`, which is the default. If the graph has no geometry (or if it has one, but argument `use.geometry = FALSE`), the graph is plotted as an augmented tree (or as a regular tree if it is a non-reticulated graph). This tree is obtained using the `as.phylo` method for graph-class objects herein described. #' Package `ape`'s internal functions are used to determine the coordinates of the plot. For a reticulated graph, the discarded edges are added to the backbone tree obtained from the `as.phylo` method,

thereby resulting in an augmented tree. The plot method also has the possibility to display trait values at the vertices using maker sizes.

Evolutionary distances (ie., edge lengths) are abstract measurements that may represent time (eg., ka, Ma), the number of generations, or any relevant metric of the evolutionary processes occurring along the edges of the evolutionary graph. This information can be included in the graph as an edge property (see 'Format').

Value

print NULL.

nedge An integer; the number of edges in the graph.

edge A data frame of the edge information, namely the origin and destination of each edge (integers), and, possibly, additional edge features.

edgenames A vector of type 'character' giving the name of each of the edges.

geometry A geometry object (e.g., points, lines, polygons).

as.phylo An object of class 'phylo', possibly with an attribute called 'discarded' containing the edges that had been discarded in order to coerce the graph into a tree and (in every cases) an attribute called 'subst' containing an integer vector giving the indices of the vertices on the tree plot.

plot When the geometry is used, the graph object given through argument x, when the geometry is not used, the object of class 'phylo' produced internally by the as.phylo method, with an attribute called 'xy' containing the display coordinates of the tree plot.

locate A list with two elements: a graph-class object (\$x) and a data frame (\$location). The graph-class object is the residual graph after the targets have been removed and has two [logical](#) attributes; one called "removedVertex" used to identify any vertex of the input graph (argument x) that is no longer found in the residual graph, and one called "removedEdge" used to identify any edge of the input graph that is no longer found in the residual graph. The data frame has three columns; a first column called 'ref' giving the index of the vertex or edge where each target can be found, a second column called 'dist' giving the (evolutionary) distance along the edge where the target can be found (NA for a target located at a vertex) and 'lca' giving the distance between the latest common ancestor of the target in the graph and the target itself.

Functions

- **print(graph)**: Print Graph
A print method for graph-class objects.
- **nedge(graph)**: Number of Edges
Get the number of edges in a graph.
- **geometry(graph)**: Extract Geometry
Extracts a geometry from a graph-class object.
- **geometry(graph) <- value**: Assign Geometry
Assigns a geometry to a graph-class object.
- **`\$` (graph) <- value**: Insert or Replace a Vertex Property
Insert or replace a vertex property into a graph-class object.

- `as.phylo(graph)`: Transformation to a Tree
An `as.phylo` method to transforms a graph-class object into a "phylo" class object.
- `as.graph(phylo)`: Transformation From a Tree Into a Graph
An `as.graph` method to transforms a "phylo" class object into a graph-class object.
- `plot(graph)`: Plot Graph
A plotting method for graph-class objects.
- `locate(graph)`: Locate Graph Targets
Calculate target locations and residual graph from an initial graph and a set of targets.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

See Also

`graph-functions`, `plot.phylo`, and `plot.sf`.

Examples

```
## Create an exemplary graph:
data.frame(
  species = rep(TRUE,13),
  type = c(2,2,3,1,2,2,2,2,2,3,3,3),
  x = c(1,3,4,0,1.67,4,1,1.33,2.33,3.33,4.33,4,5),
  y = c(1,1,1,0,0.5,0,-1,0,0,-0.5,-1,-0.5,-0.5),
  row.names = sprintf("V%d",1:13)
) %>%
  st_as_sf(
    coords=c("x","y"),
    crs = NA
  ) %>%
  graph %>%
  add.edge(
    from = c(1,2,1,5,4,4,5,9,4,8,9,4,7,7,6,6,9,10,10),
    to = c(2,3,5,2,1,5,9,2,8,9,6,7,8,9,3,13,10,12,11),
    data = data.frame(
      distance = c(4.2,4.7,3.9,3.0,3.6,2.7,4.4,3.4,3.6,3.3,
        4.8,3.2,3.5,4.4,2.5,3.4,4.3,3.1,2.2),
      row.names = sprintf("E%d",1:19)
    )
  ) -> x

## The object appears as follows:
```

```

x

## The number of vertices is the number of rows in the data frame:
nrow(x)

## The number of vertex properties is the number of columns in the data
## frame:
ncol(x)

## Methods defined for data frames are applicables to graph-class objects:
dim(x)
rownames(x)
colnames(x)
dimnames(x)
labels(x)
names(x)
as.data.frame(x)

## MPSEM defines new generics and implements methods for them:
nedge(x)      ## To obtain the number of edges
edge(x)       ## To obtain the edge data frame
edgenames(x)  ## To obtain the names of the edges
geometry(x)   ## To obtain any geometry associated with the edges

edgenames(x) <- NULL
edgenames(x)

edgenames(x) <- sprintf("E_%d", 1:nedge(x))
edgenames(x)
edge(x)

## Plotting the graph:
plot(x)
plot(x, use.geometry=FALSE)

geom <- geometry(x)
geometry(x) <- NULL    ## Removing the geometry.
plot(x)

geometry(x) <- geom    ## Putting the geometry back into place.
plot(x)

## The graph is transformed or coerced into a phylogenetic tree as follows:
phy <- as.phylo(x)
plot(phy, show.node.label=TRUE)

## A phylogenetic tree with 7 tips and 6 internal nodes:
tree1 <- read.tree(
  text=paste(
    "(((A:0.15,B:0.2)N4:0.15,C:0.35)N2:0.25,((D:0.25,E:0.1)N5:0.3,",
    "(F:0.15,G:0.2)N6:0.3)N3:0.1)N1:0.1;", sep="")

```

```
## Default: excluding the root vertex:
as.graph(tree1)

## Including the root vertex:
as.graph(tree1, includeRoot = TRUE)
```

graph-functions

*MPSEM Graph Manipulation Functions***Description**

A set of primitive functions for creating and manipulating MPSEM graphs.

Usage

```
graph(data = data.frame())

add.vertex(x, data = data.frame())

add.edge(x, from, to, data = data.frame())

rm.edge(x, id)

rm.vertex(x, id)
```

Arguments

data	A data.frame providing data (and labels) to be assigned to the vertices or edges. It can be an empty data frame with row names providing the vertex labels.
x	A graph-class object.
from	An integer or character vector. References to the origin of the edges to be added.
to	An integer or character vector. The destinations of the edges to be added (vertex references).
id	An integer or character vector. The identity of vertex or edge to be removed (references).

Details

A new graph is populated with vertices using function `graph()`. The function must be provided with a data frame. This data frame may be a 0 column data frame as long as its `row.names` attribute contains the number of elements necessary to reference the vertices. Additional vertices can be added later with function `add.vertex()`. The graph thus created contains no edge; the latter are added using function `add.edge()`. Edges and vertices are removed using functions `rm.edge()` and `rm.vertex()`, respectively.

Functions

- `graph()`: Create Graph
Create a graph and populates it with vertices.
- `add.vertex()`: Add Vertices
Add vertices to an existing graph.
- `add.edge()`: Add Edges
Add edges to a graph.
- `rm.edge()`: Remove Edges
Remove edges from a graph.
- `rm.vertex()`: Remove Vertices
Remove vertices from a graph.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

Makarek, V., Legendre, L. & Desdevisse, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89-96

Blanchet, F. G., Legendre, P. & Borcard, D. 2008. Modelling directional spatial processes in ecological data. *Ecological Modelling* 215: 325-336

Examples

```
## Populate a graph with 7 vertices labeled A-G and edge properties x and y:
data.frame(
  quantitative = c(1.1,7.2,7.2,4.1,5.5,6.9,3.3),
  factor = factor(c("A","A","A","B","B","B","B")),
  row.names = c("A","B","C","D","E","F","G"))
) %>%
  graph -> x

## Note from package magrittr:
## x %>% f(y, z) is equivalent to f(x, y, z)
## x %<>% f(y, z) is equivalent to x <- f(x, y, z)

## Add three vertices without descriptors:
x %>% add.vertex(
  data = data.frame(row.names = c("H","I","J"))
)

## This is another way to add vertices:
x %<>% add.vertex(
```

```

    data = data.frame(
      factor = factor(c("C", "C", "C")),
      ordered = ordered(c(1,2,2)),
      row.names = c("H", "I", "J")
    )
  )

## Adding 10 edges, labeled E1-E10 and with properties d and r, to the graph:
x %<>% add.edge(
  from = c("A", "B", "B", "C", "C", "D", "D", "E", "E", "F"),
  to = c("A", "C", "D", "E", "F", "F", "G", "H", "I", "J"),
  data = data.frame(
    distance = c(1,5,2,3,3,2,5,1,1,1),
    reversible = c(TRUE,FALSE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE),
    row.names = paste("E",1:10,sep="")
  )
)

## Adding three more edges, this time without variable 'reversible', but
## adding a variable called 'factor':
x %<>% add.edge(
  from = c("E", "F", "G"),
  to = c("A", "B", "C"),
  data.frame(
    distance = c(2,3,1),
    factor = factor(c("S", "S", "G")),
    row.names = c("E1", "E11", "E23")
  )
)

## Removing two edges (E3 and E5):
x %<>% rm.edge(id=c("E3", "E5"))

## Removing vertices B, F, and G with their associated edges:
x %<>% rm.vertex(id=c("B", "F", "G"))

```

graph-purge

MPSEM Graph Purging Functions

Description

A set of functions for purging possibly uninformative edges from MPSEM graphs.

Usage

```
purge.terminal(x, ...)
```

```
purge.median(x, combine = function(x, ...) 1/sum(1/x), ...)
```

Arguments

x	A graph-class object.
...	Further argument to be internally passed to the function given as argument combine or function <code>graphDist</code> .
combine	A function for combining the distances of redundant edges, should they occur. The default function calculates the inverse of the sum of the inverse distances.

Details

Function `purge.terminal` or `purge.median` will only purge the vertices that are not marked as species. Removal follows certain constraints. Function `purge.terminal` removes the terminal vertices, irrespective of the number of incoming edges. Function `purge.median` removes the median vertices (vertices with a single incoming edge and a single outgoing edge) by joining the incoming and outgoing edges into a single edge (having the name of the incoming edge). When joining is done, care is taken that any edge having the same origin vertex and destination vertex be consolidated. The default function for consolidating the distances is the inverse of the sum of the inverse distances. For all the other edge characteristics, values of the incoming edge involved in the last removal is taken.

The latter are vertices that have only a single incoming edge and a single outgoing edge, whereas the former are vertices that have no outgoing edges. These vertices are generally uninformative for phylogenetic modelling and do not carry known trait values; thus making them safe for removal.

Value

The purged graph-class object, possibly with attributes `removedVertex` (whenever vertices has to be removed) and/or `removedEdge` (whenever edges had to be removed).

