RERconverge Analysis Walkthrough

February 28, 2024

Contents

This walkthrough provides instructions for implementing the RERconverge package to identify genes whose evolutionary rates shift in association with change in a trait. For information on how to download and install RERconverge, see the wiki. Source code and a quick start guide are available on github.

Overview

The following document describes the steps necessary to perform a standard RERconverge analysis to identify genomic elements with convergent **rates of evolution** in phenotypically convergent species using a **binary** trait or a **continuous** trait.

Output is a the list of genomic elements with statistics that represent the strength and direccon of the relationship between the genomic element's evolutionary rate and the phenotype. These statistics can be used to make inferences about the genomic elements' importances to the phenotype. Genomic elements that evolve more *slowly* for a given phenotype may be under *increased evolutionary constraint* because their function is important for the development of the convergent phenotype, for example. On the other hand, genomic elements that evolve more *quickly* for a given phenotype may either be under *decreased evolutionary constraint* due to loss of function or relatively decreased functional importance or, conversely, undergoing *directional selection* to increase or alter functionality. The ranked gene list can be further used as input for functional enrichment methodologies to find pathways or other functional groups under convergent evolutionary pressures.

Data Input Requirements and Formatting

The analysis requires two sources of data as input:

1. Phylogenetic trees for every genomic element with branch lengths that represent element-specific evolutionary rates
   - Trees should be in Newick format with tip labels and no node labels
   - Tree topologies must all be subsets of the same master tree topology
2. Species-labeled phenotype values
   - Species labels must match tree tip labels
   - For continuous traits:
     - a named numeric vector of trait values
   - For binary traits:
     - a vector of foreground species names OR
     - a Newick tree with branch lengths 0 for background branches and values between 0 and 1 for foreground branches (for a basic analysis, set all foreground branches to 1) OR
     - the user can specify foreground branches (which will be set to 1) using an interactive tool (see below)

We now provide tools for users to estimate approximate maximum likelihood trees from nucleotide or amino acid alignments using the `pml` and `optim.pml` functions from the phangorn package (Schliep 2011).

When choosing a dataset to work with, consider the availability and accuracy of both genomic and phenotypic data, and be sure to select a valid convergent phenotype that is observed in multiple independent clades in your phylogeny, and at high and low levels for continuous traits.

For a more detailed description of data formatting requirements and examples, please see the relevant sections of the walkthrough.

Analysis Walkthrough

Installing and loading RERconverge

Note: Prior to running the vignette, be sure to follow all the steps for installation on the wiki, up to the "Install from Github" step.

In R, load the RERConverge library.

```
if (!require("RERconverge", character.only=T, quietly=T)) {
    require(devtools)
    install_github("nclark-lab/RERconverge", ref="master")
    #"ref" can be modified to specify a particular branch
}
library(RERconverge)
```

This should also download all the files we will be working with to your computer, in the directory where your R library lives. If you'd like to visualize or work with any of these files separately, this is where you can find them:

```
rerpath = find.package('RERconverge') #If this errors, there is an issue with installation
print(rerpath)
```

```
## [1] "/Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RERconverge"
```

Reading in gene trees with `readTrees`

To run RERconverge, you will first need to supply a file containing **gene trees** for all genes to be included in your analysis. This is a tab delimited file with the following information on each line:

Gene_name Newick_tree

An example file is provided in *inst/extdata/subsetMammalGeneTrees.txt*, which you can view in any text editor.

Now in R, read in the gene trees. The `readTrees` function takes quite a while to read in trees for all genes, so we will limit ourselves to the first 200 using `max.read` (this will still take a minute or so, so be patient):

```
toytreefile = "subsetMammalGeneTrees.txt"
toyTrees=readTrees(paste(rerpath,"/extdata/",toytreefile,sep=""), max.read = 200)
```

```
## Read 500 items
```

```
## max is 62
```

```
## Rotating trees
```

```
## estimating master tree branch lengths from 32 genes
```

```
## Naming columns of paths matrix
```

First, the code tells us that there are 500 items, or gene trees, in the file. Since we have set `max.read = 200`, it will only read the first 200 of these. Then it says that the maximum number of tips in the gene trees is 62 and, later, it reports that it will use the 32 genes in this set that have data for all 62 species to estimate a **master tree**. The master tree will be used for subsequent analyses.

