Package ‘MPSEM’

July 21, 2025

Type Package

Encoding UTF-8

Title Modelling Phylogenetic Signals using Eigenvector Maps

Version 0.6-1

Date 2025-05-24

Description Computational tools to represent phylogenetic signals using adapted eigenvector maps.

Depends R (>= 3.0.0), ape, magrittr

Suggests knitr, caper, xfun

Imports MASS

License GPL-3

LazyLoad yes

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

RoxygenNote 7.3.1

Author 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@gmail.com`>

Date/Publication 2025-05-24 22:00:02 UTC

Contents

`MPSEM-package` *Modelling Phylogenetic Signals using Eigenvector Maps*

Description

Computational tools to represent phylogenetic signals using adapted eigenvector maps.

Details

Phylogenetic eignevector 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 inducted by evolution. These patterns are used to model the data, using a linear model for instance.

If one in 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.

Functions `PEM.build`, `PEM.updater`, `PEM.fitSimple`, and `PEM.forcedSimple` allow one to build, update (i.e. recalculate with alternative weighting parameters) as well as to estimate or force arbitrary values for the weighting function parameters.

Functions `getGraphLocations` and `Locations2PEMscores` allow one to make predictions using method `predict.PEM` and a linear model. To obtain this linear model, one can use either function `lm` or auxiliary functions `lmforwardsequentialsidak` or `lmforwardsequentialAICc`, which perform forward-stepwise variable addition on the basis of either familiwise type I error rate or the Akaike Information Criterion (AIC), respectively.

The package provides low-level utility functions for performing operations on graphs (see graph-functions), calculate influence matrix (`InflMat`), and simulate trait values (see trait-simulator).

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

From version 0.6-1, no new features will be added to MPSEM. A new package is currently under development to replace MPSEM as a platform for future PEM development. We will continue to maintain MPSEM to provide backward compatibility for old scripts. The release of the new package under development will be announced here along with reference to any relevant research article.

The DESCRIPTION file:

Package:	MPSEM
Type:	Package
Encoding:	UTF-8
Title:	Modelling Phylogenetic Signals using Eigenvector Maps
Version:	0.6-1
Date:	2025-05-24

Authors@R:	c(person(given = "Guillaume", family = "Guénard", role = c("aut","cre"), email = "guillaume.guenard
Description:	Computational tools to represent phylogenetic signals using adapted eigenvector maps.
Depends:	R (>= 3.0.0), ape, magrittr
Suggests:	knitr, caper, xfun
Imports:	MASS
License:	GPL-3
LazyLoad:	yes
NeedsCompilation:	yes
VignetteBuilder:	knitr
Repository:	CRAN
RoxygenNote:	7.3.1
Author:	Guillaume Guénard [aut, cre] (ORCID: <https://orcid.org/0000-0003-0761-3072>), Pierre Legendre [ct
Maintainer:	Guillaume Guénard <guillaume.guenard@gmail.com>

Index of help topics:

```
MPSEM-package           Modelling Phylogenetic Signals using
                        Eigenvector Maps
PEM-class               Class and Methods for Phylogenetic Eigenvector
                        Maps (PEM)
PEM-functions           Phylogenetic Eigenvector Maps
dstGraph                Distance-Based Directed Graph
graph-class             Class and Method for Directed Graphs
graph-functions         MPSEM graph Manipulation Functions
lm-utils                Linear Modelling Utility Functions
trait-simulator         Simulate the Evolution of a Quantitative Trait
```

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@gmail.com>

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

Makarenkov, V., Legendre, P. & Desdevise, 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")
```

`dstGraph` *Distance-Based Directed Graph*

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@gmail.com>

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. & Desdevise, Y. 2004. Modelling phylogenetic relationships using reticulated networks. Zoologica Scripta 33: 89-96

Examples

```
## A simple example

## Create a 10-vertex graph with 16 edges:
## The vertices have x and y coordinates to help in plotting the graph.
pop.graph(
  n = 10,
  vertex = list(
    species = rep(TRUE,10),
    x = c(0,1,1,2,2,2,3,3,4,4),
    y = c(0,1,-1,2,0,-2,1,-0.5,0,-2)
  ),
  label = sprintf("V%d",1:10)
) %>%
  add.edge(
    from = c(1,1,2,2,3,3,4,5,5,1,7,7,8,6,9,8),
    to = c(2,3,4,5,5,6,7,7,8,7,6,9,9,9,10,10),
    edge = list(distance=c(1,1,1,1,1,1,1,1,1,4,2,1,1,3,1,1)),
    label = sprintf("E_%d",1:16)
  ) -> x

