A phylogenetic modelling tutorial using Phylogenetic Eigenvector Maps (PEM) as implemented in R package MPSEM (0.3-6).

Guillaume Guénard

June 3, 2019

Contents

1 Introduction

Phylogenetic Eigenvector Maps (PEM) is a method to perform phylogenetic modelling. Phylogenetic modelling consists in modelling trait evolution and predicting trait values using phylogeny as an explanatory factor (Guénard et al., 2013). Phylogenetic modelling allows one to predict trait values when it is difficult or impractical to obtain them, for instance when species are rare, extinct, or

when information is needed for several species and trait values are only available for a relatively small number of them (Guénard et al., 2011, 2014).

To apply phylogenetic modelling, one needs to have a set of species with known phylogeny and trait values (hereafter referred to as the "model species") as well as to know the locations, with respect to the phylogeny of the models species, of the species for which trait values are being predicted (hereafter referred to as the "target species"). Phylogenetic modelling can be performed conjointly with trait correlation modelling: it is possible to use other traits with known (or estimable) values for the target species to help predict a trait of interest. Phylogenetic trees being acyclic graphs, I will hereby prefer terms belonging to the graph theory over terms phylogeneticists may be more familiar with. Therefore I will use "edge" over "branches" and "vertex" over "root", "node" or "tip"; safe in cases where I want to be specific about what a vertex represents.

The PEM work flow consists in 1) calculating the influence matrix of the graph, 2) specifying a model of trait evolution along the edges of the phylogenetic tree, 3) calculating the left eigenvectors of the weighted and centred influence matrix and 4) use these eigenvectors as descriptors (Guénard et al., 2013). An `R` implementation of that approach is found in package `MPSEM`. `MPSEM` is meant to make the aforementioned process as seamless as possible. It is a work in progress; I welcome anyone to provide relevant suggestions and constructive remarks aimed at making `MPSEM` a better, more efficient and user-friendly, interface to phylogenetic modelling.

Assuming package `MPSEM` is installed, the first step to calculate a PEM is to load package `MPSEM`, which depends on packages `ape` and `MASS`:

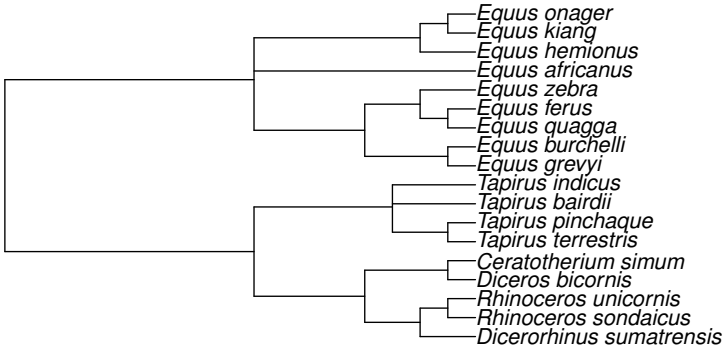
```
library(MPSEM)

## Loading required package:  ape
## Loading required package:  MASS
```

2 Preparing the data

For the present tutorial, we will use the data set `perissodactyla` from R package `caper`. These data from Purvis and Rambaut (1995) are loaded into your `R` workspace as follows:

```
data(perissodactyla,package="caper")
```

The `perissodactyla` data set contains `perissodactyla.tree`, a phylogenetic tree encompassing 18 odd-toed ungulate species:



as well as `perissodactyla.data`, a data frame containing trait information about the species. For the present study we will model the $\log_{10}$ gestation weight as a function of phylogeny and $\log_{10}$ adult female weight:

	Binomial	log.female.wt	log.neonatal.wt
1	Ceratotherium simum	6.26	4.90
2	Dicerorhinus sumatrensis	5.91	4.54
3	Diceros bicornis	6.18	4.70
4	Equus africanus	5.44	4.40
5	Equus burchelli	5.48	4.48
6	Equus grevyi	5.65	4.60
7	Equus hemionus	5.46	4.40
8	Equus zebra	5.46	4.40
9	Rhinoceros sondaicus	6.15	4.70
10	Rhinoceros unicornis	6.24	4.84
11	Tapirus indicus	5.46	3.92
12	Tapirus pinchaque	5.38	3.70
13	Tapirus terrestris	5.33	3.76