RERconverge is intended to be used on genome-scale datasets, containing a large number of gene trees with data present for all species. It thus has a minimum number of such gene trees required for `readTrees` to use to estimate a **master tree**; this is set with the `minTreesAll` option and is 20 by default. If your dataset is smaller, you may adjust this or supply your own master tree using the option `masterTree` (this should be a `phylo` object generated using ape's `read.tree`); however, we recommend interpreting results with caution in this case.

Estimating relative evolutionary rates (RER) with `getAllResiduals`

The next step is to estimate **relative evolutionary rates**, or RERs, for all branches in the tree for each gene. Intuitively, a gene's RER for a given branch represents how quickly or slowly the gene is evolving on that branch relative to its overall rate of evolution throughout the tree.

Briefly, RERs are calculated by normalizing branch lengths across all trees by the master branch lengths. Branch lengths are then corrected for the heteroskedastic relationship between average branch length and variance using weighted regression. For a more detailed description of how RERs are computed, see (Chikina, Robinson, and Clark 2016) and (Partha et al. 2017).

We will use the `getAllResiduals` function to calculate RERs. This uses the following input variables (all the options set here are also the defaults):

- `useSpecies`: a vector that can be used to specify a subset of species to use in the analysis. Here we will use the species in our AdultWeightLog vector that will be used for continuous trait analysis. Note that these are also the same species used for binary trait analysis. These species should be a subset of species included in `toyTrees$masterTree$tip.label`.
- `transform`: the method used to transform the raw data. By transforming the raw data, we reduce the heteroscedasticity (relationship between mean and variance) and the influence of outliers. Here we will use a square-root transform ("sqrt"), which has performed the best at reducing heteroskedasticity in our datasets. Also available are "none" (no transformation) and "log" (natural logarithm transformation).
- `weighted`: whether to use a weighted regression to estimate RER. Weighting allows further correction for the relationship between mean and variance, which can be directly estimated from the data.
- `scale`: whether to scale the individual branches of the gene trees to account for variance across trees. This scales the variance, though not the mean, of each branch length, using the R function `scale`.

The `useSpecies` input variable can be provided to most RERconverge functions. Excluding one or more species from this vector will exclude them from the analyses.