## Plotting the graph:
plot(x=x$vertex$x, y=x$vertex$y, type="n")
for(i in 1:attr(x,"ev")[1])
  arrows(x0=x$vertex$x[x$edge[[1]][i]], x1=x$vertex$x[x$edge[[2]][i]],
         y0=x$vertex$y[x$edge[[1]][i]], y1=x$vertex$y[x$edge[[2]][i]],
         length=0.2)
points(x=x$vertex$x, y=x$vertex$y, pch=21, bg="white", cex=3)

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

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

## Generate a PEM for that graph:
pem_x <- PEM.build(x)

## Plotting the different eigenvectors one-by-one on the graph plot:
for(i in 1:ncol(pem_x$u)) {
  v <- pem_x$u[,i]
  plot(x=x$vertex$x, y=x$vertex$y, type="n", main=colnames(pem_x$u)[i])
  for(i in 1:attr(x,"ev")[1])
    arrows(x0=x$vertex$x[x$edge[[1]][i]], x1=x$vertex$x[x$edge[[2]][i]],
```

```
           y0=x$vertex$y[x$edge[[1]][i]], y1=x$vertex$y[x$edge[[2]][i]],
           length=0.2)
  points(x=x$vertex$x, y=x$vertex$y, pch=21,
         bg=gray(c(0.9,0.1))[1 + (sign(v) + 1)/2], cex=7*abs(v))
  if(is.null(locator(1))) break
}

## A more elaborate example

## Here, we set the seed to obtain a consistent example, but feel free 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$vertex$order) + 1), max(gr$vertex$order))
for(i in 1L:attr(gr,"ev")[1])
  arrows(x0=coords[gr$edge[[1]][i],1], x1=coords[gr$edge[[2]][i],1],
         y0=coords[gr$edge[[1]][i],2], y1=coords[gr$edge[[2]][i],2],
         length=0.05, col=col[gr$vertex$order[gr$edge[[2]][i]]])
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$vertex$order) + 1), max(gr$vertex$order))
for(i in 1L:attr(gr,"ev")[1])
  arrows(x0=coords[gr$edge[[1]][i],1], x1=coords[gr$edge[[2]][i],1],
         y0=coords[gr$edge[[1]][i],2], y1=coords[gr$edge[[2]][i],2],
         length=0.05, col=col[gr$vertex$order[gr$edge[[2]][i]]])
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$vertex$order) + 1), max(gr$vertex$order))
for(i in 1L:attr(gr,"ev")[1])
  arrows(x0=coords[gr$edge[[1]][i],1], x1=coords[gr$edge[[2]][i],1],
         y0=coords[gr$edge[[1]][i],2], y1=coords[gr$edge[[2]][i],2],
         length=0.05, col=col[gr$vertex$order[gr$edge[[2]][i]]])
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)

## Generate a PEM for that graph:
pem_gr <- PEM.build(gr)

## Plotting the different eigenvectors one-by-one on the graph plot:
for(i in 1:ncol(pem_gr$u)) {
  v <- pem_gr$u[,i]
  plot(coords, type="n", asp=1, main=colnames(pem_gr$u)[i])
  col <- head(rainbow(max(gr$vertex$order) + 1), max(gr$vertex$order))
  for(j in 1L:attr(gr,"ev")[1])
    arrows(x0=coords[gr$edge[[1]][j],1], x1=coords[gr$edge[[2]][j],1],
           y0=coords[gr$edge[[1]][j],2], y1=coords[gr$edge[[2]][j],2],
           length=0.05, col=col[gr$vertex$order[gr$edge[[2]][j]]])
  points(coords, pch=21, bg=gray(c(0.9,0.1))[1 + (sign(v) + 1)/2],
         cex=10*abs(v))
  if(is.null(locator(1L))) break
}
```

`graph-class` *Class and Method for Directed Graphs*

Description

Class and methods to handle MPSEM graphs.

Usage

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

Arguments

`x`	An object of `graph-class`.
`...`	Additional parameters to be passed to the method. Currently ignored.

Format

A `graph-class` object contains:

edge A list whose first two unnamed members are the indices of the origin and destination vertices. Additional members must be named and are additional edge properties (e.g. length).

vertex A list that optionally contains vertex properties, if any (or an empty `list` if none).

Details

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

Functions

- `print(graph)`: Print Graph

  A print method for graph-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@gmail.com>

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`.

`graph-functions` *MPSEM graph Manipulation Functions*

Description

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

Usage

```
pop.graph(n, vertex = list(), label = NULL)

add.vertex(x, n, vertex = list(), label = NULL)

add.edge(x, from, to, edge = list(), label = NULL)
```

```
rm.edge(x, id)

rm.vertex(x, id)

collapse.vertex(x, id)