Functions

- `purge.terminal()`: Purge Terminal Vertices
Attempts to purge the terminal vertices of a graph that are not marked as species.
- `purge.median()`: Purge Median Vertices
Attempts to purge the median vertices of a graph that are not marked as species, connecting the two end vertices.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

Examples

```
## A 16-vertex graph with 24 edges:
data.frame(
  species = as.logical(c(1,0,0,0,1,0,0,0,1,1,1,1,1,0,0,0)),
  x = c(1,3,4,0,1.67,4,1,1.33,2.33,3.33,4.33,4,5,5,5,2.33),
  y = c(1,1,1,0,0.5,0,-1,0,0,-0.5,-1,-0.5,-0.5,1,0.5,-1),
```

```

    row.names = sprintf("V%d",1:16)
  ) %>%
  st_as_sf(
    coords=c("x","y"),
    crs = NA
  ) %>%
  graph %>%
  add.edge(
    from = c(1,2,1,5,4,4,5,9,4,8,9,4,7,7,3,6 ,9 ,10,10,3 ,3 ,7 ,9, 10),
    to =   c(2,3,5,2,1,5,9,2,8,9,6,7,8,9,6,13,10,12,11,14,15,16,16,16),
    data = data.frame(
      distance = c(4.2,4.7,3.9,3.0,3.6,2.7,4.4,3.4,3.6,3.3,4.8,3.2,3.5,
                    4.4,2.5,3.4,4.3,3.1,2.2,2.1,0.9,1.0,2.1,0.9),
      row.names = sprintf("E%d",1:24)
    )
  ) -> gr1

## Plotting the exemplary graph:
plot(gr1)

## Purging the terminal vertices:
tmp <- purge.terminal(gr1)
plot(tmp)
attr(tmp,"removedVertex")
attr(tmp,"removedEdge")

## Purging the median vertices:
tmp2 <- purge.median(tmp)
plot(tmp2)
attr(tmp2,"removedVertex")
attr(tmp2,"removedEdge")

```

graph-utils

Graph Utility Functions

Description

A suite of graph utility functions.

Usage

```
getOrigin(x)
```

```
getConnected(x)
```

```
getNonConnected(x)
```

```
getTerminal(x)
```

```

getMedian(x)

isTree(x)

isDivergent(x)

isLinear(x)

graphDist(
  x,
  ...,
  mean = c("arithmetic", "geometric", "harmonic"),
  weighting = invDistWeighting
)

reorderGraph(x, order, shuffleEdge = FALSE)

```

Arguments

x	A graph-class object.
...	Further argument(s) to be passed to the weighting function described below.
mean	One of character strings "arithmetic", "geometric", "harmonic", or any unambiguous abbreviation thereof, specifying the type of mean used for averaging the distances when a vertex has multiple ascendent edges.
weighting	A weighting function; it takes a set of distances as its first argument and returns a set of weights summing to 1 (default: invDistWeighting).
order	An integer vector of the vertex indices.
shuffleEdge	A Boolean. Whether to randomly shuffle the order that the edges are stored in the graph-class object (FALSE).

Details

A origin vertex is one having only outgoing edge(s) and no incoming edge, whereas a terminal vertex is one having only incoming edge(s) and no outgoing edge. A median vertex has one incoming edge and one outgoing edge. A non-connected vertex has no edge, whereas a connected vertex may have incoming edge(s), outgoing edge(s), or both.

Reordering a graph with a `processOrder` attribute will come with a recalculation of the process order, whereas doing so on a graph with a `dist` attribute cause the pairwise distance matrix to also be reordered.

Value

getOrigin A vector of integer.

getConnected A vector of integer.

getNonConnected A vector of integer.

getTerminal A vector of integer.

getMedian A vector of integer.

isTree A logical stipulating whether the graph is a tree.

isDivergent A logical stipulating whether the graph has divergence.

isLinear A logical stipulating whether the graph is a linear sequence of vertices.

graphDist A pairwise distance matrix such as the one obtained from function [dist](#).

Functions

- **getOrigin():** Get Origin Vertex
Obtain the origin vert(ex/ices) of a directed graph; an origin vertex is one with no incoming edge.
- **getConnected():** Get Connected Vertex
Obtain the connected vert(ex/ices) of a graph.
- **getNonConnected():** Get Non-connected Vertex
Obtain the non-connected connected vert(ex/ices) of a graph.
- **getTerminal():** Get Terminal Vertex
Obtain the terminal vert(ex/ices) of a directed graph; a terminal vertex is one with no outgoing edge.
- **getMedian():** Get Terminal Vertex
Obtain the median vert(ex/ices) of a directed graph; a median vertex is one with one incoming edge and one outgoing edge.
- **isTree():** Tree Test
Testing whether the graph is a tree.
- **isDivergent():** Divergence Test
Testing whether the graph has divergence.
- **isLinear():** Linearity Test
Testing whether the graph is a linear sequence.
- **graphDist():** Graph Distance Matrix
Obtain a matrix of the (average) graph distance among the vertices.
- **reorderGraph():** Reorder Vertices
Reorder the vertices of a directed graph.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

See Also

[graph-class](#).

Examples

```
## Create an example graph with 10 vertices and 16 edges:
data.frame(
  species = rep(TRUE,10),
  x = c(2,3,2,4,3,4,2,1,1,0),
  y = c(-2,1,2,0,-0.5,-2,0,-1,1,0),
  row.names = sprintf("V%d",1:10)
) %>%
graph %>%
add.edge(
  from = c(10,10,9,9,8,8,3,7,7,10,2,2,5,1,4,5),
  to = c(9,8,3,7,7,1,2,2,5,2,1,4,4,4,6,6),
  data = data.frame(
    distance = c(1,1,1,1,1,1,1,1,1,4,2,1,1,3,1,1),
    row.names = sprintf("E%d",1:16)
  )
) -> x

getOrigin(x)      ## The graph has a single origin vertex.
getConnected(x)   ## All the vertices
getNonConnected(x) ## are connected.
getTerminal(x)    ## The graph has a single terminal vertex.
getMedian(x)      ## The graph has a single median vertex.
isTree(x)         ## The graph is not a tree.
isDivergent(x)    ## The graph has divergences.
isLinear(x)       ## The graph is not a linear vertex sequence.

## The average pairwise distances between the vertices:
graphDist(x)

## Reordering of the vertices:
xr <- reorderGraph(x, order=c(5:1,8,6,7,10,9))
xr

getOrigin(xr)     ## Same origin vertex, but at a different index.
getTerminal(xr)   ## Same terminal vertex, but at a different index.
graphDist(xr)     ## Same distances, but in a different order.

## Comparison between distances obtained using various means and weighting
## parameter:
cmpr <- function(x, y) 200*(x - y)/(x + y)

cmpr(graphDist(x), graphDist(x, mean="geo"))
cmpr(graphDist(x), graphDist(x, mean="har"))
cmpr(graphDist(x), graphDist(x, mean="ari", a=1))
cmpr(graphDist(x), graphDist(x, mean="geo", a=1))
cmpr(graphDist(x), graphDist(x, mean="har", a=1))
cmpr(graphDist(x), graphDist(x, mean="ari", a=2))
cmpr(graphDist(x), graphDist(x, mean="geo", a=2))
cmpr(graphDist(x), graphDist(x, mean="har", a=2))
```

Description

Interactively modify the plot of a graph object by manually changing the location where vertices are being placed.

Usage

```
graphModplot(
  x,
  col = "grey80",
  bg = c("red", "black", "blue"),
  pch = 21L,
  length = 0.05,
  pt.cex = 0.75,
  ...
)
```

Arguments

<code>x</code>	A graph-class object.
<code>col</code>	Color of the arrows representing the edges. Default is "grey80".
<code>bg</code>	Background colors for the origin, intermediate, and terminal vertices, respectively. Default is <code>c("red", "black", "blue")</code> .
<code>pch</code>	Plotting 'character' (see points) for the details. Default is 21 (a solid dot with a colored background).
<code>length</code>	Length of the edge of the arrow head (see arrows). The default value is 0.05.
<code>pt.cex</code>	The relative point size used for the vertex markers. The default value is 0.75.
<code>...</code>	Additional parameters to be passed to other functions of methods.

Details

After calling the function, a mouse-click on the display will select the nearest vertex. Which one has been selected is indicated by the background of its marker turning white, with its incoming edges appearing as red dashed segments and its outgoing edges appearing as blue dashed segments. A second mouse-click will cause the vertex to be relocated at the location of the mouse pointer.

Value

A point geometry object.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

See Also

[graph-class](#).

Examples

```
## Create an exemplary graph:
data.frame(
  species = rep(TRUE,13),
  type = c(2,2,3,1,2,2,2,2,2,3,3,3),
  x = c(1,3,4,0,1.67,4,1,1.33,2.33,3.33,4.33,4,5),
  y = c(1,1,1,0,0.5,0,-1,0,0,-0.5,-1,-0.5,-0.5),
  row.names = sprintf("V%d",1:13)
) %>%
  st_as_sf(
    coords=c("x","y"),
    crs = NA
  ) %>%
  graph %>%
  add.edge(
    from = c(1,2,1,5,4,4,5,9,4,8,9,4,7,7,6,6,9,10,10),
    to = c(2,3,5,2,1,5,9,2,8,9,6,7,8,9,3,13,10,12,11),
    data = data.frame(
      distance = c(4.2,4.7,3.9,3.0,3.6,2.7,4.4,3.4,3.6,3.3,
        4.8,3.2,3.5,4.4,2.5,3.4,4.3,3.1,2.2),
      row.names = sprintf("E%d",1:19)
    )
  ) -> x

## Original coordinates:
plot(x)

## Edit the vertex locations manually:
x <- graphModplot(x)

## Plot with the new coordinates
plot(x)
```

Description

Functions to calculate the harmonic mean and weighted harmonic mean of a set of data.

Usage

```
hmean(x)
```

```
weighted.hmean(x, w)
```

Arguments

x	A numeric vector containing a set of observations.
w	A numeric vector containing the observation weights.

Details

The (weighted) harmonic mean of a data set is the inverse of the (weighted) mean of the inverse values. As such, any 0 value in the data (or in the non-zero weighted data in the weighted case) will result in the functions returning a value of 0. It is worth recall that whereas the value of the arithmetic mean is strongly influenced by even a few extreme values, the value of the harmonic mean is strongly influenced by even a few small values. The function take any numeric values. However, the harmonic mean of data containing both positive and negative values is unlikely to be stable.

Value

A numeric value.

Functions

- `hmean()`: Harmonic Mean.
Calculates the (unweighted) harmonic mean.
- `weighted.hmean()`: Weighted Harmonic Mean
Calculates the weighted harmonic mean.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

InflMat-class

*Influence Matrix***Description**

Functions and methods calculate and manipulate graph influence matrix.

Usage

```
InflMat(x)

## S3 method for class 'InflMat'
print(x, ...)

## S3 method for class 'InflMat'
nedge(x)

## S3 method for class 'InflMat'
edge(x)

## S3 replacement method for class 'InflMat'
edge(x) <- value

## S3 method for class 'InflMat'
edgenames(x)

## S3 replacement method for class 'InflMat'
edgenames(x) <- value
```

Arguments

x	A graph-class or InflMat-class object.
...	Further arguments to be passed internally to other functions or methods.
value	A vector or data.frame containing the values to be given to the InflMat-class object.

Value

The returned value depends on the function:

InflMat A binary influence matrix of the graph with as many rows as its number of vertices and as many columns as its number of edges.