Here is the basic method, with the recommended settings:

```
data("logAdultWeightcm")
mamRERw = getAllResiduals(toyTrees,useSpecies=names(logAdultWeightcm),
                          transform = "sqrt", weighted = T, scale = T)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```
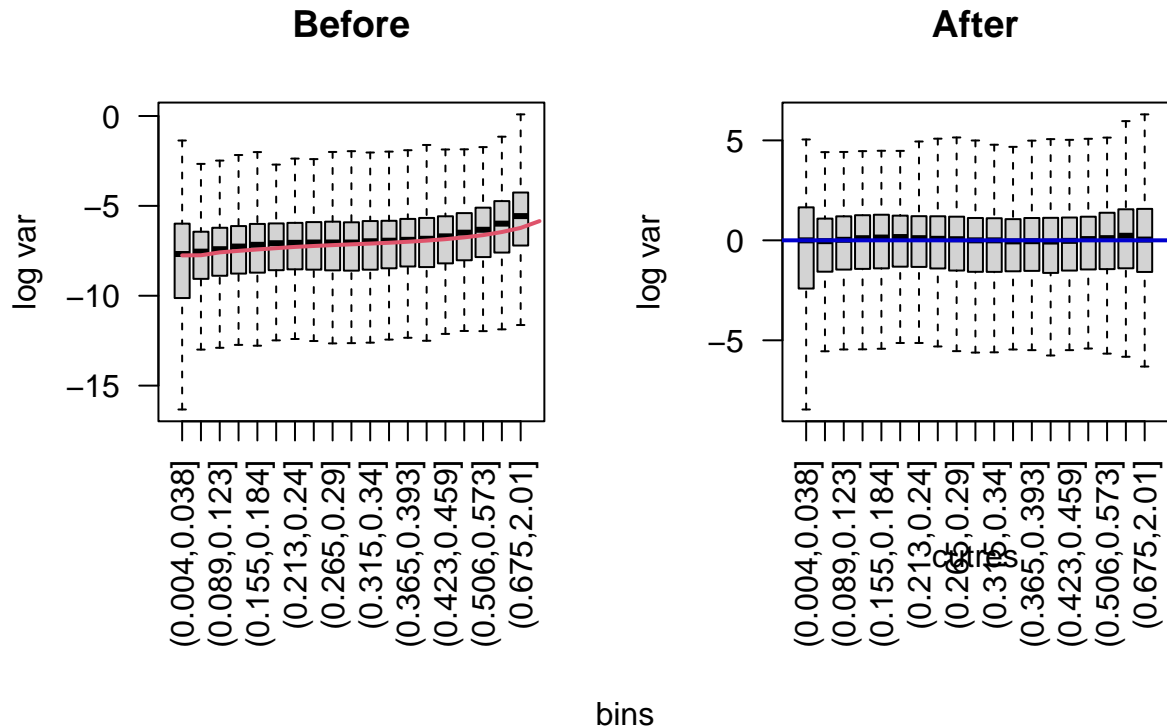

The first output of this function tells you that the cutoff is set to 3.2e-05. Any branches shorter than this will

be excluded from the analysis. It then prints out i= 1 ... 200, showing the progress as it calculates relative evolutionary rates for sets of gene trees.

The plots generated by this function show the log variance of the RERs resulting from the original method (on the left) and the variance after transformation and weighted regression (on the right). Notice the heteroscedasticity (positive trend between values and their variance) that is present before the new transformation method is applied, is now gone. The x-axis displays bins of branch lengths on the tree, and the y-axis is the (log-scaled) variance in these branch lengths across trees. As you can see by comparing the right plot to the left plot, transforming and performing a weighted regression reduces the relationship between the mean branch length (x-axis) and the variance in branch length (y-axis). You can alter values for `transform`, `scale`, and `weighted` to attempt to optimize heteroskedasticity correction.

If you wish to save this RER object for later, you can use R's `saveRDS` function. This will allow you to load it later with `readRDS`, using a different name, if you wish.

```
saveRDS(mamRERw, file="mamRERw.rds")
newmamRERw = readRDS("mamRERw.rds")
```

Now that we have RERs, we can visualize these for any given gene using the `plotRers` function. Here is an example.