Phylo2DirectedGraph(tp)
```

Arguments

`n`	The number of vertices to populate a new graph (`pop.graph`) or to add to an existing graph (`add.vertex`).
`vertex`	A list of vertex properties.
`label`	Labels to be given to edges or vertices.
`x`	A `graph-class` object.
`from`	The origins of the edges to be added (vertex labels or indices).
`to`	The destinations of the edges to be added (vertex labels or indices).
`edge`	A list of edge properties.
`id`	Indentity (label or index) of vertex or edge to be removed.
`tp`	Phylogenetic tree object of class ‘phylo’, as defined in `ape-package`.

Details

A new graph can be populated with `n` vertices using function `pop.graph`. Additional vertices can be added later with function `add.vertex`. The graphs so created contain no edges; the latter are added using function `add.edge`. Vertices and edges are removed using functions `rm.vertex` and `rm.edge`, respectively.

Function `collapse.vertex` allows one to remove a vertex while reestablishing the connections between the vertices located above and below that vertex using a new set of edges.

Function `Phylo2DirectedGraph` uses the MPSEM graph functions to convert a rooted 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 acyclic graph can then be edited to represent cases that do not have a tree topology.

Value

The function returns a `graph-class` object. Objects returned by `Phylo2DirectedGraph` have a `numeric` edge property called ‘distance’ featuring branch lengths, and a `link{logical}` vertex property called ‘species’ specifying whether a vertex is a tree tip or an internal node.

Functions

- `pop.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.
- `collapse.vertex()`: Collapse Vertices
  Remove vertices from a graph: remove vertices together with their associated edges.
- `Phylo2DirectedGraph()`: Phylogenetic Tree Conversion
  Create a new `graph-class` object from a phylo-class object (phylogenetic tree).

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@gmail.com>

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. & Desdevise, 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

`graph-class`.

Examples

```
## Populate a graph with 7 vertices labeled A-G having properties x and y:
gr <- pop.graph(n=7,
                vertex=list(x=rnorm(7,0,1),y=rnorm(7,0,1)),
                label=c("A","B","C","D","E","F","G"))
gr

## Adding 3 vertices H, I, and J with property x (y is absent) and a new
## property z (type character), which is unknown for A-G:
gr <- add.vertex(x=gr,
                 n=3,
                 label=c("H","I","J"),
                 vertex=list(x=rnorm(3,0,1),z=c("A","B","C")))
gr
```

```
gr$vertex

## Adding 10 edges, labeled E1-E10 and with properties a and b, to the graph:
gr <- add.edge(x=gr,
               from=c("A","B","B","C","C","D","D","E","E","F"),
               to=c("A","C","D","E","F","F","G","H","I","J"),
               edge=list(a=rnorm(10,0,1),b=rnorm(10,0,1)),
               label=paste("E",1:10,sep=""))
gr
gr$edge

## Removing edges 2, 4, and 7 from the graph:
print(rm.edge(gr,id=c(2,4,7)))

## Removing vertices 1, 3, 7, and 10 from the graph:
print(rm.vertex(gr,id=c(1,3,7,10)))
# Notice that the edges that had one of the removed vertex as their
# origin or destination are also removed:
print.default(rm.vertex(gr,id=c(1,3,7,10)))

## Vertex collapsing.
x <- pop.graph(n=9,label=c("A","B","C","D","E","F","G","H","I"))
x <- add.edge(x,from=c("A","A","B","B","C","C","D","D","E","E"),
              to=c("B","C","D","E","E","I","F","G","G","H"),
              label=paste("E",1:10,sep=""),
              edge=list(length=c(1,2,3,2,1,3,2,2,1,3)))
print.default(x)
for(i in c("A","B","C","D","E","F","G","H","I"))
  print(collapse.vertex(x,id=i))

if(require(ape)) {
  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;",sep=""))
  x <- Phylo2DirectedGraph(tree1)
  print(x)
}
```

`lm-utils` *Linear Modelling Utility Functions*

Description

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

Usage