Functions

- `InflMat()`: Influence Matrix
Calculates the influence matrix of a phylogenetic graph. The influence matrix is a binary matrix whose rows and columns correspond to the vertices and edges of the phylogenetic graph, respectively, and whose elements describe whether a given edge had been taken by any ancestors of a vertex (representing extinct or extant species) during evolution (value = 1) or not (value = 0).
- `print(InflMat)`: Print Graph
A print method for InflMat-class objects.
- `nedge(InflMat)`: Number of Edges
Get the number of edges in an InflMat-class object.
- `edge(InflMat)`: Edge Extraction
Extracts the edges of an InflMat-class object.
- `edge(InflMat) <- value`: Edge Assignment
Assigns edges to an InflMat-class object.
- `edgenames(InflMat)`: Edge Names Extraction
Extracts the edge names of an InflMat-class object.
- `edgenames(InflMat) <- value`: Edge Names Assignment
Assigns edge names to an InflMat-class object.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) – Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

- Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution*. 4: 1120–1131
- Makarenkov, V., Legendre, L. & Desdevisse, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89–96
- Blanchet, F. G., Legendre, P. & Borcard, D. 2008. Modelling directional spatial processes in ecological data. *Ecological Modelling* 215: 325–336

See Also

[PEM-class PEM-functions](#)

Examples

```
## Exemplary graph:
data.frame(
  species = as.logical(c(0,0,1,0,0,0,0,0,0,0,1,1,1)),
  type = c(2,2,3,1,2,2,2,2,2,3,3,3),
  x = c(1,3,4,0,1.67,4,1,1.33,2.33,3.33,4.33,4,5),
```

```

y = c(1,1,1,0,0.5,0,-1,0,0,-0.5,-1,-0.5,-0.5),
row.names = sprintf("V%d",1:13)
) %>%
  st_as_sf(
    coords=c("x", "y"),
    crs = NA
  ) %>%
  graph %>%
  add.edge(
    from = c(1,2,1,5,4,4,5,9,4,8,9,4,7,7,6,6,9,10,10),
    to = c(2,3,5,2,1,5,9,2,8,9,6,7,8,9,3,13,10,12,11),
    data = data.frame(
      distance = c(4.2,4.7,3.9,3.0,3.6,2.7,4.4,3.4,3.6,3.3,
                  4.8,3.2,3.5,4.4,2.5,3.4,4.3,3.1,2.2),
      row.names = sprintf("E%d",1:19)
    )
  ) -> x

## Calculation of the influence matrix:
y <- InflMat(x)

## Obtain the number of edges:
nedge(y)

## Obtain the edge names:
edgenames(y)

## Obtain the edge data frame:
edge(y)

```

invDistWeighting	<i>Inverse Distance Weighting</i>
------------------	-----------------------------------

Description

A function to calculate inverse distance weights associated with a vector of weights.

Usage

```
invDistWeighting(x, a = 0, ...)
```

Arguments

x	A numeric vector containing a set of distances.
a	A numeric value for the distance exponent (default 0).
...	Any further arguments to be ignored by the function.

Details

Particular cases are when $a = 0$ (the default), whereby all the weights have equal values ($1/n$, where n is `length(x)`) and when any $d[i] == 0$, whereby the weights are non-zero only for the zero-valued distance(s). The weights are equal to $w[i] = d[i]^(-a)/S$, where S is the sum of $d[i]^(-a)$ for all the i .

Value

A numeric vector containing a set of weights summing to 1.

A numeric value.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

Examples

```
## Example 1, equal weights (the function's default when a = 0):
d0 <- c(10,1,0.1,0.2,2,20,0.3)
w0 <- invDistWeighting(d0)
w0
sum(w0)

## Example 2, inverse distance (a = 1):
w1 <- invDistWeighting(d0, a=1)
w1
sum(w1)

## Example 3, inverse squared distance (a = 2):
w2 <- invDistWeighting(d0, a=2)
round(w2, 5)
sum(w2)

## A case some of the distances are 0:
d1 <- c(10,0,0.1,0.2,0,20,0.3)
w3 <- invDistWeighting(d1, a=1) ## or any a != 0
w3
sum(w3)
```

Description

Utility functions to build linear models using Phylogenetic Eigenvector Maps among their explanatory variables.

Usage

```
model.data(
  formula,
  data,
  ...,
  na.action = na.pass,
  contrasts = NULL,
  discard.intercept = TRUE
)
```

```
Psquare(obs, prd)
```

Arguments

formula	an object of class " formula " (or one that can be coerced to that class): a symbolic description of the model data to be prepared. See ‘Details’ in lm for further information about model specification.
data	An optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
...	Additional parameters to be passed to the method.
na.action	A function (or the name of a function) for treating missing values (NAs) (default: <code>na.pass</code>).
contrasts	An optional list. See the <code>contrasts.arg</code> of model.matrix.default . (default: <code>NULL</code>).
discard.intercept	A logical ; whether or not to discard the intercept from the model matrix (default: <code>TRUE</code>).
obs	A numeric vector of observations.
prd	A numeric vector of model predictions.

Details

Function `model.data` is useful to prepare data to be given as response and auxiliary trait(s) to other functions such as [evolution.model.PEM](#). In general, the implicit constant term (intercept) is not useful and can be explicitly discarded.

Value

model.data A three-member list with member `$y` (a vector or matrix of response traits), member `$x` (a matrix auxiliary traits coded as numeric values), and member `$terms` (A model description).

Psquare A numeric value.

Functions

- `model.data()`: Model Data Preparation
Transforms data from various types into a list containing the response trait(s) and a strictly numeric auxiliary traits data matrix.
- `Psquare()`: Coefficient of Prediction
Calculates the prediction coefficient between observations and model predictions.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

molEvolSim

Molecular Evolution Simulator

Description

Functions to simulate the evolution of DNA sequences following a Markov substitution process.

Calculates the shift rate matrix associated with one of eight DNA evolution models.

Instantiate a DNA (molecular) evolution simulator from a shift intensity matrix over a given evolutionary time.

Generates a random sequence of gaps or DNA bases.

Generates a set of random nucleotide evolution rate from a gamma distribution.

Generates a set of DNA sequences along the edge of a phylogenetic network by evolving filial sequences from parental ones following a random Markov process.

Usage

```
DNArate(
  model = c("JC69", "K80", "K81", "F81", "HKY85", "T92", "TN93", "GTR"),
  piGap = 0,
  deletionRate = 0,
  insertionRate = 0,
  pi,
  piGC,
  par
)

molEvolSim(Q, step, rho)
```

```

drawDNASequence(NN, piGap = 0, pi = rep(0.25, 4L))

drawEvolRate(NN, gamma.shape = 5, gamma.scale = 5e-04)

simulateSequence(
  x,
  Q,
  sqn,
  rate,
  contrib = function(x, a = 0, ...) (x^-a)/sum(x^-a),
  ...
)

```

Arguments

model	A character string identifying one of eight DNA evolution models, namely "JC69": Jukes and Cantor (1969); "K80": Kimura (1980; aka K2P); "K81": Kimura (1981, aka K3P); "F81": Felsenstein (1981); "HKY85": Hasegawa, Kishino and Yano (1985); "T92": Tamura (1992); "TN93": Tamura and Nei (1993); "GTR": Tavaré (1986); or any unambiguous abbreviation thereof (default: "JC69").
piGap	Numeric; the equilibrium frequency of the gaps (a value between 0 and 1, default: 0).
deletionRate	Numeric; the rate of nucleotide deletion (default: 0).
insertionRate	Numeric; the rate of nucleotide insertion (default: 0).
pi	Optional. A length 4 numeric between 0 and 1, summing to 1 and giving the equilibrium base frequencies or the probabilities for drawing one of the four DNA bases (A, G, C, or T, in that order). When missing, the function assumes equal equilibrium base frequencies ($\text{pi} = \text{rep}(0.25, 4)$).
piGC	Optional. A numeric between 0 and 1; the GC relative frequency with respect to AT. When missing, the function assumes equal equilibrium base frequencies ($\text{piGC} = 0.5$).
par	Optional. A set with 1 to 6 evolution rate parameters (numeric; see the details below).
Q	A 5 x 5 shift intensity matrix (transition and transversion) such as the one obtained from function DNAevol (see details).
step	The simulation time step (arbitrary units of time).
rho	An evolution rate factor to be used on top of the shift intensity matrix.
NN	An integer; the number of locations to be generated. A locations is either one of the four nucleotides or a gap.
gamma.shape	A numeric; shape parameter of the beta distribution used to draw the nucleotide (and gaps) evolution rates.
gamma.scale	A numeric; scale parameter of the beta distribution used to draw the nucleotide (and gaps) evolution rates.
x	A graph-class object.

sqn	A raw vector or matrix containing one or more initial DNA sequence.
rate	A numeric vector of mean evolution rates.
contrib	A function that determine the contribution of the parent vertices as a function of the evolutionary distances (passed as its first argument. It may have any number of argument, each with a default value, and must accept arbitrary arguments (...). Default: <code>function(x, a = 0, ...) (x^a)/sum(x^a)</code> .
...	Any named arguments to be internally passed to other functions or methods.

Details

The molecular evolution model is based on a set of locations that can take one of five states, namely gap ('-'), adenine ('A'), guanine ('G') cytosine ('C'), or Thymine ('T'). A seed sequence is evolved one location at a time. Changes from one nucleotides to another appear as nucleotide transitions or transversions (shifts), whereas changes from a gap to one of the nucleotides appear as an insertion and changes from one the nucleotides to a gap as a deletion.

The changes are simulated as a simple Markov process, using a shift probability matrix, which is calculated using a shift intensity matrix *Q*. The off-diagonal elements of this matrix can be calculated from a DNA evolution model by `DNARate()`, The diagonal elements are defined by construct. Multiple shift intensity matrices can be employed for various simulated sequences, it necessary.

Gap-only locations are discarded at the outset of the process, yielding sets of sequences that are more or less shorter than the prescribed number of nucleotides depending on the gap frequency used in the initial sequence. The initial (root) sequence can be drawn from a uniform distribution with user-defined frequencies (i.e., using function `link{drawDNASequence}`), whereas each location's evolution rate can be drawn from a gamma distribution (i.e., using function `link{drawEvolRate}`).

The molecular evolution simulator can be instantiated multiple times, by calling `molEvolSim` multiple times and storing the results into a list. For instance, if every single location is assigned its own evolution rate, a simulator is implemented for every single location. Then, member function `$evolve(N)` is called to evolve a location 'N', with the returned value being the new location value. When the time step of evolution rate change, member function `$recalculate(step, rho)` is called to update the shift probability matrix with time step 'step' and evolution rate 'rho'. The shift matrix is obtained using member function `$getMt()`.

The gap opening and closure rates (arguments `gspOpen` and `gspClose`, respectively) correspond to the total of the four changes (from A, G, C, or T to a gap in the case of the gap opening rate and from a gap to A, G, C, or T, in the case of a gap closure).