Before going any further, it is important to make sure that the species in the tree object are the same and presented in the same order as those in the data table. Glancing at the data table, species clearly cannot match since the latter feature information for only 13 of the 18 species in the tree. We will therefore match the tip labels of the original tree in the data table using the binary (Latin) species names in a character vector `spmatch`. When no matching element from the data table is found, an `NA` value appears at the corresponding position in `spmatch`. We can therefore use these `NA`s to reference the species

that can be dropped from the tree using `ape`'s function `drop.tip()` as follows:

```
spmatch <- match(perissodactyla.tree$tip.label,
                 perissodactyla.data[,1L])
perissodactyla.tree <- drop.tip(perissodactyla.tree,
                  perissodactyla.tree$tip.label[is.na(spmatch)])
```

Now that the data match the tree in content, one needs to verify whether they do so in order.

```
cbind(perissodactyla.tree$tip.label,perissodactyla.data[,1L])
```

```
##       [,1]                        [,2]
##  [1,] "Dicerorhinus sumatrensis" "Ceratotherium simum"
##  [2,] "Rhinoceros sondaicus"     "Dicerorhinus sumatrensis"
##  [3,] "Rhinoceros unicornis"     "Diceros bicornis"
##  [4,] "Diceros bicornis"         "Equus africanus"
##  [5,] "Ceratotherium simum"      "Equus burchelli"
##  [6,] "Tapirus terrestris"       "Equus grevyi"
##  [7,] "Tapirus pinchaque"        "Equus hemionus"
##  [8,] "Tapirus indicus"          "Equus zebra"
##  [9,] "Equus grevyi"             "Rhinoceros sondaicus"
## [10,] "Equus burchelli"          "Rhinoceros unicornis"
## [11,] "Equus zebra"              "Tapirus indicus"
## [12,] "Equus africanus"          "Tapirus pinchaque"
## [13,] "Equus hemionus"           "Tapirus terrestris"
```

Since they do not, we need to recalculate `spmatch` with the new, reduced, tree and re-order the data accordingly.

```
spmatch <- match(perissodactyla.tree$tip.label,
                 perissodactyla.data[,1L])
perissodactyla.data <- perissodactyla.data[spmatch,]
all(perissodactyla.tree$tip.label==perissodactyla.data[,1L])
```

```
## [1] TRUE
```

The last code line is just a last check to guarantee that all species names are matching. As a last step before we are done with data manipulation, I will put the binary names in place of the row names and delete the table's first row:

```
rownames(perissodactyla.data) <- perissodactyla.data[,1L]
perissodactyla.data <- perissodactyla.data[,-1L]
```

Our data of interest now appear as follows:

Finally, for the sake of demonstrating how to obtain predictions, we will remove the Sumatran rhinoceros (*Dicerorhinus sumatrensis*, the first species on

	log.female.wt	log.neonatal.wt
Dicerorhinus sumatrensis	5.91	4.54
Rhinoceros sondaicus	6.15	4.70
Rhinoceros unicornis	6.24	4.84
Diceros bicornis	6.18	4.70
Ceratotherium simum	6.26	4.90
Tapirus terrestris	5.33	3.76
Tapirus pinchaque	5.38	3.70
Tapirus indicus	5.46	3.92
Equus grevyi	5.65	4.60
Equus burchelli	5.48	4.48
Equus zebra	5.46	4.40
Equus africanus	5.44	4.40
Equus hemionus	5.46	4.40

top of the table) to obtain our training data set `perissodactyla.train`, keep the withdrawn data as `perissodactyla.test`, and calculate a tree without the target species:

```
perissodactyla.train <- perissodactyla.data[-1L,,drop=FALSE]
perissodactyla.test <- perissodactyla.data[1L,,drop=FALSE]
perissodactyla.tree.train <- drop.tip(perissodactyla.tree,
                              tip="Dicerorhinus sumatrensis")
```