```
lmforwardsequentialAICc(y, x, object)

lmforwardsequentialsidak(y, x, object, alpha = 0.05)
```

Arguments

`y`	A response variable.
`x`	Descriptors (numeric of `factor`) to be used as auxiliary traits.
`object`	A `PEM-class` object.
`alpha`	The p-value threshold above which the function will stop adding variables.

Details

Function `lmforwardsequentialsidak`, performs a forward stepwise selection of the PEM eigenvectors until the familywise test of significance of the new variable to be included exceeds the p-value threshold `alpha`. The familiwise type I error probability is obtained using the Holm-Sidak correction of the testwise probabilities, thereby correcting for type I error rate inflation due to multiple testing.

Function `lmforwardsequentialAICc` carries out forward stepwise selection of the eigenvectors as long as the candidate model features a sample-size-corrected Akaike information criterion lower than the previous model. The final model should be regarded as overfitted from the Neyman-Pearson (*i.e.* frequentist) point of view, but this is the model that minimizes information loss from the standpoint of information theory.

Value

An `lm-class` object.

Functions

- `lmforwardsequentialAICc()`: Forward Stepwise Regression AICc

  Forward stepwise variable addition using the sample-size-corrected Akaike Information Criterion.

- `lmforwardsequentialsidak()`: Forward Stepwise Regression Sidak

  Forward stepwise variable addition using a Sidak multiple testing corrected alpha error threshold as the stopping criterion.

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@gmail.com>

References

Burnham, K. P. & Anderson, D. R. 2002. Model selection and multimodel inference: a practical information-theoretic approach, 2nd ed. Springer-Verlag. xxvi + 488 pp.

Holm, S. 1979. A simple sequentially rejective multiple test procedure. Scand. J. Statist. 6: 65-70

Sidak, Z. 1967. Rectangular confidence regions for means of multivariate normal distributions. J. Am. Stat. Ass. 62, 626-633

`PEM-class` *Class and Methods for Phylogenetic Eigenvector Maps (PEM)*

Description

Class and methods to handle Phylogenetic Eigenvector Maps (PEM).

Usage

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

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

## S3 method for class 'PEM'
predict(
  object,
  targets,
  lmobject,
  newdata,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

`x`	A `PEM-class` object containing a Phylogenetic Eigenvector Map.
`...`	Additional parameters to be passed to the method. Currently ignored.
`row.names`	Included for method consistency reason; ignored.
`optional`	Included for method consistency reason; ignored.
`object`	A `PEM-class` object.
`targets`	Output of `getGraphLocations`.
`lmobject`	An object of class ‘lm’ (see `lm` for details).
`newdata`	Auxiliary trait values.
`interval`	The kind of limits (confidence or prediction) to return with the predictions; `interval="none"`: do not return a confidence interval.
`level`	Probability associated with the confidence of prediction interval.

Format

A `PEM-class` object contains:

x The `graph-class` object that was used to build the PEM (see `PEM.build`).

sp A `logical` vector specifying which of the vertices are tips.

B The influence matrix for those vertices that are tips.

ne The number of edges.

nsp The number of species (tips).

Bc The column-centred influence matrix.

means The column means of `B`.

dist Edge lengths.

a The steepness parameter (see `PEM.build` for details).

psi The relative evolution rate along the edges (see `PEM.build` for details).

w Edge weights.

BcW The weighted and column-centred influence matrix.

d The singular values of `BcW`.

u The eigenvectors (left singular vectors) of `BcW`.

vt The right singular vectors of `BcW`.

In addition to these standard component, function, `PEM.fitSimple` and `PEM.forcedSimple` add the following members, which are necessary to make predictions:

S2 The variances of response data (one value for each response variable).

y A copy of the response data.

opt The list returned by `optim`.

The estimated weighting parameters are also given as an edge property.

Details

The `print.PEM` method provides the number of eigenvectors, the number of observations these vectors are spanning, and their associated eigenvalues.

The `as.data.frame.PEM` method extracts the eigenvectors from the object and allows one to use `PEM-class` objects as `data` parameter in function such as `lm` and `glm`.

The `predict.PEM` method is a barebone interface to make predictions. It must be given species locations with respect to the phylogenetic graph (`target`), which are provided by function `getGraphLocations` and a linear model in the form of an object from `lm`. The user must provide auxiliary trait values if `lmobject` involves such traits.

Functions

- `print(PEM)`: Print PEM-class
  A print method for PEM-class objects.
- `as.data.frame(PEM)`: Method `as.data.frame` for PEM-class Objects
  A method to extract the phylogenetic eigenvectors from a PEM-class object.
- `predict(PEM)`: Predict Method for PEM-class Objects
  A predict method to predict species trait values using Phylogenetic Eigenvector Maps.

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@gmail.com>

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`

`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

```
InflMat(x)

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

PEM.build(
  x,
  d = "distance",
  sp = "species",
  a = 0,
  psi = 1,
  tol = .Machine$double.eps^0.5
)

PEM.updater(object, a, psi = 1, tol = .Machine$double.eps^0.5)

PEM.fitSimple(
  y,
  x,
  w,
  d = "distance",
  sp = "species",
  lower = 0,
```

```
  upper = 1,
  tol = .Machine$double.eps^0.5
)

