The `adephylo` package

Thibaut Jombart

December 16, 2008

Contents

1 Introduction

This document describes the `adephylo` package for the R software. `adephylo` aims at implementing exploratory methods for the analysis of phylogenetic comparative data, i.e. biological traits measured for taxa whose phylogeny is also provided. Procedures implemented in this package rely on exploratory data analysis. They include data visualization and manipulation, tests for phylogenetic autocorrelation, multivariate analysis, computation of phylogenetic proximities and distances, and modelling phylogenetic signal using orthonormal bases.

These methods can be used to visualize, test, remove or investigate the phylogenetic signal in comparative data. The purpose of this document is to provide a general view a the main functionalities of `adephylo`, and to show how this package can be used along with `ape`, `phylobase` and `ade4` to analyse comparative data.

2 First steps into `adephylo`

2.1 Data representation: why we are not reinventing the weel

Data representation can be defined as the way data are stored in a software (R, in our case). Technically, they are classes of objects containing the information. In the case of phylogeny, and comparative data, very efficient data representation are already defined in other packages. Hence, it made much more sense using directly objects from these classes.

Phylogenies are best represented in Emmanuel Paradis's `ape` package (`http://ape.mpl.ird.fr/`), as the class `phylo`. Note that as `ape` is by far the largest package dedicated to phylogeny, using the `phylo` class assures a good interoperability of data. This class is defined in an online document: `http://ape.mpl.ird.fr/misc/FormatTreeR_28July2008.pdf`.

However, data that are to be analyzed in `adephylo` do not only contain trees, but also traits associated to the tips of a tree. The package `phylobase` (`http://r-forge.r-project.org/projects/phylobase/`) is a collaborative effort designed to the handling of such data. Its representation of phylogenies is very similar to that of `ape`: the class `phylo4` basically is an extension of `phylo` class into formal (S4) class. More interestingly, the S4 class `phylo4d` can be used to store a tree and data associated to tips, internal nodes, or even edges of a tree. Classes of `phylobase` are described in a vignette of the package, accessible by typing:

```
> vignette("phylobase", package = "phylobase")
```

As trees and comparative data are already handled by `ape` and `phylobase`, no particular data format shall be defined in `adephylo`. In particular, we are no longer using `phylog` objects, which were used to represent phylogenies in `ade4`. This class is now deprecated, but all previous functionalities available for `phylog` objects have been re-implemented and – in some cases – improved in `adephylo`.

2.2 Installing the package

What is tricky here is that a vignette is basically available once the package is installed. Assuming you got this document before installing the package, here are some clues about installing adephylo.

First of all, adephylo depends on other packages, being methods, ape, phylobase, and ade4. These dependencies are mandatory, that is, you actually need to have these packages installed (with or without their dependencies) before using adephylo. Also, it is better to make sure you are using the latest versions of these packages. This can be achieved by typing update.packages, or (better for ade4 and phylobase) by installing devel versions from R-Forge (http://r-forge.r-project.org/). In all cases, the latest version of adephylo can be found from http://r-forge.r-project.org/R/?group_id=303.

When loading the package, dependencies are also loaded:

```
> library(adephylo)
```

Note that possibly conflicting, deprecated functions or datasets from ade4 are masked by adephylo. In case the converse would occur (i.e. deprecated function masking a function of adephylo), one can refer to the 'good' version of a function by adding the prefix adephylo:: to the function, without space. Hence, it is possibly to coerce the version of a masked function, using a kludge like:

```
> cat("\n=== Old - deprecated- version ===\n")

=== Old - deprecated- version ===

> orthogram <- ade4::orthogram
> args(orthogram)

function (x, orthobas = NULL, neig = NULL, phylog = NULL, nrepet = 999,
    posinega = 0, tol = 1e-07, na.action = c("fail", "mean"),
    cdot = 1.5, cfont.main = 1.5, lwd = 2, nclass, high.scores = 0,
    alter = c("greater", "less", "two-sided"))
NULL

> cat("\n=== New version === \n")

=== New version ===

> orthogram <- adephylo::orthogram
> args(orthogram)

function (x, tre = NULL, orthobas = NULL, prox = NULL, nrepet = 999,
    posinega = 0, tol = 1e-07, cdot = 1.5, cfont.main = 1.5,
    lwd = 2, nclass, high.scores = 0, alter = c("greater", "less",
        "two-sided"))
NULL
```