3 Calculating PEM

3.1 Edge weighting function

As previously announced, I use the vocabulary of the graph theory when describing PEM: a tree is a (directed) graph, a branch is an edge, and the root, nodes, and tips are vertices. PEM allows one to specify a model of trait evolution along the edges of the tree. That model is given as a function having edge lengths as its first argument, followed by an arbitrary number of parameters provided as named arguments. Although PEM allows one to specify different parameter sets for different parts of the phylogeny as well as arbitrary weighting functions, the current implementation of `MPSEM` $(0.3-6)$ only supports the following power function:

$$w_{a,\psi}(\phi_j) = \begin{cases} \psi\phi^{\frac{1-a}{2}} & \phi_j > 0 \\ 0 & \phi_j = 0, \end{cases}$$

where a is the steepness parameter describing how abrupt the changes in trait values occur with time following branching, ψ is the evolution rate of the trait, and ϕ_j is the length of edge j (Guénard et al., 2013). As the steepness pa-

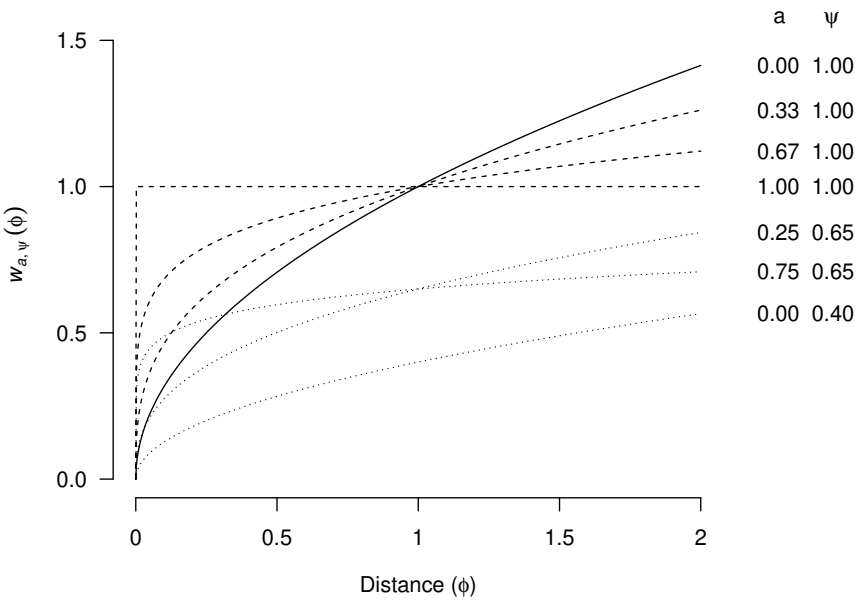


Figure 1: Values of the edge weighting function used as a model of trait evolution by `MPSEM` for different values of steepness (a) and evolution rate (ψ).

rameter increases, the weight assigned to a given edge increases more sharply with respect to the phylogenetic distance (or evolutionary time; Fig. 1). In the context of PEM, the edge weight represent the relative rate of evolution of the trait; the greater the edge weight, the greater the trait change along that edge. When $a = 0$, trait evolution is neutral and therefore proceeds by random walk along edges. When $a = 1$, edge weights no longer increase as a function of edge lengths. That situation corresponds to the scenario in which trait evolution is driven by the strongest possible natural selection: following a speciation event, trait either change abruptly (directional selection) at the vertex or do not change at all (stabilizing selection).

3.2 Phylogenetic graph

The first step to build a PEM is to convert the phylogenetic tree. The is done by giving the tree to function `Phylo2DirectedGraph()` as follows:

```
perissodactyla.pgraph <-
                Phylo2DirectedGraph(perissodactyla.tree.train)
```

Here's a snipet showing how `MPSEM`'s graph container stores graph information:

```
## List of 2
##  $ edge  :List of 3
##   ..$         : num [1:21] 13 14 15 16 16 15 17 17 14 18 ...
##   ..$         : num [1:21] 14 15 16 1 2 17 3 4 18 19 ...
##   ..$ distance: int [1:21] 9 4 3 1 1 3 1 1 5 2 ...
##  $ vertex:List of 1
##   ..$ species: logi [1:22] TRUE TRUE TRUE TRUE TRUE TRUE ...
##  - attr(*, "ev")= int [1:2] 21 22
##  - attr(*, "class")= chr "graph"
##  - attr(*, "elabel")= chr [1:21] "E1" "E2" "E3" "E4" ...
##  - attr(*, "vlabel")= chr [1:22] "Rhinoceros sondaicus" "Rhinoceros unicornis" "Diceros bicornis" "Ceratotherium simum" ...
```