PEM.forcedSimple(
  y,
  x,
  w,
  d = "distance",
  sp = "species",
  a = 0,
  psi = 1,
  tol = .Machine$double.eps^0.5
)

getGraphLocations(tpall, targets)

getAncGraphLocations(x, tpall)

Locations2PEMscores(object, gsc)
```

Arguments

`x`	A `graph-class` object containing a phylogenetic graph.
`d`	The name of the member of `x$edge` where the phylogenetic distances (edge lengths) can be found.
`a`	The steepness parameter describing whether changes occur, on average: progressively long edges (a close to 0) or abruptly at vertices (a close to 1).
`psi`	Relative evolution rate along the edges (default: 1). This parameter is only relevant when multiple values are assigned to different portions of the phylogeny.
`sp`	Name of the member of `x$vertex` where a `logical` vertex property can be found, specifying which vertices are species (see `graph-class`).
`tol`	Eigenvalue threshold indicating that eigenvectors as usable.
`object`	A `PEM-class` object containing a Phylogenetic Eigenvector Map.
`y`	One or many response variable(s) in the form of a single numeric vector or a `matrix`, respectively.
`w`	A `graph-class` object containing a phylogenetic graph.
`lower`	Lower limit for the L-BFGS-B optimization algorithm implemented in `optim`.
`upper`	Upper limit for the L-BFGS-B optimization algorithm implemented in `optim`.
`tpall`	First parameter of function `getGraphLocations`: Phylogenetic tree object with class ‘phylo’ (package ape) containing all species (model and target) used in the study.
`targets`	Name of the target species to extract using the tree `tpall`.
`gsc`	The output of `getGraphLocations`.

Details

Functions `InflMat` and `PEMweights` are used internally by `PEM.build` to create a binary matrix referred to as an 'influence matrix' and weight its columns. That matrix has a row for each vertex (or node) of graph 'x' and a column for each of its edges. The elements of the influence matrix are 1 whenever the vertex associated with a row is located in the tree, either directly or indirectly downward the edge associated with a column. That function is implemented in `C` language using recursive function calls. Although `InflMat` allows one to use multiple roots. User must therefore make sure that the graph provided to `PEMap` is single-rooted.

Function `PEM.build` is used to produce a phylogenetic eigenvector map, while function `PEM.updater` allows one to re-calculate a `PEM-class` object with new weighting function parameters. Function `PEM.fitSimple` performs a maximum likelihood estimation of parameters `a` and `psi` assuming single values for the whole tree, whereas function `PEM.forcedSimple` allows one to impose values to arguments `a` and `psi` of a `PEM-class` object, while making the function produce the same details as `PEM.fitSimple` would have produced; these details are necessary to make predictions.

Functions `getGraphLocations` returns the coordinates of a species in terms of its position with respect to the influence matrix while function `Locations2PEMscores` transforms these coordinates into sets of scores that can be used to make predictions. Function `getAncGraphLocations` produces the same output as `getGraphLocations`, but for the ancestral species (i.e. the nodes of the phylogeny) in order to estimate ancestral trait values.

Value

Function `InflMat` returns the influence matrix of graph `x` and function `PEMweights` returns weights corresponding to the distances. Functions `PEM.build`, `PEM.fitSimple` and `PEM.forcedSimple` return a `PEM-class` object. Function `getGraphLocations` returns a list whose first member is an influence coordinate matrix whose rows refer to the target species and columns refer to the edges. The second member contains the lengths of the terminal edges connecting each target species to the rest of the phylogeny.

Function `Locations2PEMscores` returns a list whose first member is a PEM score matrix whose rows refer to the target species and columns refer to the eigenvectors. The second member contains the variance associated with the terminal edges connecting the target species to the phylogeny.

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 of extant species) during evolution (value = 1) or not (value = 0).

- `PEMweights()`: PEM Weighting

  A power function to obtain the edge weights used during PEM calculation.

- `PEM.build()`: PEM Building

  Calculates a PEM with parameters given by arguments a and psi.

- `PEM.updater()`: PEM Update

  Update a PEM with new parameters given by arguments a and psi.

- `PEM.fitSimple()`: Fitting a PEM to Data while Estimating Global Steepness

  Fits a PEM to a data set estimating the selection (steepness) parameter using gradient descent. The selection and evolution rate (psi = 1) are assumed to be homogeneous for the whole phylogenetic network.
- `PEM.forcedSimple()`: Fitting a PEM to Data while Forcing Global Steepness

  Fits a PEM to a data set forcing a user-provided selection (steepness) parameter. The selection and evolution rate (psi = 1) are assumed to be homogeneous for the whole phylogenetic network.