Models "JC69", "K80", and "K81" do not require equilibrium base frequencies (argument `pi`), since they assume equal equilibrium base frequencies, whereas model "T92" uses parameter `piGC` to calculate these frequencies. Therefore, these models will disregard any values provided to argument `pi`. For the other model, unequal equilibrium base frequencies may be provided through argument `pi`. Otherwise, equal equilibrium base frequencies are assumed in all cases. Argument `par` is disregarded by models "JC69" and "F81". The number of values provided to that argument varies depending on the model. "K80", "HKY85", and "T92" require a single value, "K81" and "TN93", require two; and "GTR" requires six. In all cases, omitting `par` will yield a model equivalent to "F81" (or "JC69" with equal equilibrium base frequencies). Permissible values for argument `par` are positive values and are subjected to further constraints that depend on the model.

The "JC69" and "F81" models take no `par` value. The value of the single parameter (`par[1]`) of models "K80", "HKY85", and "T92", as well as the two parameters of model "TN93", have to be

between 0 and 1. For the "K81" model, the two parameters also must have a sum smaller than, or equal to 1. Finally, the six parameters of the "GTR" model must sum to 6, but can be individually larger than 1.

Function `simulateSequence()` is assigned a random sequence at the origin vertex (or vertices) of the evolutionary graph through its argument `sqn`, and also takes a set of evolution rate (one for each location) through its argument `rate`. The DNA sequence evolution is simulated as a Markov process described by the transition intensity matrix given as argument `Q`.

Value

`DNARate` A 5 x 5 shift intensity matrix (transition and transversion)

`molEvolSim` ...

`drawDNASequence` A raw vector of length `NN` containing the ASCII values for characters '-' (0x2d), 'A' (0x41), 'C' (0x43), 'G' (0x47), and 'T' (0x54) representing random nucleotides to be used as the seed sequence for a DNA fragment.

`drawEvolRate` A numeric vectors of length `NN` containing the evolution rate for each of the nucleotides in the sequence.

`simulateSequence` A raw matrix with as many rows as the number of vertices in the graph and as many columns as the number of nucleotides in the simulated sequence. The elements of the matrix are ASCII values for characters '-' (0x2d), 'A' (0x41), 'C' (0x43), 'G' (0x47), and 'T' (0x54) representing the simulated nucleotides.

Functions

- `DNARate()`:
- `molEvolSim()`:
- `drawDNASequence()`:
- `drawEvolRate()`:
- `simulateSequence()`:

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

References

- Jukes, T.H. & Cantor, C.R. (1969). Evolution of Protein Molecules. New York: Academic Press pp. 21-132. doi:10.3389/fgene.2015.00319
- Kimura, M. 1980. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution* 16(2): 111-120. doi:10.1007/BF01731581
- Kimura, M. 1981. Estimation of evolutionary distances between homologous nucleotide sequences. *Proceedings of the National Academy of Sciences of the United States of America* 78(1): 454-458. doi:10.1073/pnas.78.1.454

- Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution* 17(6): 368-376. doi:10.1007/BF01734359
- Hasegawa, M.; Kishino, H. & Yano, T. 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution* 22(2): 160-174. doi:10.1007/BF02101694
- Tamura, K. 1992. Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+C-content biases. *Molecular Biology and Evolution*. 9(4): 678-687. doi:10.1093/oxfordjournals.molbev.a040752
- Tamura, K. & Nei, M. 1993. Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution* 10(3): 512-526. doi:10.1093/oxfordjournals.molbev.a040023
- Tavaré S. 1986. Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. *Lectures on Mathematics in the Life Sciences*. 17: 57-86.

Examples

```
## Examples of various molecular evolution models

## Jukes and Cantor (1969):
DNARate("JC69", piGap=0.30, insertionRate=0.02, deletionRate=0.02)

## Kimura 1980:
DNARate("K80", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("K80", piGap=0.30, insertionRate=0.02, deletionRate=0.02, par=0)
DNARate("K80", piGap=0.30, insertionRate=0.02, deletionRate=0.02, par=2/3)
DNARate("K80", piGap=0.30, insertionRate=0.02, deletionRate=0.02, par=1)

## Kimura 1981:
DNARate("K81", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("K81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(0,1/3))
DNARate("K81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(1/3,0))
DNARate("K81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(1/3,0))
DNARate("K81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(0.1,0.4))

## Felsenstein 1981:
DNARate("F81", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("F81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.15,0.10,0.25))
DNARate("F81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.15,0.5,0.12,0.23))
DNARate("F81", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.15,0.5,0.05,0.3))

## Hasegawa, Kishino, and Yano (1985)
DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.15,0.10,0.25))
```

```

DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.15,0.10,0.25), par=0.1)
DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.15,0.10,0.25), par=0.5)
DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.15,0.10,0.25), par=0.9)
DNARate("HKY85", piGap=0.30, insertionRate=0.02, deletionRate=0.02, par=0.9)

## Hasegawa, Kishino, and Yano (1985)
DNARate("HKY85")
DNARate("HKY85", pi = c(0.5, 0.15, 0.10, 0.25))
DNARate("HKY85", par = 0.1, pi = c(0.5, 0.15, 0.10, 0.25))
DNARate("HKY85", par = 0.5, pi = c(0.5, 0.15, 0.10, 0.25))
DNARate("HKY85", par = 0.9, pi = c(0.5, 0.15, 0.10, 0.25))
DNARate("HKY85", par = 0.5)

## Tamura (1992)
DNARate("T92", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("T92", piGap=0.30, insertionRate=0.02, deletionRate=0.02, par=0.1)
DNARate("T92", piGap=0.30, insertionRate=0.02, deletionRate=0.02, piGC=0.25)
DNARate("T92", piGap=0.30, insertionRate=0.02, deletionRate=0.02, piGC=1/3,
        par=0.2)

## Tamura & Nei (1993)
DNARate("TN93", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("TN93", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(0.1,0.2))
DNARate("TN93", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par = c(1/3,1/2))

## Generalized time-reversible (GTR; Tavaré 1986)
DNARate("GTR", piGap=0.30, insertionRate=0.02, deletionRate=0.02)
DNARate("GTR", piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        par=c(1.5,1,0.5,0,2,1))
DNARate("GTR", , piGap=0.30, insertionRate=0.02, deletionRate=0.02,
        pi=c(0.5,0.25,0.15,0.10), par=c(1.5,1,0.5,0,2,1))

## The transition intensity matrix from a Kimura (1980) model:
Q <- DNARate(model="K80", piGap = 0.3, deletionRate=0.1, insertionRate=0.1)
Q

## Implement the molecular evolution simulator for a single nucleotide:
molEvolSim(Q = Q, step = 1, rho = 5) -> em1

## Get the transition probability matrix as follows:
em1$getMt()

## A vector of raw as examples of initial traits:
tr <- charToRaw("-AGCT")

## Set the RNG seed.
set.seed(28746549L)

```

```

## Simulate molecular evolution from:
rawToChar(em1$evolve(tr[1L])) ## a gap.
rawToChar(em1$evolve(tr[2L])) ## an adenine base.
rawToChar(em1$evolve(tr[3L])) ## a guanine base.
rawToChar(em1$evolve(tr[4L])) ## a cytosine base.
rawToChar(em1$evolve(tr[5L])) ## a thymine base.

## Recalculate the probabilities for a lower mean evolution rate (one tenth
## the previous one):
em1$recalculate(1, 0.1)

em1$getMt() ## The recalculated transition probability matrix.

## Simulate molecular evolution from:
rawToChar(em1$evolve(tr[1L])) ## a gap.
rawToChar(em1$evolve(tr[2L])) ## an adenine base.
rawToChar(em1$evolve(tr[3L])) ## a guanine base.
rawToChar(em1$evolve(tr[4L])) ## a cytosine base.
rawToChar(em1$evolve(tr[5L])) ## a thymine base.

## Base changes are now less probable.

## Simulate the evolution of a sequence with 100 base pairs for 250 generations.
Ngeneration <- 250
Nnucleotide <- 100

## This is the matrix holding the sequences:
seq <- matrix(raw(), Ngeneration, Nnucleotide)

## Drawing the initial sequence:
seq[1,] <- drawDNASequence(Nnucleotide, piGap = 0.25, pi = c(0.4,0.4,0.1,0.1))

## Each site has its own mean evolution rate, which are drawn as follows:
erate <- drawEvolRate(Nnucleotide, gamma.shape = 5, gamma.scale = 5e-03)

## Using the Hasegawa, Kishino, and Yano (1985) model:
DNARate(
  model = "HKY85",
  piGap = 0.25,
  deletionRate = 0.1,
  insertionRate = 0.1,
  pi = c(0.4, 0.4, 0.1, 0.1),
  par = 0.25
) -> Q

## Instantiating a molecular evolution models for each site using the single
## change rate matrix, a constant time step of 1 (Ma), and individual mean
## evolution rates drawn previously:
em <- list()
for(j in 1:Nnucleotide)
  em[[j]] <- molEvolSim(Q = Q, step = 1, rho = erate[j])

## These for loops call the $evolve() function to evolve each site for each

```

```

## generation as follows:
for(i in 2:Ngeneration)
  for(j in 1:Nnucleotide)
    seq[i, j] <- em[[j]]$evolve(seq[i - 1, j])

## Sequences with the gaps (perfect alignment):
concatenate(seq) %>%
  show.sequence

## Sequences with the gaps removed (prior to multiple sequence alignment):
concatenate(seq, discard="-") %>%
  show.sequence

### Examples for molEvolSim() here...

## Clean up:
rm(Q, em1, tr, Ngeneration, Nnucleotide, seq, erate, em, i, j)

```

MPSEM-generics

Generic Function (Methods)

Description

A set of non-standard generic functions used by package MPSEM.

Usage

```

nedge(x)

edge(x)

edge(x) <- value

edgenames(x)

edgenames(x) <- value

geometry(x)

geometry(x) <- value

labels(object) <- value

as.graph(x, ...)

locate(x, target, ...)

```

```

evolution.model(object, y, ..., x = NULL)

## S3 method for class 'graph'
edge(x)

## S3 replacement method for class 'graph'
edge(x) <- value

## S3 method for class 'graph'
edgenames(x)

## S3 replacement method for class 'graph'
edgenames(x) <- value

```

Arguments

x	A graph-class object or an object of any class that can be converted into a graph-class object, or a numeric vector or numeric matrix of auxiliary trait(s) to help in estimating the parameters of the trait evolution model. In the second case, the default is NULL, which entails that no auxiliary trait is used.
value	A two-column data.frame giving the coordinates (e.g., x, y, z) to be applied to the graph-class object.
object	A graph-class object or an object of any class that can be converted into a graph-class object.
...	Further arguments of the particular conversion method that is being implemented.
target	A numeric or character vector specifying one or more target species in graph x.
y	A numeric vector or numeric matrix of trait(s) against which to estimate the parameters of the trait evolution model.

Value

The return value depends on the implementation of the methods for a particular S3 class.

Functions

- `nedge()`: Number of Edges
A method to access the number of edges
- `edge()`: Edge Extraction
A method to extract edges from an object.
- `edge(x) <- value`: Edge Assignment
A method to assign edges to an object.
- `edgenames()`: Edge Names Extraction
A method to extract edge names from an object.
- `edgenames(x) <- value`: Edge Assignment
A method to assign edges to an object.