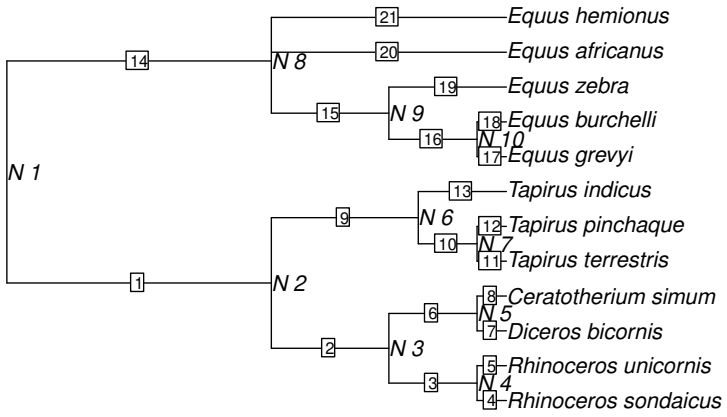
It is a list of two elements themselves being two lists. The element `$edge` is a list containing information about the graph's edges, namely the indices of their origin and destination vertices (the two first unnamed elements) and an arbitrary number of supplementary elements storing other edge properties. In the present case, a numeric vector created by `Phylo2DirectedGraph()` and called `$distance` stores the phylogenetic distances (ϕ_j), which correspond to the branch lengths of `perissodactyla.tree`. The element `$vertex` is a list containing an arbitrary number of elements storing vertex properties. In the present case, a logical vector created by `Phylo2DirectedGraph()` and called `$species` stores whether a given vertex represents a species (i.e., it is a tip). In addition to edge and vertex information, the container stores other useful information in the form of attributes: `ev` stores the number of edges and vertices whereas `elabel` and `vlabel` store edge and vertex labels, respectively.

3.3 Building the eigenvector map

In `MPSEM`, PEM are build using function `PEM.build()`. As an example, let us assume that the steepness and evolution rate are $a = 0.25$ and $\psi = 2$ among

genus *Equus*, $a = 0.8$ and $\psi = 0.5$ among genus *Tapirus*, and $a = 0$ and $\psi = 1$ from the root of the tree up to the vertex where the two latter genera begin as well as among the other genera. The following figure will help us figure out the indices of the edges involved:

```
tree <- perissodactyla.tree.train
tree$node.label <- paste("N",1L:tree$Nnode)
plot(tree,show.node.label=TRUE)
edgelabels(1L:nrow(tree$edge),
           edge=1L:nrow(tree$edge),bg="white",cex=0.75)
```




```
rm(tree)
```

Hence, $a = 0.25$ and $\psi = 2$ for edges $15 - 21$, $a = 0.8$ and $\psi = 0.5$ for edges $10 - 13$, and $a = 0$ and $\psi = 1$ for edges $1 - 9$ and 14:

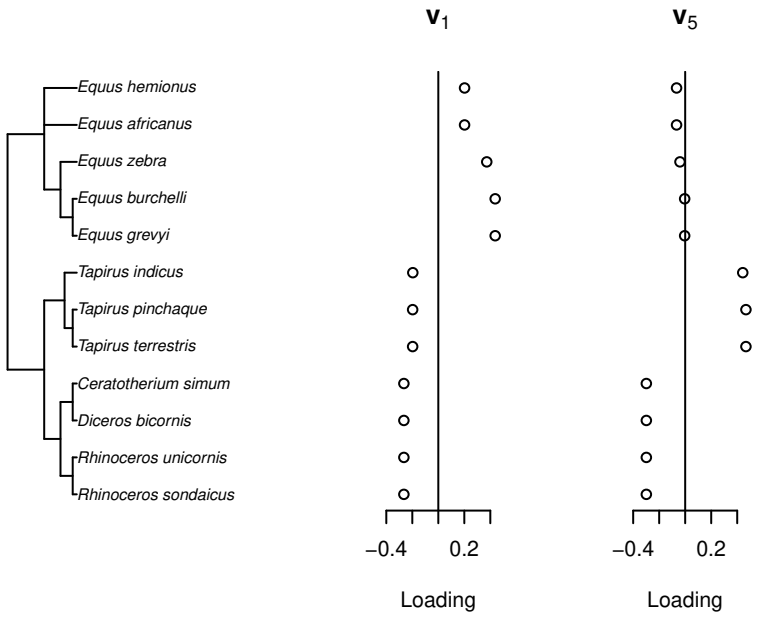
```
steepness <- rep(0,attr(perissodactyla.pgraph,"ev")[1L])
evol_rate <- rep(1,attr(perissodactyla.pgraph,"ev")[1L])
steepness[15L:21] <- 0.25
evol_rate[15L:21] <- 2
steepness[9L:13] <- 0.8
evol_rate[9L:13] <- 0.5
```