```
#make average and gene tree plots
noneutherians <- c("Platypus","Wallaby","Tasmanian_devil","Opossum")
par(mfrow=c(1,2))
avgtree=plotTreeHighlightBranches(toyTrees$masterTree, outgroup=noneutherians,
                                  hlspecies=c("Vole","Squirrel"), hlcols=c("blue","red"),
                                  main="Average tree") #plot average tree
bend3tree=plotTreeHighlightBranches(toyTrees$trees$BEND3, outgroup=noneutherians,
                                    hlspecies=c("Vole","Squirrel"), hlcols=c("blue","red"),
                                    main="BEND3 tree") #plot individual gene tree
```
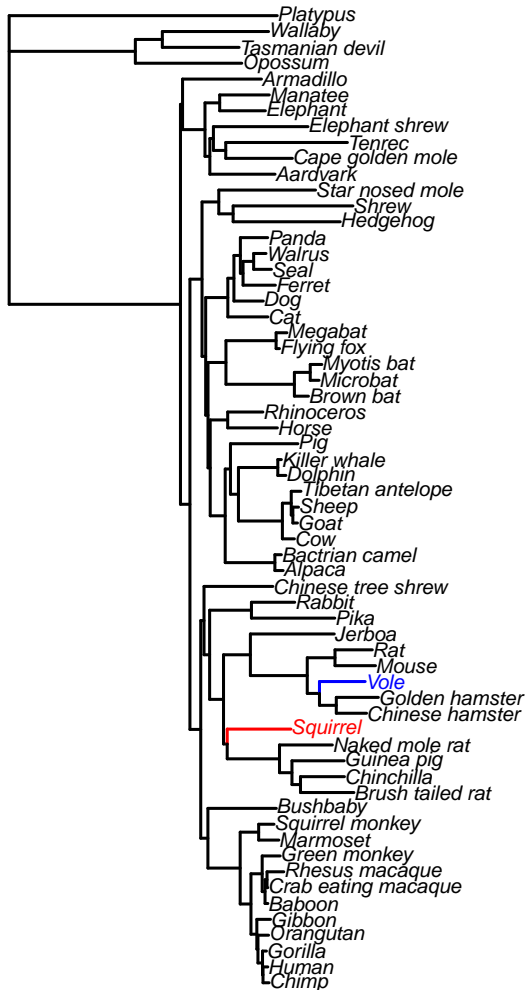
Average tree　　　　　　　　　　　　　　**BEND3 tree**
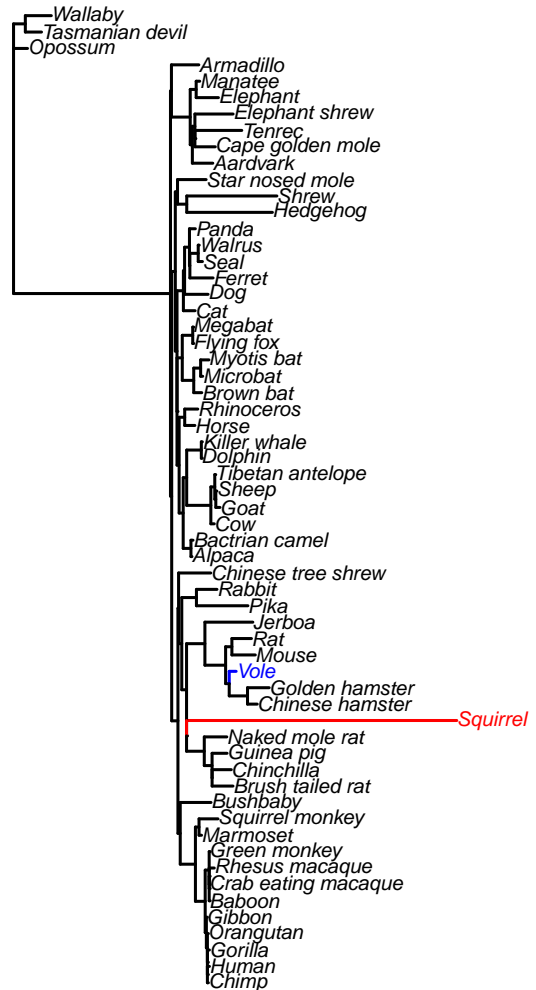
The left plot is a tree with branch lengths representing the average rates across all genes. The right plot is the same tree, but with branch lengths representing rates specifically for the BEND3 gene.