- `getGraphLocations()`: Get Phylogenetic Graph Locations

  Takes a phylogenetic tree and a list of species to be removed, and produce a phylogenic graph without these species together with the locations of the removed species on that graph (i.e., the location where the removed species would be found should they be inserted again in the phylogenetic graph).
- `getAncGraphLocations()`: Get Ancestral Species Location

  Get the location on the phylogenetic graph of the immediate ancestors for a list of species. The species of the list remain in the resulting phylogenetic graph. This function is useful for estimating the ancestral state of a trait.
- `Locations2PEMscores()`: PEM Score Calculation

  Calculates the scores of an extant or ancestral species on a phylogenetic eigenvector map (i.e., its value on the eigenvectors of the map) from its location on the phylogenetic graph used to build that map.

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@gmail.com>

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. & Desdevise, 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`

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;",sep=""))
t1                 # Summary of the structure of the tree
summary(t1)

x <- Phylo2DirectedGraph(t1)

## Calculate the (binary) influence matrix; E1 to E12 are the tree edges
## Edge E12 comes from the tree origin
InflMat(x)
InflMat(x)[x$vertex$species,]

## Building phylogenetic eigenvector maps
PEM1 <- PEM.build(x)
PEM2 <- PEM.build(x, a = 0.2)
PEM3 <- PEM.build(x, a = 1)
PEM4 <- PEM.updater(PEM3,a=0.5)

## Print summary statistics about PEM1
print(PEM1)

## Extract the eigenvectors (species A--G, 6 eigenvectors)
as.data.frame(PEM4)

## 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)

## Estimate a single steepness parameter for the whole tree
PEMfs1 <- PEM.fitSimple(y=y, x=NULL, w=x, d="distance", sp="species",
                        lower=0, upper=1)
PEMfs1$optim       # Optimisation results

## Force neutral evolution over the whole tree
PEMfrc1 <- PEM.forcedSimple(y=y,x=NULL,w=x,d="distance",sp="species",a=0)
PEMfrc1$x$edge$a   # Steepness parameter forced on each individual edge

## Graph locations for target species X, Y, and Z not found in the original
## data set
tpAll <- 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=""))
tpAll
summary(tpAll)     # Summary of the structure of the tree

grloc <- getGraphLocations(tpAll, c("X","Y","Z"))
grloc

PEMfs2 <- PEM.fitSimple(y=y, x=NULL, w=grloc$x, d="distance", sp="species",
```

```
                        lower=0, upper=1)
PEMfs2

## Same as for PEMfs1$optim
PEMfs2$optim

## Get the PEM scores from the species graph locations:
PEMsc1 <- Locations2PEMscores(PEMfs2, grloc)
lm1 <- lm(y ~ V_2 + V_3 + V_5, data=PEMfs2)

## Making prdictions for the species in locations `grloc`
## using linear model `lm1`:
ypred <- predict(object=PEMfs2, targets=grloc, lmobject=lm1, interval="none")

## Removing species X, Y, and Z from the tree in `tpAll`:
tpModel <- drop.tip(tpAll, c("X","Y","Z"))

## Plot the results
layout(t(c(1,1,2)))
par(mar=c(6,2,2,0.5)+0.1)
plot(tpModel, show.tip.label=TRUE, show.node.label=TRUE, root.edge = TRUE,
     srt = 0, adj=0.5, label.offset=0.08, font=1, cex=1.5, xpd=TRUE)
edgelabels(paste("E", 1:nrow(tpModel$edge), sep=""),
           edge=1:nrow(tpModel$edge), bg="white", font=1, cex=1)
points(x=0.20,y=2.25,pch=21,bg="black")
lines(x=c(0.20,0.20,0.65), y=c(2.25,0.55,0.55), xpd=TRUE, lty=2)
text("X",x=0.69, y=0.55, xpd=TRUE, font=1, cex=1.5)
points(x=0.35, y=4.5,pch=21,bg="black")
lines(x=c(0.35,0.35,0.6), y=c(4.5,5.47,5.47), xpd=TRUE, lty=2)
text("Y", x=0.64, y=5.47, xpd=TRUE, font=1, cex=1.5)
points(x=0.35, y=3, pch=21, bg="black")
lines(x=c(0.35,0.35,0.55), y=c(3,3.5,3.5), xpd=TRUE, lty=2)
text("Z", x=0.59, y=3.5, xpd=TRUE, font=1, cex=1.5)
text(c("NX","NY","NZ"), x=c(0.20,0.35,0.35), y=c(2.25,4.5,3)+0.3*c(1,-1,-1),
     font=1, cex=1)
add.scale.bar(length=0.1, cex=1.25)
par(mar=c(3.75,0,2,2)+0.1)
plot(x=y, y=1:7, ylim=c(0.45,7), xlim=c(-4,4), axes=FALSE, type="n", xlab="")
axis(1, label=c("-4","-2","0","2","4"), at=c(-4,-2,0,2,4))
abline(v=0)

## Plot the observed values
points(x=y, y=1:7, xlim=c(-2,2), pch=21, bg="black")
text("B)", x=-3.5, y=7, cex=1.5, xpd=TRUE)
text("Trait value", x=0, y=-0.5, cex=1.25, xpd=TRUE)

## Plot the predicted values
points(x=ypred, y=c(0.5,5.5,3.5), pch=23, bg="white", cex=1.25)

## Estimate the ancestral trait values
ANCloc <- getAncGraphLocations(x)
PEMfsAnc <- PEM.fitSimple(y=y, x=NULL, w=ANCloc$x, d="distance",
                          sp="species", lower=0, upper=1)
```

```
PEMfsAnc$optim

## Get the PEM scores from the species graph locations:
PEManc1 <- Locations2PEMscores(PEMfsAnc, ANCloc)

## Making predictions for the ancestral species whose locations are found in
## `ANCloc` using the linear model `lm1`:
y_anc <- predict(object=PEMfsAnc, targets=ANCloc, lmobject=lm1,
                 interval="confidence")
```