The PEM is obtained as follows:

```
perissodactyla.PEM <- PEM.build(perissodactyla.pgraph,
                                d="distance",sp="species",
                                a=steepness,psi=evol_rate)
```

In addition to the phylogenetic graph, function `PEM.build()` needs `d`, the name of the edge property where the phylogenetic distances are stored, `sp`, the name of the vertex property specifying what vertex is a species, as well as the user-specified steepness and evolution rate. When the vectors given to `a` or `psi`, have smaller sizes then the number of edges, values are recycled. The default values for `d` and `sp` are those produced by `Phylo2DirectedGraph()`, and can therefore be omitted in most cases. The object that `MPSEM`'s use to store PEM information is rather complex and we will hereby not browse through it. Method `as.data.frame` can be used to extract the eigenvector from a PEM. For a set of n species, that method returns a matrix encompassing $n-1$ column vectors that can be used in model to represent phylogenetic structure in traits. Here the phylogenetic patterns of variation described by two eigenvectors of the PEM we calculated above:

```
layout(matrix(c(1,1,1,2,2,3,3),1L,7L))
par(mar=c(5.1,2.1,4.1,2.1))
plot(perissodactyla.tree.train,x.lim=60,cex=0.75)
plot(y = 1L:nrow(perissodactyla.train), ylab="", xlab = "Loading",
     x = as.data.frame(perissodactyla.PEM)[,1L], xlim=0.5*c(-1,1),
     axes=FALSE, main = expression(bold(v)[1]))
axis(1) ; abline(v=0)
plot(y = 1L:nrow(perissodactyla.train), ylab="", xlab = "Loading",
     x = as.data.frame(perissodactyla.PEM)[,5L], xlim=0.5*c(-1,1),
     axes=FALSE, main = expression(bold(v)[5]))
axis(1) ; abline(v=0)
```



The pattern shown by the first eigenvector essentially contrasts Equids and the other odd-toed ungulate species whereas the pattern shown by the second eigenvector essentially contrasts tapirs and Rhinocerotids.

3.4 Estimate weighting parameters empirically

Because users do often not have information about the best set of weighting function parameters to use for modelling, `MPSEM` as has a function called `PEM.fitSimple()` that allows them to empirically estimate a single value of parameter a for the whole phylogeny[1] using restricted maximum likelihood[2]. That function requires a response variable that will be used to optimize the steepness parameter (here the $\log_{10}$ neonate weight) as well as lower and upper bounds for the admissible parameter values and is called as follows:

```
perissodactyla.PEM_opt1 <- PEM.fitSimple(
                      y = perissodactyla.train[,"log.neonatal.wt"],
                      x = NULL,
                      w = perissodactyla.pgraph,
                      d = "distance", sp="species",
                      lower = 0, upper = 1)
```

[1] Function `PEM.fitSimple()` does not estimate parameter ψ because the latter has no effect when its value is assumed to be constant throughout the phylogeny.

[2] A function to estimate different sets of weighting function parameters for different portions of the phylogeny has yet to be included in `MPSEM`.