Luckily, this should not be required as long as one is not playing with loading and unloading ade4 once adephylo is loaded.

2.3 Getting started

All the material of the package is summarized in a manpage accessible by typing:

```
> `?`(adephylo)
```

Note that a html version may be preferred to browse easily the content of adephylo; this is accessible by typing:

```
> help("adephylo", package = "adephylo", html = TRUE)
```

To revert help back to text mode, simply type:

```
> options(htmlhelp = FALSE)
```

2.4 Putting data into shape

While this is not the purpose of this document to go through the details of phylo, phylo4 and phylo4d objects, we shall show briefly how these objects can be obtained.

2.4.1 Making a phylo object

The simplest way of turning a tree into a phylo object is using ape's function read.tree. This function reads a tree with the Newick (or 'parentetic') format, from a file (default, argument file) of from a character string (argument text).

```
> data(ungulates)
> ungulates$tre
```

```
[1] "((Antilocapra_americana,((Gorgon_taurinus,Oryx_leucoryx)W1,(Taurotragus_livingstoni,Tautragus_oryx)W2,(Gazel
```

```
> myTree <- read.tree(text = ungulates$tre)
> myTree
```

```
Phylogenetic tree with 18 tips and 13 internal nodes.

Tip labels:
        Antilocapra_americana, Gorgon_taurinus, Oryx_leucoryx, Taurotragus_livingstoni, Tautragus_oryx, Gazella_t
        Node labels:
        Root, W11, W10, W1, W2, W7,...

Rooted; no branch lengths.
```

```
> plot(myTree, main = "ape's plotting of a tree")
```

It is easy to convert ade4's phylog objects to a phylo, as phylog objects store the Newick format of the tree in the $tre component.

Note that phylo trees can also be constructed from alignements (see read.GenBankdist.dna, read.dna, dist.dna, nj, bionj, and mlphylo, all in ape), or even simulated (for instance, see rtree).

Also note that, if needed, conversion can be done back and forward with phylo4 trees:

```
> temp <- as(myTree, "phylo4")
> class(temp)
```

```
[1] "phylo4"
attr(,"package")
[1] "phylobase"
```

```
> temp <- as(temp, "phylo")
> class(temp)
```

```
[1] "phylo"
```

```
> all.equal(temp, myTree)
```

```
[1] TRUE
```