- `geometry()`: Geometry Extraction
A method to obtain an object's geometry.
- `geometry(x) <- value`: Geometry Assignment
A method to assign a geometry to an object.
- `labels(object) <- value`: Labels Assignment
A method to assign labels to an object.
- `as.graph()`: Graph-class Conversion
A method to convert an object into a graph-class object.
- `locate()`: Locate Targets
A method to locate a set of targets species from a graph-class object.
- `evolution.model()`: Estimate Evolution Model
Estimate the parameters of an evolution model on a PEM2-class object.
- `edge(graph)`: Edge Extraction
Extracts the edges of a graph-class object.
- `edge(graph) <- value`: Edge Assignment
Assigns edges to a graph-class object.
- `edgenames(graph)`: Edge Names Extraction
Extracts the edge names of a graph-class object.
- `edgenames(graph) <- value`: Edge Names Assignment
Assigns edge names to a graph-class object.

PEM-class

Class and Methods for Phylogenetic Eigenvector Maps (PEM)

Description

Class and methods to calculate and manipulate Phylogenetic Eigenvector Maps (PEM).

Usage

```

PEM(
  x,
  ...,
  a = -Inf,
  mm_a = matrix(1, nedge(x), 1L),
  psi = NULL,
  mm_psi = matrix(NA, nedge(x), 0L),
  d = "distance",
  sp = "species",
  tol = .Machine$double.eps^0.5
)

## S3 method for class 'PEM'
```

```

print(x, ...)

## S3 method for class 'PEM'
as.matrix(x, ...)

## S3 method for class 'PEM'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'PEM'
update(object, a, psi, ...)

## S3 method for class 'PEM'
evolution.model(object, y, ..., x = NULL)

## S3 method for class 'PEM'
predict(object, newdata, ...)

```

Arguments

x	A graph-class object, a PEM-class object, or a matrix containing auxiliary traits.
...	Additional parameters to be passed to the method. Currently ignored.
a	A numeric vector of linear coefficients to estimate the steepness in the trait evolution model; it must have as many values as the number of columns in argument mm_a (default: -Inf).
mm_a	A numeric matrix with as many rows as the number of edges and as many columns as the number of values of argument a (default: an all-ones single-column matrix).
psi	A numeric vector of linear coefficients to estimate the evolution rate in the trait evolution model; it must have as many values as the number of columns in argument mm_psi (default: NULL, meaning an homogeneous evolution rate throughout the graph).
mm_psi	A numeric matrix with as many rows as the number of edges and as many columns as the number of values of argument psi (default: a zero-column matrix).
d	A character string; the name of the (numeric) edge property containing the phylogenetic distances (default: "distance").
sp	A character string; the name of the (logical) vertex property containing vertex property (default: "species").
tol	A numeric; the lowest singular value above which to retain a singular vector as part of a PEM (default: .Machine\$double.eps^0.5).
row.names	Included for method consistency reason; ignored.
optional	Included for method consistency reason; ignored.
object	A PEM-class object.
y	A vector or matrix of (numeric) trait values to estimate the parameters of the trait evolution model.

newdata A three-column data frame giving the locations of a set of targets in the graph such as the one provided by function `locate.graph` (element `$location`; see Details below).

Format

A `PEM-class` object contains:

d the name of the edge property containing the phylogenetic distances (see argument `d` above),
sp the name of the vertex property identifying which of the vertices are bearing trait values (see argument `sp` above),
tol the tolerance value (see argument `tol` above),
mm a list with the model matrices (see arguments `mm_a` and `mm_psi` above) that are used to estimate the steepness and relative evolution rate throughout the graph manifold,
nsp the number of vertices that bear trait values,
B the influence matrix (see `inflMat`),
print function; prints the PEM object,
graph function; returns the graph object used to calculate the PEM,
pem function; returns a list of components (ie., vectors and matrices) involved in PEM calculation,
par function; returns the actual model parameter values as a list,
nodal function; returns the steepness and evolution rates at any given vertex of the graph,
update function; updates the PEM with a new set of parameters,
var function; returns the PVCV matrix: phylogenetic variance and covariances matrix among the vertices with trait values,
inv_var function; returns the inverse PVCV matrix,
logdet function; returns the natural logarithm of the PVCV matrix determinant,
dev function; calculates the deviance of a target trait, or many such traits, with respect to the PVCV matrix, optionally conditional to one or more auxiliary trait(s),
S2 function; calculates the variance(s) of one or more trait(s) with respect to the PVCV matrix or the residual variance(s) of said trait(s) conditioned to one or more auxiliary trait(s),
predictVertex function; calculates the PEM prediction scores at the graph's vertices, and
predictEdge function; calculates the PEM prediction scores at the graph's edge.

Details

The class is implemented using function `PEM`. It provides methods `as.matrix` and `as.data.frame` to extract the phylogenetic vectors in the forms of a matrix or a data frame, respectively, an update method to recalculate an existing PEM with new parameters (ie., arguments `a` and `psi`), an `evolution.model` method to empirically estimate parameter values from a data set of traits (ie., argument `y` and `x`), and a `predict` method to estimate PEM scores for arbitrary graph locations (using argument `newdata`).

Graph locations are given as a three-column data frame:

ref the index of the vertex or edge where each target can be found,

dist the phylogenetic distance along the edge where each target can be found, or NA for a target located at a vertex,

lca the phylogenetic distance between the latest common ancestor of the target in the graph and the target itself.

One such data frame is one of the second element of the list produced by method [locate.graph](#) (called "location").

Functions

- `PEM()`: Create PEM-class
A function creating a PEM-class object.
- `print(PEM)`: Print PEM-class
A print method for PEM-class objects.
- `as.matrix(PEM)`: Method `as.matrix` for PEM-class Objects
A method to extract the phylogenetic eigenvectors from a PEM-class object.
- `as.data.frame(PEM)`: Method `as.data.frame` for PEM-class Objects
A method to extract the phylogenetic eigenvectors from a PEM-class object.
- `update(PEM)`: Update Method for PEM-class Objects
An updating method to recalculate a PEM with new smoothness and evolution rate parameters.
- `evolution.model(PEM)`: Evolution Model Method for PEM-class Objects
A method to estimate the smoothness and evolution rate parameters of a PEM-class object empirically on the basis of an observed quantitative trait and, optionally, optional trait(s).
- `predict(PEM)`: Predict Method for PEM-class Objects
A predict method to obtain PEM prediction scores for arbitrary graph locations.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

See Also

[PEM-functions](#)

Examples

```
## Synthetic example

## This example describes the phyogeny of 7 species (A to G) in a tree with 6
## nodes, presented in Newick format, read by function
## read.tree of package ape.

t1 <- read.tree(text=paste(
  "(((A:0.15,B:0.2)N4:0.15,C:0.35)N2:0.25,((D:0.25,E:0.1)N5:0.3,",
  "(F:0.15,G:0.2)N6:0.3)N3:0.1)N1:0.1;", sep=""))
t1
summary(t1)

## Example of a made up set of trait values for the 7 species
y <- c(A=-1.1436265,B=-0.3186166,C=1.9364105,D=1.7164079,E=1.0013993,
      F=-1.8586351,G=-2.0236371)

## The phylogenetic tree is turned into an evolutionary graph as follows:
x <- as.graph(t1)

## The graph:
x

## This is the edge information of that simple graph:
edge(x)

## Calculate the (binary) influence matrix; E1 to E12 are the tree edges:
IM1 <- InflMat(x)
IM1

## This is the edges associated with the influence matrix:
edge(IM1)

## This is the influence matrix for the tree leaves (ie., the graph's
## terminal vertices):
IM1[x$species,]

## Here is a suite of weighting function profiles:
seq(0,1.5,0.001) %>%
  plot(y=PEMweights(., a=0), ylim=c(0,1.7), type="l", xlab="distance",
       ylab="weight")

seq(0,1.5,0.001) %>%
  lines(y=PEMweights(., a=0.5), col="red")

seq(0,1.5,0.001) %>%
  lines(y=PEMweights(., a=0.5, psi=1.5), col="green")

seq(0,1.5,0.001) %>%
  lines(y=PEMweights(., a=0.9), col="blue")

## A Phylogenetic eigenvector maps (PEM) is build with the default parameters
```

```

## as follows:
PEM1 <- PEM(x)

## The PEM object:
PEM1

## The as.matrix methods returns the phylogenetic eigenvectors as a matrix:
as.matrix(PEM1)

## The as.data.frame methods returns the phylogenetic eigenvectors as a data
## frame:
as.data.frame(PEM1)

## The update method recalculates the PEM with new parameters:
update(PEM1, a=log(0.25/(1 - 0.25)), psi=NULL)

## Note: The way 'a' is estimated is based on an inverse-logit-transformed
## linear sub-model. This approach is used to facilitate the bounding of the
## steepness parameter in the [0,1] interval. Therefore, to set a given
## steepness value, one has to provide its logit-transformed value. As a
## reminder, the logit transformation is  $f(x) = \log(x/(1 - x))$ .

## To show the edges of the graph within the PEM object:
edge(PEM1$graph())

## A second PEM is initialized with a = 0.2 as follows:
PEM2 <- PEM(x, a=log(0.2/(1 - 0.2)))
PEM2

## The edges of PEM2 graph:
edge(PEM2$graph())

## A third PEM is initialized with a = 1 as follows:
PEM3 <- PEM(x, a=Inf)
PEM3

## The edges of PEM3 graph:
edge(PEM3$graph())

## Estimation on the steepness parameter empirically using data set y
## Initial value must be finite (not -Inf nor Inf, meaning a > 0 and a < 1):
PEM4 <- PEM(x, a=0)

## Method evolution.model carry out the estimation:
opt <- evolution.model(PEM4, y=y)
opt

## The edges of PEM4 graph after the estimation:
edge(PEM4$graph())

## Graph locations for target species X, Y, and Z not found in the original
## data set:
read.tree(

```

```

text = paste(
  "((X:0.45,((A:0.15,B:0.2)N4:0.15,(C:0.25,Z:0.2)NZ:0.1)N2:0.05)NX:0.2,",
  "(((D:0.25,E:0.1)N5:0.05,Y:0.25)NY:0.25,(F:0.15,G:0.2)N6:0.3)N3:0.1)N1;",
  sep=""
)
) -> tree
tree

## The tree is converted into a graph as follows:
x2 <- as.graph(tree)
x2

## The target X, Y, and Z are extracted from graph x2 as follows:
loc <- locate(x2, target = c("X","Y","Z"))
loc

## The list contains the residual graph and a data frame with the locations
## of the target in the residual graph.

## Building a PEM on the residual graph (assuming a = 0):
PEM5 <- PEM(loc$x)

## For making predictions, we need to do any of the following:
## 1. run evolution.model(); doing this operation will change the parameter
##    values,
## 2. estimate the deviance using $dev(y, ...)
## 3. estimate the variance using $S2(y, ...)

## Before any of these operations is done, no variance estimate is available:
PEM5$S2()

## Presenting a variable to the PEM makes its variance estimate available
## later on. For instance, presenting 'y' to the PEM5 while estimating its
## deviance as follows:
PEM5$dev(y)

## This operation makes the variance estimate for 'y' available for later
## access:
PEM5$S2()

## The predict method is used to generate the predictors as follows:
scr <- predict(PEM5, loc$location)
scr

## The 'vf' attribute, which can be accessed as follows:
attr(scr,"vf")
## is the amount of variance associated with the evolutionary distance
## between each target species and its latest common ancestor in the
## evolutionary graph.

## Given a model built using the PEM as follows:
lm1 <- lm(y ~ U_2 + U_3 + U_5, data=as.data.frame(PEM5))

```

```

## Predictions are obtained as follows:
ypred <- predict(lm1, as.data.frame(scr))
ypred

## Estimate the ancestral trait values

## The ancestors of the species of the tree are the vertices without extant
## species:
data.frame(
  ref = which(!x$species),
  dist = rep(NA_real_, sum(!x$species)),
  lca = rep(0, sum(!x$species)),
  row.names = rownames(x)[!x$species]
) -> anc

## The scores of the ancestors are obtained as follows:
src_anc <- predict(PEM1, anc)
## Note the absence of the 'vf' attributes since no variable has been
## presented to PEM1.

## Predictions are obtained from the previous linear model as follows:
predict(lm1, as.data.frame(src_anc))

```

PEM-functions

Phylogenetic Eigenvector Maps

Description

Functions to calculate and manipulate Phylogenetic Eigenvector Maps (PEM), which are sets of eigenfunctions describing the structure of a phylogenetic graph. Each computation function is briefly described in section Functions below.