```
#plot RERs
par(mfrow=c(1,1))
phenvExample <- foreground2Paths(c("Vole","Squirrel"),toyTrees,clade="terminal")
plotRers(mamRERw,"BEND3",phenv=phenvExample) #plot RERs
```

BEND3: rho = 0.0021, p = 0.9802
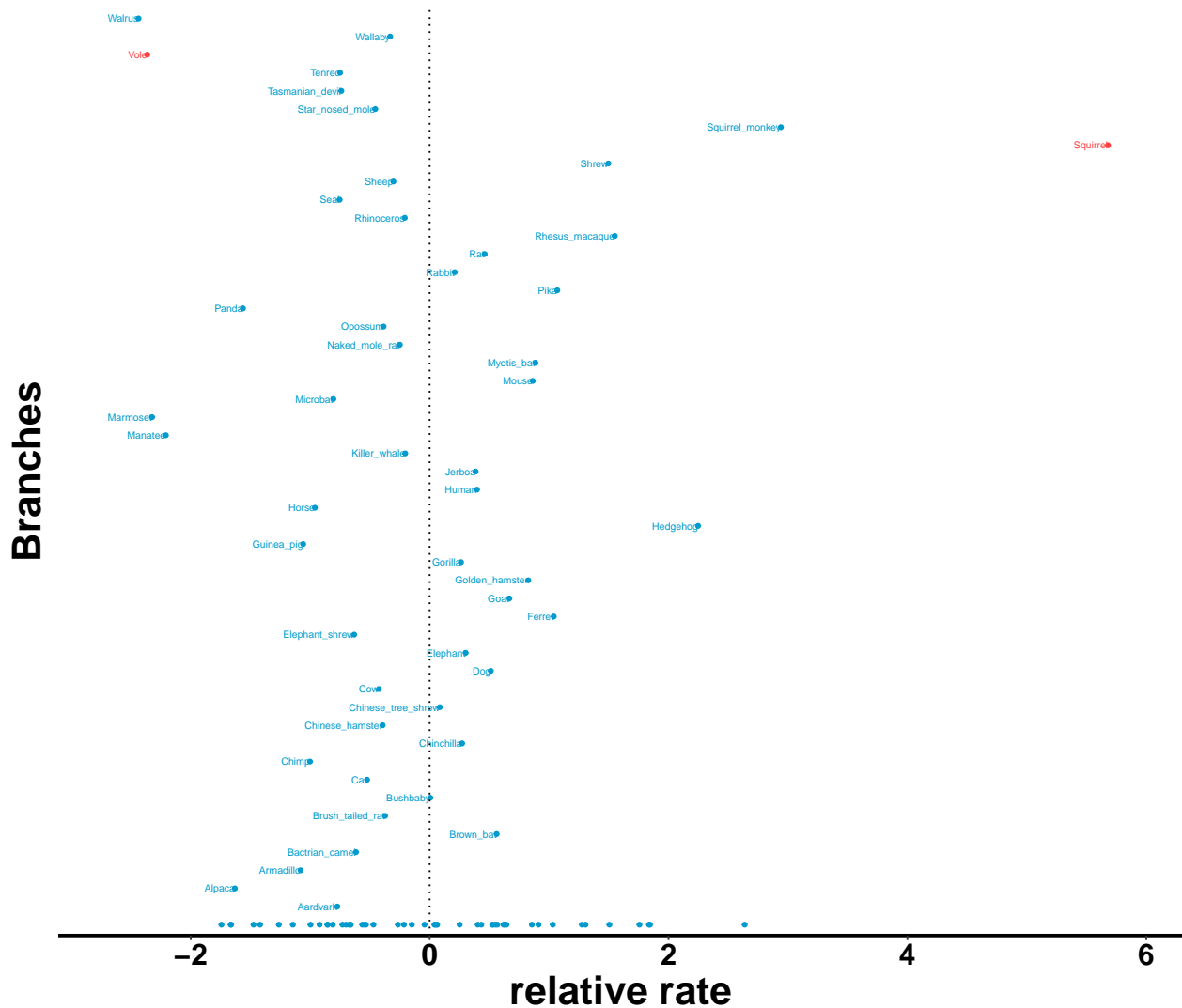
This plot represents the estimated RERs for terminal branches (labeled with species names along the y-axis) and internal branches (plotted in a single row at the base of the y-axis). The foreground branches (set here using `foreground2Paths`) are highlighted in red; these are currently only terminal branches, but if any internal branches were included, there would be a red point or points among the RER points at the base of the y-axis. Notice how the RER for vole is negative; this is because the branch leading to vole in the BEND3 tree is shorter than average. On the other hand, the RER for squirrel is positive because the branch leading to squirrel in the BEND3 tree is longer than average.

We can also save all RERs for a given gene using the `returnRersAsTree` function. If we include `plot=TRUE`, this function will also generate a cladogram for the gene tree with branches labeled with their RERs.