If other traits are to be used in the model (here the $\log_{10}$ female weight), they are passed to the parameter `x` as follows:

```
perissodactyla.PEM_opt2 <- PEM.fitSimple(
                     y = perissodactyla.train[,"log.neonatal.wt"],
                     x = perissodactyla.train[,"log.female.wt"],
                     w = perissodactyla.pgraph,
                     d = "distance", sp="species",
                     lower = 0, upper = 1)
```

The results of the latter calls are PEMs similar to that obtained using `PEM.build()`, with additional information resulting from the optimization process. It is noteworthy that estimates of the steepness parameter (stored as element `$optim$par` of the PEM objects) and, consequently, the resulting phylogenetic eigenvectors, will be different depending on the use of auxiliary traits. In the example above, for instance, a was estimated to 0 by `PEM.fitSimple()` when no auxiliary trait is involved (first call) and to 0.08 when the female weight is used as an auxiliary trait (second call).

3.5 Phylogenetic modelling

To model trait values, PEM are used as descriptors in other modelling method. any suitable method can be used. For instance, package `MPSEM` contains a utility function called `lmforwardsequentialAICc()` that does step-wise variable addition in multiple regression analysis on the basis of the corrected Akaike Information Criterion (AICc; Hurvich and Tsai, 1993):

```
lm1 <- lmforwardsequentialAICc(
                     y = perissodactyla.train[,"log.neonatal.wt"],
                     object = perissodactyla.PEM_opt1)
summary(lm1)

##
## Call:
## lm(formula = as.formula(paste(p1, p2, sep = "")), data = df1)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.122057 -0.053781 -0.004971  0.055333  0.186364
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.39975    0.02740  160.58  < 2e-16 ***
## V_2         -1.31105    0.09491  -13.81  7.7e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.09491 on 10 degrees of freedom
## Multiple R-squared:  0.9502,Adjusted R-squared:  0.9452
## F-statistic: 190.8 on 1 and 10 DF,  p-value: 7.699e-08
```

```
lm2 <- lmforwardsequentialAICc(
            y = perissodactyla.train[,"log.neonatal.wt"],
            x = perissodactyla.train[,"log.female.wt",drop=FALSE],
            object = perissodactyla.PEM_opt2)
summary(lm2)
```

```
##
## Call:
## lm(formula = as.formula(paste(p1, p2, sep = "")), data = df1)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.021821 -0.014495 -0.004301  0.005790  0.040584
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.67127    0.13188 -20.256 9.41e-07 ***
## log.female.wt  1.23886    0.02307  53.698 2.80e-09 ***
## V_1            0.91697    0.02876  31.883 6.33e-08 ***
## V_10          -0.09891    0.02481  -3.987  0.00723 **
## V_8            0.08275    0.02483   3.333  0.01575 *
## V_3            0.07172    0.02499   2.870  0.02843 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0248 on 6 degrees of freedom
## Multiple R-squared:  0.998,Adjusted R-squared:  0.9963
## F-statistic:   587 on 5 and 6 DF,  p-value: 5.561e-08
```

Notice that to pass a single auxiliary trait to `lmforwardsequentialAICc()` in the current version of `MPSEM`, it is mandatory to set `drop=FALSE` to the bracket operator so that the variable name be conserved. Failure to do so will preclude one to make predictions using the resulting linear model. To obtain predictions, we need to calculate the locations of the target species with respect to the phylogeny of the model species. This is accomplished by `getGraphLocations()`, to which we give the tree for all species (model + targets) and the names (or indices) of the target species. Then, we use the `predict()` method for PEM objects. The latter takes, in addition to the PEM object, the locations of the target species as obtained by `getGraphLocations()`, an lm (or glm) object involving the eigenvectors of the PEM, and a table of auxiliary trait values for the target species, which can be omitted if no auxiliary trait is present in the

linear model.

```
perissodactyla.loc <- getGraphLocations(perissodactyla.tree,
                                targets="Dicerorhinus sumatrensis")
pred <- predict(object=perissodactyla.PEM_opt2,
                targets=perissodactyla.loc,
                lmobject=lm2,
                newdata=perissodactyla.test,
                "prediction",0.95)
```

Here, the predicted neonatal weight for the Sumatran rhinoceros is 26.7 kg and the bounds of the 95% prediction interval are 17.5 and 40.5 kg, while the observed value was actually 35 kg.