Usage

```
PEMweights(d, a = 0, psi = 1)
```

Arguments

d	A numeric vector of the evolutionary distances (PEMweights) or a character string specifying the row name of the edge table (located in the graph-class object's attribute edge) where the evolutionary distances (edge lengths) can be found.
a	The steepness parameter describing whether changes occur, on average: progressively long edges (a close to 0) or abruptly at vertices (a close to 1; default: 0).
psi	Relative evolution rate along the edges. This parameter only becomes relevant when multiple values are assigned to different portions of the phylogeny (default: 1).

Value

The returned value depends on the function:

PEMweights A set of numeric value to be used as weights during PEM calculation.

Functions

- **PEMweights()**: PEM Weighting
A power function to obtain the edge weights used during PEM calculation.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) – Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

- Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution*. 4: 1120–1131
- Makarek, V., Legendre, L. & Desclève, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89–96
- Blanchet, F. G., Legendre, P. & Borcard, D. 2008. Modelling directional spatial processes in ecological data. *Ecological Modelling* 215: 325–336

See Also

[PEM-class](#)

PEM-utils

Utility Function for Phylogenetic Eigenvector Maps

Description

A set of functions to help in calculating Phylogenetic Eigenvector Maps (PEM).

Usage

```
isUnderEdge(x, e)

isUnderVertex(x, v)

aggregateOnVertex(
  x,
  data,
  fun,
  ...,
```

```

    v = 1L:NROW(x),
    default = rep(0, NCOL(data))
  )

```

Arguments

x	A graph-class object.
e	A numeric vector of indices or a character vector of names representing one or more edges.
v	A numeric vector of indices or a character vector of names representing one or more vertices. The default for function <code>aggregateOnVertex</code> is all the vertices of x.
data	A matrix or data.frame containing edge data to be aggregated on the graph's vertices.
fun	A function to be applied on the incoming edges of the vertices.
...	Supplementary arguments to be internally passed to fun.
default	A vector of values to be assigned to vertices that have no incoming edges. The default is a vector of zeros with the same length as the number of columns in data.

Details

Functions `isUnderEdge` and `isUnderVertex` are used to set individual graph locations, in terms of edges or vertices where shifts in the values of selection (a) and/or evolution rate (psi) parameters are occurring. The binary contrasts matrices generated by function `isUnderEdge` and `isUnderVertex` are used as descriptors in models representing these subordinate parameters using higher order ones.

Function `aggregateOnVertex` allows one to aggregate variables associated with the edges on their destination vertices using a user-defined function.

Value

Functions `isUnderEdge` and `isUnderVertex` return a matrix of binary contrasts whose rows represent the edges of the graph and columns the edges or vertices given as indices or names to function `isUnderEdge` (argument `e`) or `isUnderVertex` (argument `v`), respectively. Function `aggregateOnVertex` returns a matrix whose rows represent the vertices specified by its argument `v` and columns represent the variables given using its argument `data`.

Functions

- `isUnderEdge()`: Edges Under or At an Edge
Identify which of the edges of a graph are located at or under one or more of the graph's edges using a binary contrast matrix.
- `isUnderVertex()`: Edges Under a Vertex
Identify which of the edges of a graph are located under one or more of the graph's vertices using a binary contrast matrix.
- `aggregateOnVertex()`: Aggregate on Vertex
Aggregate incoming edge values on a vertex or a set of vertices.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) – Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution*. 4: 1120–1131

Makarenkov, V., Legendre, L. & Desdevisse, Y. 2004. Modelling phylogenetic relationships using reticulated networks. *Zoologica Scripta* 33: 89–96

Blanchet, F. G., Legendre, P. & Borcard, D. 2008. Modelling directional spatial processes in ecological data. *Ecological Modelling* 215: 325–336

See Also

[PEM-functions](#), [PEM-class](#)

Examples

```
## Synthetic example
data.frame(
  species = as.logical(c(0,0,1,0,0,0,0,0,0,0,1,1,1)),
  type = c(2,2,3,1,2,2,2,2,2,3,3,3),
  x = c(1,3,4,0,1.67,4,1,1.33,2.33,3.33,4.33,4,5),
  y = c(1,1,1,0,0.5,0,-1,0,0,-0.5,-1,-0.5,-0.5),
  row.names = sprintf("V%d",1:13)
) %>%
st_as_sf(
  coords=c("x","y"),
  crs = NA
) %>%
graph %>%
add.edge(
  from = c(1,2,1,5,4,4,5,9,4,8,9,4,7,7,6,6,9,10,10),
  to = c(2,3,5,2,1,5,9,2,8,9,6,7,8,9,3,13,10,12,11),
  data = data.frame(
    distance = c(4.2,4.7,3.9,3.0,3.6,2.7,4.4,3.4,3.6,3.3,
                 4.8,3.2,3.5,4.4,2.5,3.4,4.3,3.1,2.2),
    row.names = sprintf("E%d",1:19)
  )
) -> gr_ex

## Plot the graph:
plot(gr_ex, cex.min=3, cex.lab=0.6)

## Show the edges of the graph:
edge(gr_ex)

## Identify the edges that are under or at the edges E7 or E17 using a binary
```

```
## contrast matrix:
isUnderEdge(gr_ex, c("E7","E17"))

## Identify the edges that are under vertices V5 or V9 using a binary
## contrast matrix:
tmp <- isUnderVertex(gr_ex, c("V5","V9"))
tmp

## Aggregate the result of isUnderVertex() using the following function
## enables one to determine which of the vertices are found under V5 and V9:
aggregateOnVertex(gr_ex, tmp, function(x) ifelse(any(as.logical(x)),1,0))
```

pemlm-class

PEM Linear Model Interface

Description

An interface to estimate a linear model

Usage

```
pemlm(formula, data, pem, ..., contrasts = NULL)

## S3 method for class 'pemlm'
print(x, ...)

## S3 method for class 'pemlm'
summary(object, ...)

## S3 method for class 'pemlm'
anova(object, ...)

## S3 method for class 'pemlm'
predict(
  object,
  newdata = data.frame(row.names = rownames(newloc)),
  newloc,
  se.fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

<code>formula</code>	An object of class " formula " (or one that can be coerced to that class): a symbolic description of the model to be fitted. (see 'details' in lm for further details on model specification.
<code>data</code>	An optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>pemlm</code> is called.
<code>pem</code>	A PEM-class object.
<code>...</code>	Additional arguments to be passed to the low level regression fitting functions.
<code>contrasts</code>	An optional list. See the <code>contrasts.arg</code> of model.matrix.default .
<code>x, object</code>	A <code>pemlm-class</code> object.
<code>newdata</code>	An optional data frame containing auxiliary traits used for making predictions.
<code>newloc</code>	A data frame containing the graph locations of species targeted for making predictions.
<code>se.fit</code>	A logical specifying whether to return standard errors (default: FALSE).
<code>interval</code>	One of "none", "confidence", or "prediction") specifying the type of interval associated with the predictions.
<code>level</code>	Tolerance/confidence level (default: 0.95).

Format

A `pemlm-class` object contains:

auxModel A [lm-class](#) object.

resetPEM A function ...

getPEM A function ...

aic A function ...

getIncluded A function ...

getCandidate A function ...

scanCandidate A function ...

forward A function ...

promote A function ...

pemModel A function ...

Details

...

Value

pemlm A pemlm-class object.
print.pemlm NULL (invisibly).
summary.pemlm A summary.lm-class object.
anova.pemlm An anova-class data frame.
predict.pemlm ...

Functions

- **pemlm()**: PEM Linear Model
Calculate a PEM-based linear model for estimating trait values.
- **print(pemlm)**: Print PEM Linear Model
A print method for pemlm-class objects.
- **summary(pemlm)**: Summarize pemlm Model
A summary method for pemlm-class objects.
- **anova(pemlm)**: Anova of a pemlm Model
An anova method for pemlm-class objects.
- **predict(pemlm)**: Predict Trait Values
A predict method for pemlm-class objects.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G., Legendre, P., and Peres-Neto, P. 2013. Phylogenetic eigenvector maps: a framework to model and predict species traits. *Methods in Ecology and Evolution* 4: 1120-1131

Examples

```
## Example here...
```

randomGraph

*Random Graph Generator***Description**

A function for generating a random directed graph.

Usage

```
randomGraph(
  NV,
  NC,
  NP,
  timestep,
  maxDist,
  ...,
  mean = c("arithmetic", "geometric", "harmonic"),
  weighting = invDistWeighting,
  verbose = TRUE,
  saveDist = TRUE
)
```

Arguments

NV	An integer. The number of vertices to generate.
NC	A function with a ... argument returning the maximum number outgoing edges from each of the vertices; effectively determining the maximum number of children vertices for any given vertex.
NP	A function with a ... argument returning the maximum number of incoming edges to each of the vertices; effectively determining the maximum number of parent vertices for any given vertex.
timestep	A function with a ... argument returning the amount of time associated with the edges.
maxDist	A function with a ... argument returning the maximum distances, in terms of time, allowed between any two parents vertices.
...	Any arguments to be passed internally to the functions given as arguments NC, NP, timestep, maxDist, or weighting.
mean	One of character strings "arithmetic", "geometric", "harmonic", or any unambiguous abbreviation thereof, specifying the type of mean used for averaging the distances when a vertex has multiple ascendant edges.
weighting	A weighting function; it takes a set of distances as its first argument and returns a set of weights summing to 1 (default: invDistWeighting).
verbose	A Boolean. Whether or not to print messages associated with the graph simulation process (default: TRUE).
saveDist	A Boolean. Whether or not to save the graph distance matrix as an attribute to the returned graph-class object (default: TRUE).

Details

Details contents...

Value

A [graph-class](#) object.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>) Maintainer: Guillaume Guénard <guillaume.guenard@umontreal.ca>

See Also

[graph-class](#).

Examples

```
## Setting the RNG seed to obtain consistent examples:
set.seed(2182955)

## A linear evolutionary sequence with random edge lengths between 2 and 5:
randomGraph(
  NV = 100,
  NC = function(...) 1,
  NP = function(...) 1,
  timestep = function(ts_min, ts_max, ...) runif(1, ts_min, ts_max),
  maxDist = function(...) NULL,
  ts_min = 2,
  ts_max = 5
)