```
#plot RERs as tree
par(mfrow=c(1,1))
bend3rers = returnRersAsTree(toyTrees, mamRERw, "BEND3", plot = TRUE,
                            phenv=phenvExample) #plot RERs
```
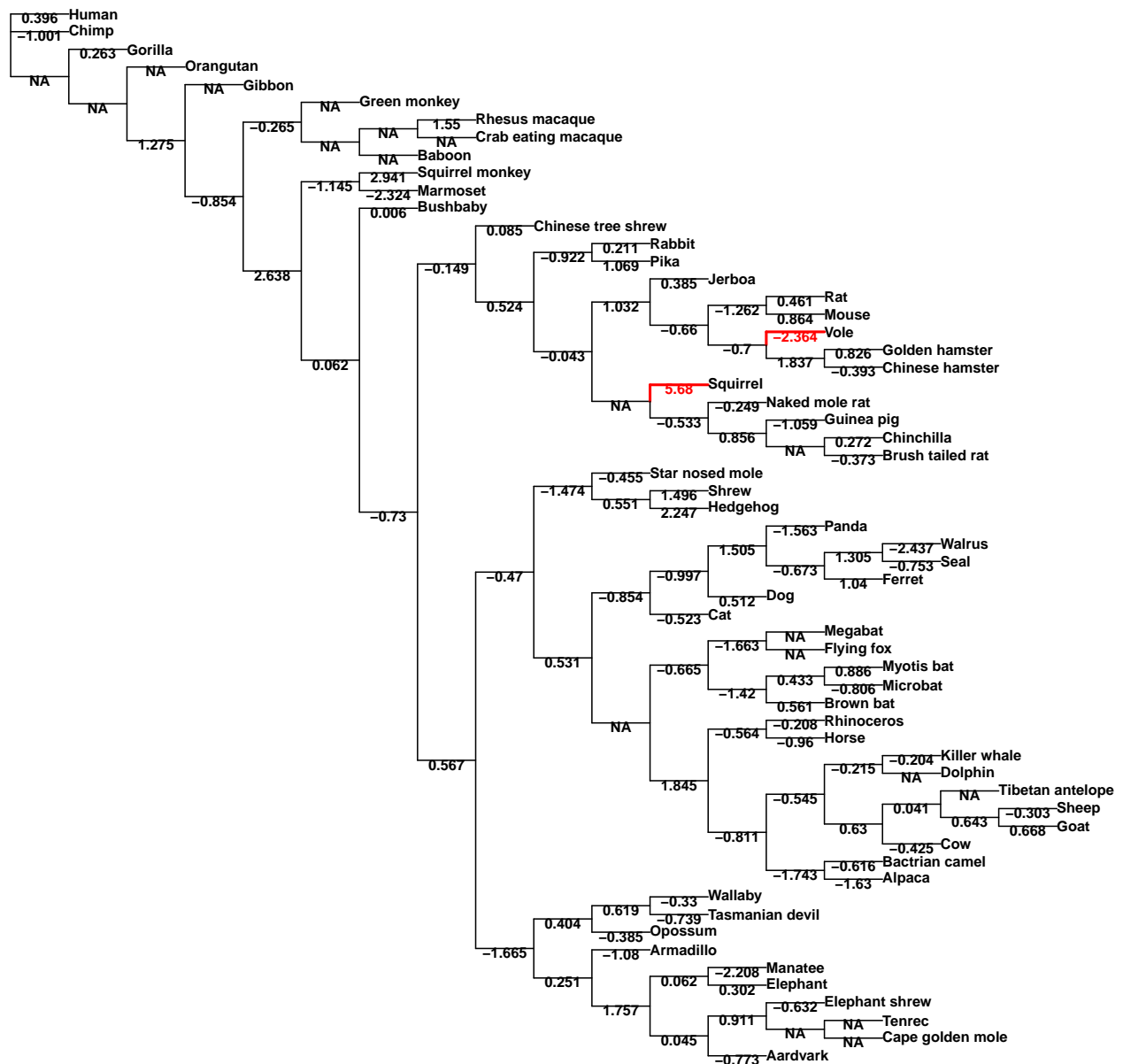
Here we can see by inspecting the branch labels that the RER for the terminal branch leading to squirrel is a large positive value (5.68), and the RER for the terminal branch leading to vole is a large negative value (-2.364).

This function also returns a tree whose branch lengths represent the RERs for this gene. We can print or save the tree using the *ape* package function `write.tree`:

```
strwrap(gsub(":",write.tree(bend3rers),replacement=": "))
```

```
##  [1] "(((((((((((((((Aardvark: -0.772901163,((Cape_golden_mole: NA,Tenrec: NA):"
##  [2] "NA,Elephant_shrew: -0.6319150881): 0.9107871023):"
##  [3] "0.04450767607,(Elephant: 0.3020873549,Manatee: -2.208494036):"
##  [4] "0.06228605514): 1.75697314,Armadillo: -1.080159105):"
##  [5] "0.2513799251,(Opossum: -0.3853210769,(Tasmanian_devil:"
##  [6] "-0.7390694418,Wallaby: -0.3302797099): 0.618648797): 0.403637805):"
##  [7] "-1.665131653,((((((Alpaca: -1.630164966,Bactrian_camel: -0.616372635):"
##  [8] "-1.74284876,((Cow: -0.4248438251,((Goat: 0.6680167833,Sheep:"
```