4 Cross-validating PEM predictions

Here, I will show you how to perform a leave-one-out cross-validation of a data set using the `R` code from the previous two sections. Predictions will be added to table `perissodactyla.data`:

```
perissodactyla.data <- data.frame(perissodactyla.data,
                          predictions = NA, lower = NA, upper = NA)
jackinfo <- list()
for(i in 1L:nrow(perissodactyla.data)) {
  jackinfo[[i]] <- list()
  jackinfo[[i]][["loc"]] <- getGraphLocations(perissodactyla.tree,
                         targets = rownames(perissodactyla.data)[i])

  jackinfo[[i]][["PEM"]] <- PEM.fitSimple(
                      y = perissodactyla.data[-i,"log.neonatal.wt"],
                      x = perissodactyla.data[-i,"log.female.wt"],
                      w = jackinfo[[i]][["loc"]]$x)
  jackinfo[[i]][["lm"]] <- lmforwardsequentialAICc(
            y = perissodactyla.data[-i,"log.neonatal.wt"],
            x = perissodactyla.data[-i,"log.female.wt",drop=FALSE],
            object = jackinfo[[i]][["PEM"]])
  predictions <- predict(object = jackinfo[[i]][["PEM"]],
      targets = jackinfo[[i]][["loc"]],
      lmobject = jackinfo[[i]][["lm"]],
      newdata = perissodactyla.data[i,"log.female.wt",drop=FALSE],
      "prediction",0.95)
  perissodactyla.data[i, c("predictions", "lower", "upper")] <-
                       unlist(predictions)
} ; rm(i, predictions)
```

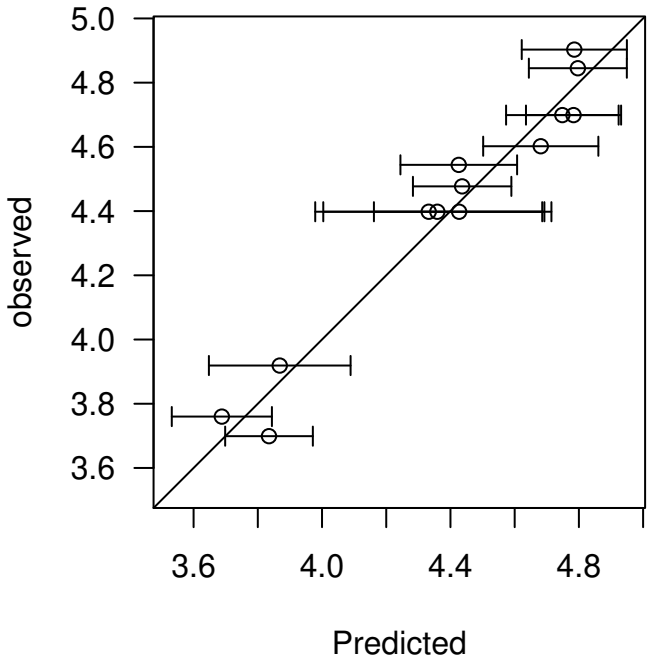


Figure 2: Predicted and observed $\log_{10}$ neonatal body mass for 13 odd-toed ungulate species.

```
## Warning in PEM.fitSimple(y = perissodactyla.data[-i, "log.neonatal.wt"],
:  No optimum found...  Message from optim() - ERROR: ABNORMAL_TERMINATION_IN_LNSRCH.
Status = 52
```

Because the result of `getGraphLocations()` includes the phylogenetic graph with the target species removed has its element `$x`, it is not necessary to re-calculate the tree with the target species dropped and the phylogenetic graph as we did previously for explanatory purposes. Also, I suggest storing the internal information about each cross-validation steps into a list (hereby called `jackinfo`), so it is possible to access the many details of the analyses later on. From the present cross-validation, we found that the (log) neonatal body mass can be predicted with a cross-validated R^2 of 0.96 (Figure 2).