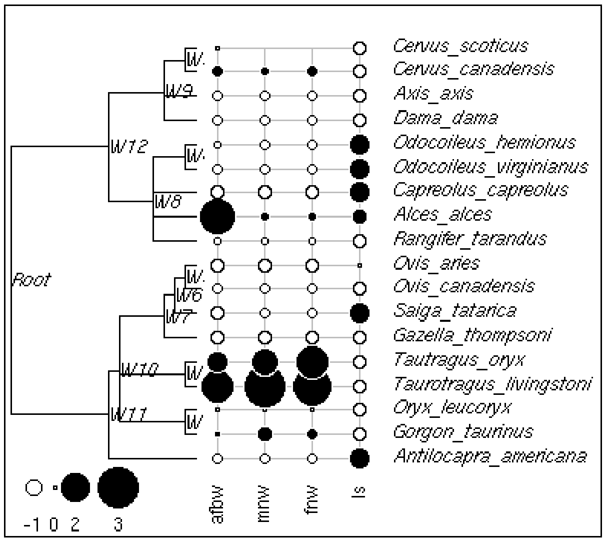
2.4.2 Making a `phylo4d` object

`phylo4d` objects are S4 objects, and are thus created in a particular way. The most immediate way of creating a `phylo4d` object is to call to the constructor, also named `phylo4d`. This is a function that takes two arguments: a tree (`phylo` or `phylo4` format) and a data.frame containing data, for tips by default (see `?phylo4d` for more information). Here is an example:

```
> ung <- phylo4d(myTree, ungulates$tab)
> class(ung)
```

```
[1] "phylo4d"
attr(,"package")
[1] "phylobase"
```

```
> table.phylo4d(ung)
```



Note that the constructor checks the consistency of the names used for the tips of the tree and for the rows of the data.frame. Inconsistencies issue an error. To override this behaviour, one can specify `use.tip.names=FALSE`. However, this can be tricky: often, mismatches between names can indicate that data are not sorted adequately; moreover, object created with such mismatches will often be invalid objects, and may issue errors in further analyses.

Data are stored inside the slot `@tip.data` of the object. They can be accessed either via this slot (in our example, `ung@tip.data`), or using the function `tdata`:

```
> x <- tdata(ung)
> head(x)
```

```
                          afbw   mnw   fnw  ls
Antilocapra_americana    50586  3832  3908 1.8
Gorgon_taurinus         165000 18600 14500 1.0
Oryx_leucoryx            87700  6840  6490 1.0
Taurotragus_livingstoni 405000 36300 28500 1.0
Tautragus_oryx          316000 26800 24700 1.0
Gazella_thompsoni        21300  2500  2500 1.0
```

3 Exploratory data analysis

3.1 Quantifying and testing phylogenetic signal

In this document, the terms 'phylogenetic signal' and 'phylogenetic autocorrelation' are used interchangeably. They refer to the fact that observations of traits are not independent in closely related taxa. Several procedures are implemented by `adephylo` to measure and test phylogenetic autocorrelation.

3.1.1 Moran's I

The function `moran.idx` computes Moran's I, the most widely-used autocorrelation measure. It can also provide additionnal information (argument `addInfo`), being the null value of I (i.e., the expected value in absence of phylogenetic autocorrelation), and the range of variation of I. It requires the degree of relatedness of tips on the phylogeny to be modelled by a matrix of phylogenetic proximities. Such a matrix can be obtained using different methods implemented by the function `proxTips`.

```
> W <- proxTips(myTree, met = "Abouheif")
> moran.idx(tdata(ung)$afbw, W)
```

```
[1] 0.1132682
```

```
> moran.idx(ung$tip.data[, 1], W, addInfo = TRUE)
```

```
[1] 0.1132682
attr(,"I0")
[1] -0.05882353
attr(,"Imin")
[1] -0.5217391
attr(,"Imax")
[1] 1.000699
```

From here, it is quite straightforward to build a non-parametric test based on Moran's I. For instance (taken from `?moran.idx`):