`trait-simulator` *Simulate the Evolution of a Quantitative Trait*

Description

Functions to simulate the evolution of a quantitative trait along a phylogenetic tree inputted as an object of class ‘phylo’ (package ape) or a `graph-class` object.

Usage

```
EvolveOptimMarkovTree(tp, tw, anc, p = 1, root = tp$edge[1, 1])

TraitOUsimTree(tp, a, sigma, opt, p = 1, root = tp$edge[1, 1])

OUvar(d, a = 0, theta = 1, sigma = 1)

PEMvar(d, a = 0, psi = 1)

TraitVarGraphSim(x, variance, distance = "distance", p = 1, ...)
```

Arguments

`tp`	A rooted phylogenetic tree of class ‘phylo’ (see package ape).
`tw`	Transition matrix giving the probability that the optimum trait value changes from one state (row) to another (column) at vertices. All rows must sum to 1.
`anc`	Ancestral state of a trait (at the root).
`p`	Number of variates to generate.
`root`	Root node of the tree.
`a`	Selection rate in function (`OUvar`) or steepness in (`PEMvar`).
`sigma`	Neutral evolution rate, i.e. mean trait shift by drift.
`opt`	An index vector of optima at the nodes.
`d`	Phylogenetic distances (edge lengths).
`theta`	Adaptive evolution rate, i.e. mean trait shift by natural selection.
`psi`	Mean evolution rate.

`x`	A `graph-class` object.
`variance`	Variance function: `OUvar`, `PEMvar`, or any other suitable user-defined function.
`distance`	The name of the member of ‘x$edge’ where edge lengths can be found.
`...`	Additional parameters for the specified variance function.

Details

Function `EvolveOptimMarkovTree` allows one to simulate the changes of optimum trait values as a Markov process. The index whereby the process starts, at the tree root, is set by parameter `anc`; this is the ancestral character state. From the root onwards to the tips, the optimum is given the opportunity to change following a multinomial random draw with transition probabilities given by the rows of matrix `tw`. The integers thus obtained can be used as indices of a vector featuring the actual optimum trait values corresponding to the simulated selection regimes.

The resulting optimum trait values at the nodes are used by `TraitOUsimTree` as its argument `opt` to simulate trait values at nodes and tips.

Function `TraitVarGraphSim` uses a graph variance function (either `OUvar` or `PEMvar`) to reconstruct a covariance matrix, used to generate covariates drawn from a multi-normal distribution.

Value

Functions `EvolveOptimMarkovTree` and `TraitOUsimTree` return a matrix whose rows represent the vertices (nodes and tips) of the phylogenetic tree and whose columns stand for the `n` different trials the function was asked to perform.

For `EvolveQTraitTree`, the elements of the matrix are integers, representing the selection regimes prevailing at the nodes and tips, whereas for `TraitOUsimTree`, the elements are simulated quantitative trait values at the nodes and tips. These functions are implemented in C language and therefore run swiftly even for large (10000+ species) trees.

Function `TraitVarGraphSim` returns `p` phylogenetic signals. It is implemented using a rotation of a matrix of standard normal random (mean=0, variance=1) deviates. The rotation matrix is itself obtained by Choleski factorization of the trait covariance matrix expected for a given set of trees, variance function, and variance function parameters.