5 Other utility functions

5.1 Influence matrix

The influence matrix is used internally to calculate PEM. It is a matrix having as many rows as the number of vertices (species + nodes) and as many columns as the number of edges. Any given element of the influence matrix is coding whether a vertex, which is represented a row of the matrix is influenced an edge, which is represented by a column of the matrix. In the context of PEM, a vertex is influenced by an edge when the former has ancestors on the latter or, in other words, when an edge is on the path leading from a tip to the root of the tree. The influence matrix is obtained as follows:

```
res <- PEMInfluence(perissodactyla.pgraph)
```

5.2 Update and forced PEM parameters

The calculation of the influence matrix performed by `PEM.build()` for a given phylogenetic graph need not be done every time new weighting function parameters are to be tried. For that reason, `MPSEM` provides a function called `PEM.updater()` that takes a previously calculated PEM object, applies new edge weighting, and recalculates the phylogenetic eigenvectors:

```
res <- PEM.updater(object = perissodactyla.PEM, a = 0, psi = 1)
```

The result of `PEM.build()` and `PEM.updater()` does not contain all the information necessary to predict trait values. Hence, neither of these functions is given information about the response variable and auxiliary traits. To perform these preliminary calculations, `MPSEM` provides the user with function `PEM.forcedSimple()` that produce the same output as `PEM.fitSimple()` with user-provided values of weighting parameters. It is called as follows:

```
res <- PEM.forcedSimple(
                    y = perissodactyla.train[,"log.neonatal.wt"],
                    x = perissodactyla.train[,"log.female.wt"],
                    w = perissodactyla.pgraph,
                    a = steepness, psi = evol_rate)
```

It is noteworthy that function `PEM.forcedSimple()` can actually apply different weighting parameters for different edges, in spite of what the adjective "Simple" in its name may suggest.

5.3 PEM scores

PEM scores are the values of target species on the eigenfunctions underlying the PEM. These scores are calculated from the graph locations and a PEM object using function `Locations2PEMscores()` as follows:

```
scores <- Locations2PEMscores(object = perissodactyla.PEM_opt2,
                              gsc = perissodactyla.loc)
```

The function is used internally by the `predict` method for PEM objects, and therefore need not be called when performing linear phylogenetic modelling as exemplified above. It comes in handy when the PEM is used together with other modelling approaches (e.g. multivariate regression trees, linear discriminant analysis, artificial neural networks) that have `predict` methods that are not specially adapted for phylogenetic modelling.

5.4 Miscellaneous

`MPSEM` comes with functions, some implemented in `C` language, to simulate quantitative traits evolution by Ornstein-Uhlenbeck process on potentially large phylogenies (Butler and King, 2004). These functions are only useful to perform simulations, which is a rather advanced matter outside the scope of the present tutorial. I refer the user to `MPSEM`'s help files for further details.

In addition to function `Phylo2DirectedGraph()`, which we have seen previously `MPSEM` also has built-in graph manipulation functions to populate a graph with vertices, add and remove vertices and edges, etc. These functions were mainly intended to be called internally by `MPSEM`'s functions. They were made visible upon loading the package because of their potential usefulness to some advanced applications that are outside the scope of the present tutorial. Again, I refer the user to `MPSEM`'s help files for further details.

References

Butler, M. A. and King, A. A. (2004). Phylogenetic comparative analysis: A modeling approach for adaptive evolution. *Am. Nat.*, 164:683–695.

Guénard, G., Legendre, P., and Peres-Neto, P. (2013). Phylogenetic eigenvector maps (PEM): a framework to model and predict species traits. *Meth. Ecol. Evol.*, 4:1120–1131.

Guénard, G., von der Ohe, P. C., de Zwart, D., Legendre, P., and Lek, S. (2011). Using phylogenetic information to predict species tolerances to toxic chemicals. *Ecol. Appl.*, 21:3178–3190.

Guénard, G., von der Ohe, P. C., Walker, S. C., Lek, S., and Legendre, P. (2014). Using phylogenetic information and chemical properties to predict species tolerances to pesticides. *Proc. R. Soc. B*, 281(20133239):20133239.

Hurvich, C. M. and Tsai, C.-L. (1993). A corrected Akaike information criterion for vector autoregressive model selection. *J. Time Ser. Anal.*, 14:271–279.

Purvis, A. and Rambaut, A. (1995). Comparative analysis by independent contrasts (CAIC): an Apple Macintosh application for analysing comparative data. *Comput. Appl. Biosci.*, 11:247–251.