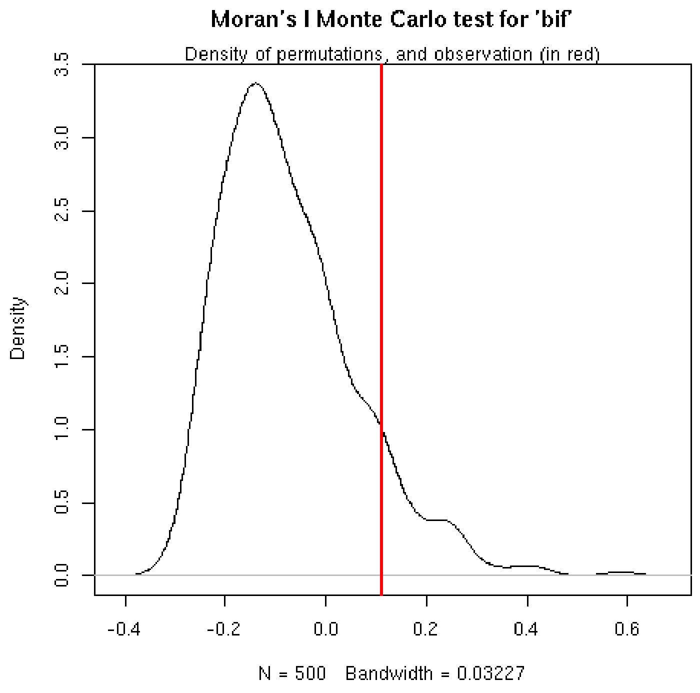
```
> afbw <- tdata(ung)$afbw
> sim <- replicate(499, moran.idx(sample(afbw), W))
> sim <- c(moran.idx(afbw, W), sim)
> cat("\n=== p-value (right-tail) === \n")
```

```
=== p-value (right-tail) ===
```

```
> pval <- mean(sim >= sim[1])
> pval
```

```
[1] 0.108
```

```
> plot(density(sim), main = "Moran's I Monte Carlo test for 'bif'")
> mtext("Density of permutations, and observation (in red)")
> abline(v = sim[1], col = "red", lwd = 3)
```



Here, `afbw` is likely not phylogenetically autocorrelated.

3.1.2 Abouheif's test

The test of Abouheif (see reference in `?abouheif.moran`) is designed to test the existence of phylogenetic signal. In fact, it has been shown that this test amounts to a Moran's I test with a particular proximity matrix (again, see references in the manpage). The implementation in `abouheif.moran` proposes different phylogenetic proximities, using by default the original one.

The function can be used on different objects; in particular, it can be used with a `phylo4d` object. In such case, all traits inside the object are tested. The returned object is a `krandtest`, a class of object defined by `ade4` to store multiple Monte Carlo tests. Here is an example using the ungulates dataset:

```
> ung.abTests <- abouheif.moran(ung)
> ung.abTests
```

```
class: krandtest
Monte-Carlo tests
Call: as.krandtest(sim = matrix(res$result, ncol = nvar, byr = TRUE), 
    obs = res$obs, alter = alter, names = test.names)

Test number:   4
```

```
Permutation number:   999
  Test       Obs  Std.Obs   Alter Pvalue
1 afbw 0.1653920 1.164201 greater  0.130
2  mnw 0.3681410 2.798732 greater  0.015
3  fnw 0.3843272 2.841451 greater  0.013
4   ls 0.3002425 2.125419 greater  0.035

other elements: NULL
```

```
> plot(ung.abTests)
```

In this case, it seems that all variables but `afbm` are phylogenetically structured.

Note that other proximities than those proposed in `abouheif.moran` can be used: on has just to pass the appropriate proximity matrix to the function (argument `W`). For instance, we would like to use the correlation corresponding to a Brownian motion as a measure of phylogenetic proximity.

First, we must estimate branch lengths, as the tree we possess does not have any:

```
> hasEdgeLength(ung)
```

```
[1] FALSE
```

```
> myTree.withBrLe <- compute.brlen(myTree)
```

Now, we can use ape's function `vcv.phylo` to compute the matrix of phylogenetic proximities, and use this matrix in Abouheif's test:

```
> myProx <- vcv.phylo(myTree.withBrLe)
> abouheif.moran(ung, W = myProx)
```

```
class: krandtest
Monte-Carlo tests
Call: as.krandtest(sim = matrix(res$result, ncol = nvar, byr = TRUE),
    obs = res$obs, alter = alter, names = test.names)

Test number:   4
Permutation number:   999
  Test        Obs    Std.Obs   Alter Pvalue
1 afbw 0.09173247 -0.2597878 greater  0.474
2  mnw 0.17740359  0.9165827 greater  0.169
3  fnw 0.17202965  0.8236364 greater  0.181
4   ls 0.15929851  0.5155151 greater  0.187

other elements: NULL
```

In the present case, traits no longer appear as phylogenetically autocorrelated. Several explanation can be proposed: the procedure for estimating branch length might not have been appropriate, or the Brownian motion may simply not be appropriate to describe the evolution of the traits under study for this set of taxa.