## As above, but allowing for dichotomic splitting.
randomGraph(
  NV = 100,
  NC = function(...) 2,
  NP = function(...) 1,
  timestep = function(ts_min, ts_max, ...) runif(1, ts_min, ts_max),
  maxDist = function(...) NULL,
  ts_min = 2,
  ts_max = 5
)

## A random evolutionary graph with random numbers of children and parents per
## node, random time steps, and a random maximum distance between the parents:
randomGraph(
  NV = 250,
  NC = function(lambda_child, ...) 1 + rpois(1, lambda_child),
  NP = function(lambda_parent, ...) 1 + rpois(1, lambda_parent),
  timestep = function(ts_min, ts_max, ...) runif(1, ts_min, ts_max),
  maxDist = function(max_anc, ...) runif(1, 0, max_anc),
```

```

lambda_child = 2.5,
lambda_parent = 4,
ts_min = 2,
ts_max = 5,
max_anc = 4
)

```

show.sequence

Show DNA Sequences

Description

A function displaying a set of DNA sequences in the form of a mosaic of colored tiles.

Usage

```

show.sequence(
  x,
  xlim,
  ylim,
  xlab = "Position",
  text = FALSE,
  col = c(A = "red", G = "yellow", C = "green", T = "blue", ` ` = "grey"),
  ...
)

```

Arguments

<code>x</code>	A vector containing the DNA sequences (ATCG-; character).
<code>xlim</code>	The range of nucleotides to show (a length 2 numeric vector).
<code>ylim</code>	The range of sequences to show (a length 2 numeric vector).
<code>xlab</code>	An optional label for the x-axis (character).
<code>text</code>	Whether to show the letters associated with the DNA bases (default: FALSE; logical).
<code>col</code>	A named vector of color values with which to show the nucleotides.
<code>...</code>	Further arguments (graphical parameters) to be passed internally to function plot and text .

Value

NULL (invisibly).

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

Examples

```
## A set of exemplary DNA sequences:
sqn <- c(Sequence_0 = "ATCG-TTT-G--C--CA--TTA--TTAA-GTGC-GTGGTCT-TCA",
        Sequence_1 = "ATCG-TTTCG--CCCCA--TTA--TTAA-GTAA-GTAATCTTTCA",
        Sequence_2 = "TTGGCTTCC--TC--CAT-TTC--TTCA-GT-ACG-ATTCTTTTA",
        Sequence_3 = "TTCG-TACC-T-T---A-ATAA--T-AA-GTCTTGTAATCGTTCA",
        Sequence_4 = "TTCG-TACC-T-T-T-C-ATAA--T-AA-GTCTGGTGGTCGTTCA",
        Sequence_5 = "TTCG-TACG-T-T-T-C-ATAT--T-AA-GTCTGGTGGTCGTTCA",
        Sequence_6 = "TTCG-TACG-T-T-T-GATTTT--T-AA-GTCTGGT---CGTTCA",
        Sequence_7 = "TTAG-TACG-T-T-T-GATTTT--T-TT-G---GGT---CGTTTT",
        Sequence_8 = "TTAG-TACG-T-T-T-GATTTT--T-TT-G---GA----CGTTTT",
        Sequence_9 = "TTAG-TACG-TATCT-GAT--TAAT-TT-G---GA----CG--TA")

## With all the defaults:
show.sequence(sqn)

## Displays the nucleotides and the gaps:
show.sequence(sqn, text=TRUE, cex=0.75)

## Clean up:
rm(sqn)
```

traitEvolSim

Trait Evolution Simulator

Description

Functions to simulate the evolution of traits as an Ornstein-Uhlenbeck process, optionally with trait optimal value evolving following a Markov process.

Usage

```
traitEvolSim(
  name,
  sigma = 1,
  step = 1,
  alpha = 0,
  optima = NULL,
  transition = NULL
)

## S3 method for class 'QTEM'
print(x, ...)

simulateTrait(x, tem, state, value, weighting = invDistWeighting, ...)
```

Arguments

name	Name of the trait whose evolution is to be created.
sigma	Variance parameter of the trait (default: sigma = 1).
step	Simulation step size (default: step = 1).
alpha	Trait selection rate (default: alpha = 0, meaning purely neutral trait evolution)
optima	Trait optimal value(s) (default: optima = NULL).
transition	A square, n-by-n, transition intensity matrix (default: transition = NULL, which is suitable for single trait optima).
x	A QTEM-class or graph-class object.
...	further arguments to be passed to other functions and methods
tem	A QTEM-class object from function traitEvolSim or a list of such objects (see details).
state	One or more initial trait state(s) (default: 1).
value	One or more initial trait value(s).
weighting	A weighting function that determine the contribution of the parent vertices as a function of the evolutionary distances (passed as its first argument. It may have any number of argument, each with a default value, and must accept arbitrary arguments (...). Default: invDistWeighting .

Details

The trait evolution simulator is based on a random walk process of one of categories: Wiener process and the Ornstein-Uhlenbeck process. The Wiener process, also known as 'Brownian motion' is a random neutral process with unbounded traits values. It is characterized using a single variance parameter called sigma that dictates to what extent the trait value fluctuates randomly in time.

The Ornstein-Uhlenbeck process is a random process with attractors (optima), with values varying about these optima (or single optimum). It is notable that whenever multiple optima are possible, only a single optimum is effective at any given time. The Ornstein-Uhlenbeck process involve two more values: the trait optimum effective at a particular time and the selection rate (alpha). The optima represents the value toward which the trait is attracted, whereas the value of alpha dictates to what extent such an attraction occurs. When alpha = 0, the optimum stops being an attractor and the Ornstein-Uhlenbeck process becomes purely a Wiener process, whereas when alpha is very large, the trait value detracts only slightly from its optimal value.

The trait simulation interface enabled to handle traits evolving neutrally (pure Wiener process), and according to an Ornstein-Uhlenbeck process with one or many optima. When many optima are used, transitions among the values are simulated as a Markov process. For that purpose, the simulator needs to be provided with a transition density matrix.

When multiple selection regimes are to be simulated, argument tem is given a list of quantitative trait evolution models instead of a single model. In that case, function simulateTrait will expect the graph (argument x) to have a edge property called 'tem'. This property contains the indices of the trait evolution models for each edge of the graph. In practice, to have reasonable chances to resolve multiple evolution regimes, the number of selection regimes has to be small with respect to the number of vertices.

Value

Function `traitEvolSim` returns a QTEM-class object, which consists of 8 member functions:

`$getName` It takes no argument and returns the name of the trait.

`$getStep` It takes no argument and returns the size of the simulation time step.

`$setStep` It takes the time step as an argument and sets, the simulation time step, including the recalculation of the transition intensity matrix in the case of a trait with multiple optima.

`$getOptima` It has no argument and returns the trait optimum values.

`$getTransition` It takes no argument and returns the transition density matrix.

`$getProb` It takes no argument and returns the transition probability matrix.

`$updateState` It takes the index of the actual trait optimum and returns the updated index (allows shifts in trait optimum).

`updateValue` It takes the value of the trait and, optionally, the index of the optimum state, at returns the updated trait value.

`$dumpConfig` It takes no argument and returns the trait evolution simulator parameters as a list.

`$print` It takes no argument and prints the parameters of the trait simulator on the screen (returns NULL invisibly).

Function `simulateTrait` returns a two-column `data.frame`:

`state` The simulated index of the trait optimum at the vertices of the graph.

`value` The simulated trait value at the vertices of the graph.

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

Examples

```
## Setting the RNG (for example consistency):
set.seed(2182955)

## A list to contain the trait simulator objects:
tem <- list()

## The first simulator is non-neutral with three optima:
traitEvolSim(
  name = "Trait 1",
  sigma = 1.5,
  alpha = 0.15,
  optima = c(30,50,80),
  transition = matrix(c(NA,0.1,0.0,0.1,NA,0.1,0.0,0.1,NA), 3L, 3L)
) -> tem[[1]]
tem[[1]]

## The second simulator is neutral:
traitEvolSim(
```

```

    name = "Trait 2",
    sigma = 2.5,
    step = 1
) -> tem[[2]]
tem[[2]]

## The third simulator is non-neutral with a single optimum:
traitEvolSim(
  name = "Trait 3",
  alpha = 0.05,
  optima = 15
) -> tem[[3]]
tem[[3]]

## The fourth simulator is also non-neutral with a single optimum (but having
## a different value than the latter):
traitEvolSim(
  name = "Trait 4",
  sigma = 2.0,
  alpha = 0.25,
  optima = -25
) -> tem[[4]]
tem[[4]]

## Each simulator has a set of embedded member functions that can be called
## directly.

## Member $getName() returns the name of the trait as follows:
unlist(lapply(tem, function(x) x$getName()))

## Member $getStep() returns the step sizes for the traits as follows:
unlist(lapply(tem, function(x) x$getStep()))

## Member $setStep() sets the step sizes as follows:
lapply(tem, function(x) x$setStep(0.1))

## and returns the recalculated transition probability matrix of simulators
## having a transition intensity matrix (i.e., for non-neutral simulators
## with multiple trait optima).

## This is the modified step sizes:
unlist(lapply(tem, function(x) x$getStep()))

## Member $getOptima() returns the simulator's optimum (if available) or a
## vector of optima (if multiple optima are available) as follows:
lapply(tem, function(x) x$getOptima())

## Member $getTransition() returns the transition intensity matrix, when
## available (NULL otherwise) as follows:
lapply(tem, function(x) x$getTransition())

## Member $getProb() returns the transition intensity matrix, whenever
## available (NULL otherwise) as follows:

```

```

lapply(tem, function(x) x$getProb())

## When multiple optima are available, member $updateState() enables to
## simulate the transition from one optimal trait state (the one given as the
## argument) to another (the one which is returned by the function):
state <- 1
newstate <- tem[[1]]$updateState(state)
newstate

## Member $updateValue() simulates the evolution of the trait from one value
## to another. For a non-neutral with multiple optima, The trait state is
## provided using argument state (default: 1, which is the only applicable
## value for a single optimum).
oldvalue <- 31.5
newvalue <- tem[[1]]$updateValue(oldvalue, state=1)
newvalue

## Member $dumpConfig() returns the configuration list, which can be used

cfg <- lapply(tem, function(x) x$dumpConfig())

## The trait evolution simulators can be re-instantiated as follows:
lapply(
  cfg,
  function(x)
    traitEvolSim(
      name = x$name,
      sigma = x$sigma,
      step = x$step,
      alpha = x$alpha,
      optima = x$optima,
      transition = x$transition
    )
) -> tem_clone

## Clean up:
rm(cfg, tem_clone)

## Simulate trait evolution using the four simulators described previously:

## Set step size to 0.05
lapply(tem, function(x, a) x$setStep(a), a = 0.05)

## Results list:
res <- NULL
trNms <- lapply(tem, function(x) x$getName())
res$state <- matrix(NA, 1001L, length(tem), dimnames = list(NULL, trNms))
res$optim <- matrix(NA, 1001L, length(tem), dimnames = list(NULL, trNms))
res$trait <- matrix(NA, 1001L, length(tem), dimnames = list(NULL, trNms))
rm(trNms)

## res$state contains the trait state at the simulation time
## res$optim contains the trait's optimum value at the simulation time

```

```

## res$trait contains the trait value at the simulation time

## Setting the optimal state of the four traits at the beginning of the
## simulation period:
res$state[1L,] <- c(2,NA,1,1) ## NBL trait #2 evolves neutrally.