Functions

- `EvolveOptimMarkovTree()`: Trait Optima Simulator
  Simulates the evolution of trait optima along a phylogeny as a Markov process.
- `TraitOUsimTree()`: Trait Value Simulator
  Simulates the evolution of trait values along a phylogeny as a Ornstein–Uhlenbeck process.
- `OUvar()`: Ornstein–Uhlenbeck Variance Calculator
  Calculates the expected covariance matrix for a trait evolving following an Ornstein–Uhlenbeck process. This function is meant to be used with function `TraitVarGraphSim`.
- `PEMvar()`: Phylogenetic Eigenvector Maps Variance Calculator
  Calculates the covariance on the basis of the covariance model (power function) associated used in calculating Phylogenetic Eigenvector Maps. This function is meant to be used with function `TraitVarGraphSim`.

- `TraitVarGraphSim()`: Covariance-based Trait Evolution Simulator.

  Simulates trait evolution as covariates drawn from a multi-normal distribution whose covariance is estimated using an external function (functions `OUvar`, `PEMvar` provided with the package or any user-provided function).

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@gmail.com>

References

Butler, M. A. & King, A. A. 2004. Phylogenetic comparative analysis: a modeling approach for adaptive evolution. American Naturalist 164: 683-695

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

```
opt <- c(-2,0,2) # Three trait optima: -2, 0, and 2
## Transition probabilities:
transit <- matrix(c(0.7,0.2,0.2,0.2,0.7,0.1,0.1,0.1,0.7),
                  length(opt),length(opt),dimnames=list(from=opt,to=opt))

## In this example, the trait has a probability of 0.7 to stay at a given
## optimum, a probability of 0.2 for the optimum to change from -2 to 0,
## from 0 to -2, and from 2 to -2, and a probability of 0.1 for the
## optimum to change from -2 to 2, from 0 to 2, and from 2 to 0.
nsp <- 25  # A random tree for 25 species.
tree2 <- rtree(nsp,tip.label=paste("Species",1:nsp,sep=""))
tree2$node.label=paste("N",1:tree2$Nnode,sep="")  # Node labels.

## Simulate 10 trials of optimum change.
reg <- EvolveOptimMarkovTree(tp=tree2,tw=transit,p=10,anc=2)
y1 <- TraitOUsimTree(tp=tree2,a=0,sigma=1,
                     opt=opt[reg[,1]],p=10)    ## Neutral
y2 <- TraitOUsimTree(tp=tree2,a=1,sigma=1,
                     opt=opt[reg[,1]],p=10)    ## Few selection.
y3 <- TraitOUsimTree(tp=tree2,a=10,sigma=1,
                     opt=opt[reg[,1]],p=10)    ## Strong selection.

## Display optimum change with colours.
displayOUprocess <- function(tp,trait,regime,mvalue) {
  layout(matrix(1:2,1,2))
  n <- length(tp$tip.label)
  ape::plot.phylo(tp,show.tip.label=TRUE,show.node.label=TRUE,root.edge=FALSE,
                  direction="rightwards",adj=0,
                  edge.color=rainbow(length(trait))[regime[tp$edge[,2]]])
  plot(y=1:n,x=mvalue[1:n],type="b",xlim=c(-5,5),ylab="",xlab="Trait value",yaxt="n",
       bg=rainbow(length(trait))[regime[1:n]],pch=21)
```

```
  text(trait[regime[1:n]],y=1:n,x=5,col=rainbow(length(trait))[regime[1:n]])
  abline(v=0)
}

displayOUprocess(tree2,opt,reg[,1],y1[,1])  # Trait evolve neutrally,
displayOUprocess(tree2,opt,reg[,1],y2[,1])  # under weak selection,
displayOUprocess(tree2,opt,reg[,1],y3[,1])  # under strong selection.

x <- Phylo2DirectedGraph(tree2)
y4 <- TraitVarGraphSim(x, variance = OUvar, p=10, a=5)

DisplayTreeEvol <- function(tp,mvalue) {
  layout(matrix(1:2,1,2))
  n <- length(tp$tip.label)
  ape::plot.phylo(tp,show.tip.label = TRUE, show.node.label = TRUE,
                  root.edge = FALSE, direction = "rightwards", adj = 0)
  plot(y=1:n, x=mvalue[1:n], type="b", xlim=c(-5,5), ylab="",
       xlab="Trait value", yaxt="n", pch=21)
  abline(v=0)
}

## Iteratively displays the simulated traits.
## Left-click on the display area to go to the next plot.
## To terminate: right-click (WIndows, X11), esc key (Mac), or hit the
## "finish" button (RStudio).

for(i in 1:10) {
  DisplayTreeEvol(tree2,y4[i,])
  if(is.null(locator(1)))
    break  ## Terminate:
}
```

Index