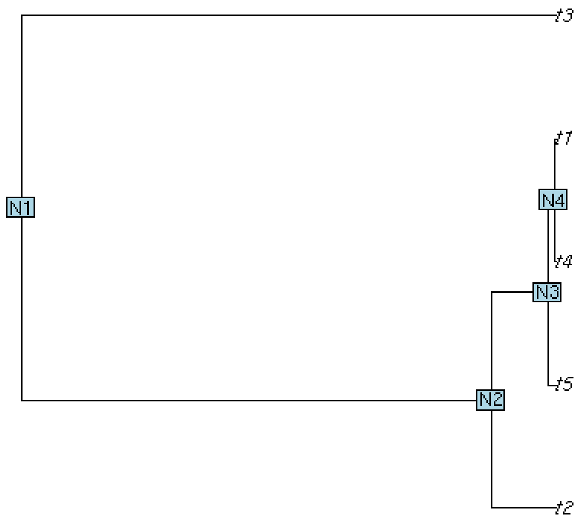
3.1.3 Phylogenetic decomposition of trait variation

The phylogenetic decomposition of the variation of a trait proposed by Ollier et al. (2005, see references in `?orthogram`) is implemented by the function `orthogram`. This function replaces the former, deprecated version from `ade4`.

The idea behind the method is to model different levels of variation on a phylogeny. Basically, these levels can be obtained from dummy vectors indicating which tip descends from a given node. A partition of tips can then be obtained for each node. This job is achieved by the function `treePart`. Here is an example using a small simulated tree:

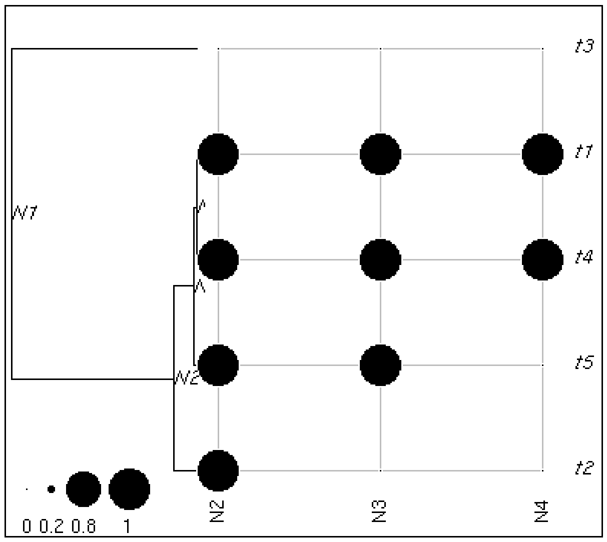
```
> x <- rcoal(5)
> plot(x, show.node = TRUE)
> nodelabels(paste("N", 1:4, sep = ""))
> treePart(x)
```

```
   N2 N3 N4
t3  1  0  0
t5  1  0  0
t2  0  1  0
t4  0  1  1
t1  0  1  1
```



The obtained partition can also be plotted:

```
> temp <- phylo4d(x, treePart(x))
> table.phylo4d(temp, cent = FALSE, scale = FALSE)
```



What we would like to do is assess where the variation of a trait is localized on the phylogeny; to do so, be could use these dummy vectors as regressors and see how variation is distributed among these vectors. However, these dummy vectors cannot be used as regressors because they are linearly dependent. The orthogram circumvents this issue by transforming and selecting dummy vectors into a new set of variables that are orthonormal. The obtained orthonormal basis can be used to decompose variation of the trait. Even if not necessary to get an orthogram, this basis can be obtained from `treePart`:

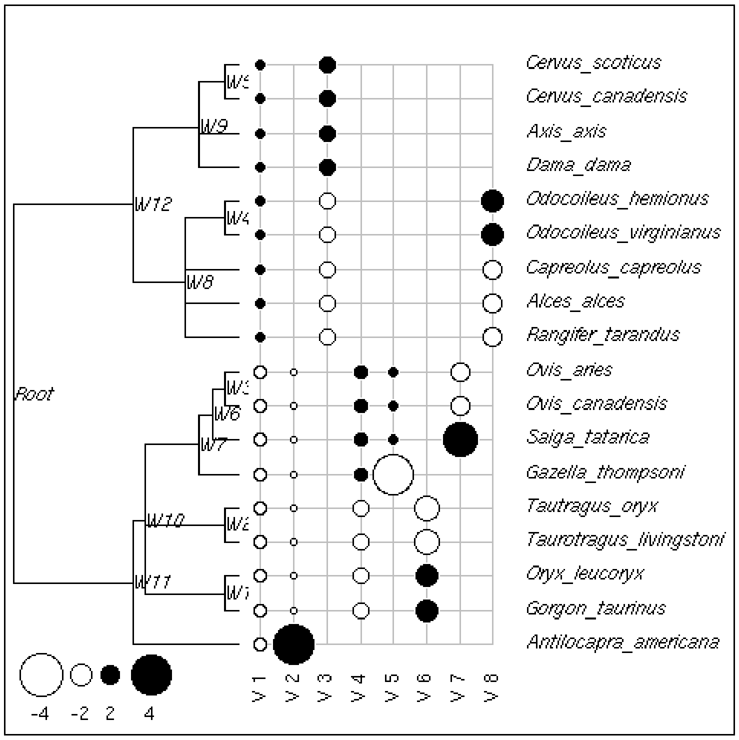
```
> args(treePart)
```

```
function (x, result = c("dummy", "orthobasis"))
NULL
```

```
> temp <- phylo4d(x, treePart(x, result = "orthobasis"))
> table.phylo4d(temp)
```

And here are the first 10 vectors of the orthonormal basis for the ungulate dataset:

```
> temp <- phylo4d(myTree, treePart(myTree, result = "orthobasis"))
> par(mar = rep(0.1, 4))
> table.phylo4d(temp, repVar = 1:8)
```



The decomposition of variance achieved by projecting a trait onto this orthonormal basis gives rise to several test statistics, that are performed by the function `orthogram`. Like the `abouheif.moran` function, `orthogram` outputs a `krandtest` object:

```
> afbw.ortgTest <- orthogram(afbw, myTree)
> afbw.ortgTest


class: krandtest
Monte-Carlo tests
Call: orthogram(x = afbw, tre = myTree)

Test number:   4
Permutation number:   999
   Test       Obs    Std.Obs   Alter Pvalue
1 R2Max 0.3298815  0.8158145 greater  0.172
2 SkR2k 8.2600870 -0.5098959 greater  0.701
3  Dmax 0.2066299  0.3084218 greater  0.336
4   SCE 0.1797097 -0.5809286 greater  0.686

other elements: NULL
```

Here again, `afbw` does not seem to be phylogenetically structured.

3.2 Modelling phylogenetic signal

3.2.1 Using orthonormal bases

In fact, the previous section describing the 'orthogram' has already shown that testing phylogenetic signal can (often) underlie modelling phylogenetic signal.

In the case of the orthogram, several tests are linked to the decomposition of the variance of a trait onto a particular basis describing tree topology. In fact, it is possible to extend the principle of the 'orthogram' to any orthonormal basis modelling phylogenetic topology. Another example of such bases is offered by Moran's eigenvectors, which can be used to model different observable phylogenetic structures (see references in `me.phylo`).