## Getting the trait optima at the beginning of the simulation period:
unlist(
  lapply(
    tem,
    function(x)
      x$getOptima()[res$state[1L,x$getName()]]
  )
) -> res$optim[1L,]

## Setting the initial trait values:
res$trait[1L,] <- c(50,0,15,-25)

## The state of the simulation at the beginning of the simulation:
head(res$state)
head(res$optim)
head(res$trait)

## Setting RNG state to obtain
set.seed(1234567)

## This loop simulates time steps #2 through #1001
for(i in 2L:1001L) {

  ## Simulate the evolution of the trait states (if relevant, which it is
  ## only for the first trait):
  unlist(
    lapply(
      tem,
      function(x)
        x$updateState(res$state[i - 1L,x$getName()])
    )
  ) -> res$state[i,]

  ## Obtain the optimal trait value (relevant for all traits but the second,
  ## trait, which evolves neutrally):
  unlist(
    lapply(
      tem,
      function(x)
        x$getOptima()[res$state[i,x$getName()]]
    )
  ) -> res$optim[i,]

  ## Simulate the evolution of the trait value:
  unlist(
    lapply(
      tem,

```

```

        function(x)
          x$updateValue(
            state = res$state[i,x$getName()],
            value = res$trait[i - 1L,x$getName()]
          )
        )
      ) -> res$trait[i,]
    }

## Plot the results:
par(mar=c(4,4,1,1))
plot(NA, xlim=c(0,0.05*1000), ylim=range(res$trait), xlab="Time",
      ylab="Trait value", las=1L)
lines(x=0.05*(0:1000), y=res$trait[,1L], col="black")
if(!is.na(tem[[1L]]$getOptima())[1L])
  lines(x=0.05*(0:1000), y=res$optim[,1L], col="black", lty=3L)
lines(x=0.05*(0:1000), y=res$trait[,2L], col="red")
if(!is.na(tem[[2L]]$getOptima())[1L])
  lines(x=0.05*(0:1000), y=res$optim[,2L], col="red", lty=3L)
lines(x=0.05*(0:1000), y=res$trait[,3L], col="blue")
if(!is.na(tem[[3L]]$getOptima())[1L])
  lines(x=0.05*(0:1000), y=res$optim[,3L], col="blue", lty=3L)
lines(x=0.05*(0:1000), y=res$trait[,4L], col="green")
if(!is.na(tem[[4L]]$getOptima())[1L])
  lines(x=0.05*(0:1000), y=res$optim[,4L], col="green", lty=3L)

## A linear evolutionary sequence with random edge lengths between 2 and 5:
randomGraph(
  NV = 100,
  NC = function(...) 1,
  NP = function(...) 1,
  timestep = function(ts_min, ts_max, ...) runif(1, ts_min, ts_max),
  maxDist = function(...) NULL,
  ts_min = 2,
  ts_max = 5
) -> gr_lin

## Simulate a trait (Ornstein Uhlenbeck):
simulateTrait(
  x = gr_lin,
  tem = tem[[1]],
  state = 2,
  value = 50,
  a = 1
) -> simTrait

## Showing the trait values with the optima (OU process):
x <- cumsum(c(0,attr(gr_lin,"edge")$distance))
plot(x=x, y=simTrait$value, type="l", las=1)
lines(x=x, y=c(30,50,80)[simTrait$state], lty=3)

## Simulate an other trait (Brownian motion):
simulateTrait(

```

```

    x = gr_lin,
    tem = tem[[2]],
    value = 10,
    a = 1
  ) -> simTrait

## Showing the trait values:
plot(x=x, y=simTrait$value, type="l", las=1)

## A distance-based network:
N <- 100
coords <- cbind(x=runif(N,-1,1), y=runif(N,-1,1))
rownames(coords) <- sprintf("N%d",1:N)
dst <- dist(coords)
gr_dst <- dstGraph(d=dst, th=0.35, origin=15)

## Simulate a trait (Brownian motion) on the distance-based network:
simulateTrait(
  x = gr_dst,
  tem = tem[[2]],
  value = 0,
  a = 1
) -> simTrait

## Showing the results of the simulation:
gp <- par(no.readonly = TRUE)
par(mar=c(5,5,1,1))

plot(NA, xlim=c(-1,1), ylim=c(-1,1), asp=1, las=1, xlab="X", ylab="Y")
points(x=coords[, "x"], y=coords[, "y"], pch=21, cex=abs(simTrait$value)/3,
       bg=gray(0.5 - 0.5*sign(simTrait$value)))

par(gp)

```

write.fasta

Write Sequences to a FASTA Text File

Description

A function to write a vector of strings representing nucleotide sequences into a text file, with optional line breaks

Usage

```
write.fasta(x, file, linebreak = NULL, sep = "\n")
```

Arguments

x	A character vector storing the nucleotide sequences.
file	A character string describing the connection (see the description of argument description in function file help page).
linebreak	Interval for line breaking to improve file readability (default: NULL, meaning no line breaking).
sep	Character used for line breaking (default: "\n").

Value

NULL (invisibly).

Author(s)

Guillaume Guénard [aut, cre] (ORCID: <<https://orcid.org/0000-0003-0761-3072>>), Pierre Legendre [ctb] (ORCID: <<https://orcid.org/0000-0002-3838-3305>>)

Examples

```
## Define a raw vector for storing nucleotide values:
c(Sequence_1 = "ATCG-TTTCG--CCCCA--TTA--TTAA-GTAA-GTAATCTTTCA",
  Sequence_2 = "TTGGCTTCC--TC--CAT-TTC--TTCA-GT-ACG-ATTCTTTTA",
  Sequence_3 = "TTCG-TACC-T-T---A-ATAA--T-AA-GTCTTGTAATCGTTCA") %>%
  sapply(charToRaw) %>%
  t -> sqn

## Display the raw sequence:
sqn

## Transforming the sequence to character strings
tmp <- concatenate(sqn)
tmp

## Transforming the sequence to character strings without the gaps:
concatenate(sqn, discard="-")

## Write the sequences into a text file:
write.fasta(tmp, file="Sequences.fst", linebreak=15)

## Clean-up:
file.remove("Sequences.fst")
rm(sqn, tmp)
```

Index

`$<-.graph` (graph-class), 7

`add.edge` (graph-functions), 13
`add.vertex` (graph-functions), 13
`aggregateOnVertex` (PEM-utils), 46
`anova.pemlm` (pemlm-class), 49
`arrows`, 9, 21
`as.data.frame`, 28, 40, 50
`as.data.frame.PEM` (PEM-class), 38
`as.graph`, 11
`as.graph` (MPSEM-generics), 36
`as.graph.phylo` (graph-class), 7
`as.matrix`, 40
`as.matrix.PEM` (PEM-class), 38
`as.phylo`, 11
`as.phylo.graph` (graph-class), 7

`character`, 63
`concatenate`, 3

`data.frame`, 8, 9, 13, 24, 37, 47, 57
`dist`, 4, 19
`dist.dna`, 4
`DNArate` (molEvolSim), 29
`drawDNASequence` (molEvolSim), 29
`drawEvolRate` (molEvolSim), 29
`dstGraph`, 4

`edge` (MPSEM-generics), 36
`edge.InflMat` (InflMat-class), 24
`edge<-` (MPSEM-generics), 36
`edge<- .InflMat` (InflMat-class), 24
`edgenames` (MPSEM-generics), 36
`edgenames.InflMat` (InflMat-class), 24
`edgenames<-` (MPSEM-generics), 36
`edgenames<- .InflMat` (InflMat-class), 24
`evolution.model`, 40
`evolution.model` (MPSEM-generics), 36
`evolution.model.PEM`, 28
`evolution.model.PEM` (PEM-class), 38

`file`, 63
`formula`, 28, 50

`geometry` (MPSEM-generics), 36
`geometry.graph` (graph-class), 7
`geometry<-` (MPSEM-generics), 36
`geometry<- .graph` (graph-class), 7
`getConnected` (graph-utils), 17
`getMedian` (graph-utils), 17
`getNonConnected` (graph-utils), 17
`getOrigin` (graph-utils), 17
`getTerminal` (graph-utils), 17
`graph` (graph-functions), 13
`graph-class`, 7
`graph-functions`, 2, 13
`graph-purge`, 15
`graph-utils`, 17
`graphDist`, 16
`graphDist` (graph-utils), 17
`graphModplot`, 21

`harmonicMean`, 22
`hmean` (harmonicMean), 22

`InflMat`, 2
`InflMat` (InflMat-class), 24
`inflMat`, 40
`inflMat` (InflMat-class), 24
`InflMat-class`, 24
`invDistWeighting`, 18, 26, 52, 56
`isDivergent` (graph-utils), 17
`isLinear` (graph-utils), 17
`isTree` (graph-utils), 17
`isUnderEdge` (PEM-utils), 46
`isUnderVertex` (PEM-utils), 46

`labels<-` (MPSEM-generics), 36
`lm`, 28, 50
`lm-utils`, 27
`locate` (MPSEM-generics), 36

locate.graph, [40](#), [41](#)
locate.graph (graph-class), [7](#)
logical, [9](#), [10](#), [28](#), [50](#)

matrix, [47](#)
model.data (lm-utils), [27](#)
model.matrix.default, [28](#), [50](#)
molEvolSim, [29](#)
MPSEM-generics, [36](#)
MPSEM-package, [2](#)

name, [8](#)
nedge (MPSEM-generics), [36](#)
nedge.graph (graph-class), [7](#)
nedge.InflMat (InflMat-class), [24](#)

PEM, [9](#)
PEM (PEM-class), [38](#)
PEM-class, [38](#)
PEM-functions, [45](#)
PEM-utils, [46](#)
pemlm (pemlm-class), [49](#)
pemlm-class, [49](#)
PEMweights (PEM-functions), [45](#)
plot, [54](#)
plot.graph (graph-class), [7](#)
plot.phylo, [11](#)
plot.sf, [11](#)
points, [8](#), [21](#)
predict, [40](#)
predict.PEM (PEM-class), [38](#)
predict.pemlm (pemlm-class), [49](#)
print.graph (graph-class), [7](#)
print.InflMat (InflMat-class), [24](#)
print.PEM (PEM-class), [38](#)
print.pemlm (pemlm-class), [49](#)
print.QTEM (traitEvolSim), [55](#)
Psquare (lm-utils), [27](#)
purge.median (graph-purge), [15](#)
purge.terminal (graph-purge), [15](#)

randomGraph, [52](#)
raw, [3](#)
reorderGraph (graph-utils), [17](#)
rm.edge (graph-functions), [13](#)
rm.vertex (graph-functions), [13](#)

show.sequence, [54](#)
simulateSequence (molEvolSim), [29](#)

simulateTrait (traitEvolSim), [55](#)
summary.pemlm (pemlm-class), [49](#)

text, [54](#)
traitEvolSim, [55](#)

update.PEM (PEM-class), [38](#)
utils (lm-utils), [27](#)

weighted.hmean (harmonicMean), [22](#)
write.fasta, [62](#)