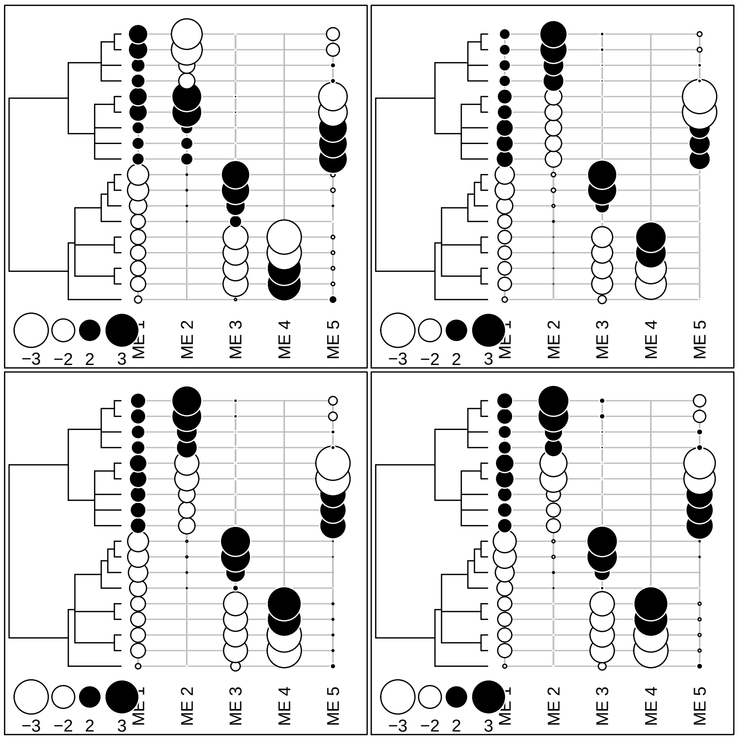
Moran's phylogenetic eigenvectors are implemented by the function `me.phylo` (also nicknamed `orthobasis.phylo`). The returned object is a data.frame with the class `orthobasis` defined in `ade4`; columns of this object are Moran's eigenvectors. An `orthobasis` be coerced to a regular data.frame or to a matrix using `as.data.frame` and `as.matrix`.

```
> me.phylo(myTree.withBrLe)
```

```
Orthonormal basis: data.frame with 18 rows and 17 columns
--------------------------------------
Columns are an orthonormal basis of 1n-orthogonal for
the inner product defined by the weights attribute
---------------------------------------
names = ME 1 ... ME 17
row.names = Antilocapra_americana ... Cervus_scoticus
weights = 0.05555556 ... 0.05555556
values = 0.5804085 ... -0.3981598
class = orthobasis data.frame
call =me.phylo(x = myTree.withBrLe)
```

Moran's eigenvectors are constructed from a matrix of phylogenetic proximities between tips. While any proximity can be used (argument `prox`), 5 proximities implemented by the `proxTips` function can be used, giving rise to different orthobases:

```
> ung.listBas <- list()
> ung.listBas[[1]] <- phylo4d(myTree, as.data.frame(me.phylo(myTree.withBrLe,
+     method = "patristic")))
> ung.listBas[[2]] <- phylo4d(myTree, as.data.frame(me.phylo(myTree,
+     method = "nNodes")))
> ung.listBas[[3]] <- phylo4d(myTree, as.data.frame(me.phylo(myTree,
+     method = "Abouheif")))
> ung.listBas[[4]] <- phylo4d(myTree, as.data.frame(me.phylo(myTree,
+     method = "sumDD")))
> par(mar = rep(0.1, 4), mfrow = c(2, 2))
> lapply(ung.listBas, table.phylo4d, repVar = 1:5, cex.sym = 0.7,
+     show.tip.label = FALSE, show.node = FALSE)
```



In this case, the first Moran's eigenvectors are all very similar. In other cases, the orthobases built from different proximities can be quite different.

3.3 Using multivariate analyses

3.4 Performing a phylogenetic Principal Component Analysis