```
##  [9] "-0.303306151): 0.6433736448,Tibetan_antelope: NA): 0.0412454544):"
## [10] "0.6304769683,(Dolphin: NA,Killer_whale: -0.2038751706): -0.2149383164):"
## [11] "-0.5446577331): -0.8108364886,(Horse: -0.9603730297,Rhinoceros:"
## [12] "-0.2079734468): -0.5638444601): 1.844886075,((Brown_bat:"
## [13] "0.5614759796,(Microbat: -0.8061174542,Myotis_bat: 0.8859062157):"
## [14] "0.4331358477): -1.419731873,(Flying_fox: NA,Megabat: NA): -1.66280716):"
## [15] "-0.6652905499): NA,(Cat: -0.5234598585,(Dog: 0.512457132,((Ferret:"
## [16] "1.039715189,(Seal: -0.7529535484,Walrus: -2.437232254): 1.305136139):"
## [17] "-0.6733796329,Panda: -1.562778374): 1.505246607): -0.9968289192):"
## [18] "-0.8543925179): 0.5309002963,((Hedgehog: 2.247416284,Shrew:"
## [19] "1.49636385): 0.5513410295,Star_nosed_mole: -0.454803983):"
## [20] "-1.474498251): -0.4700958206): 0.5672648802,(((((((Brush_tailed_rat:"
## [21] "-0.3729530647,Chinchilla: 0.2720817699): NA,Guinea_pig: -1.05860932):"
## [22] "0.8561136369,Naked_mole_rat: -0.2494834422): -0.5328674185,Squirrel:"
## [23] "5.68003212): NA,((((Chinese_hamster: -0.3933090818,Golden_hamster:"
## [24] "0.8262536632): 1.836711178,Vole: -2.363819782): -0.7003967847,(Mouse:"
## [25] "0.8635871912,Rat: 0.4613863601): -1.262274778): -0.6602931093,Jerboa:"
## [26] "0.3854019665): 1.031588863): -0.04257838007,(Pika: 1.068916501,Rabbit:"
## [27] "0.2105874859): -0.9215070493): 0.5241627888,Chinese_tree_shrew:"
## [28] "0.08536326449): -0.1489048028): -0.7295714186,Bushbaby:"
## [29] "0.005684441225): 0.06150118138,(Marmoset: -2.323552425,Squirrel_monkey:"
## [30] "2.940934638): -1.144954008): 2.638013047,((Baboon:"
## [31] "NA,(Crab_eating_macaque: NA,Rhesus_macaque: 1.550447589): NA):"
## [32] "NA,Green_monkey: NA): -0.2650410725): -0.8544241823,Gibbon: NA):"
## [33] "1.275102831,Orangutan: NA): NA,Gorilla: 0.2626488364): NA,Chimp:"
## [34] "-1.00135553,Human: 0.3963351921);"
write.tree(bend3rers, file='BEND3RER.nwk')
```

The function `returnRersAsTreesAll` will produce an object of class "multiPhylo", with each element containing a named gene tree with branch lengths representing RERs for that gene. This can then be used for further analysis or customized plotting, and it can be saved to a file using `write.tree`.

```
multirers = returnRersAsTreesAll(toyTrees,mamRERw)
write.tree(multirers, file='toyRERs.nwk', tree.names=TRUE)
```

Another useful function for visualizing RERs for a given tree is `treePlotRers`. When called with `type = label`, it will produce the same plot as shown above, i.e., a cladogram with the branches labeled by RER. Alternatively, when called with `type = color`, it will produce a cladogram with RERs displayed as a color heatmap on the branches.

```
#visualize RERs along branches as a heatmap
newbend3rers = treePlotRers(treesObj=toyTrees, rermat=mamRERw, index="BEND3",
                            type="c", nlevels=9, figwid=10)
```
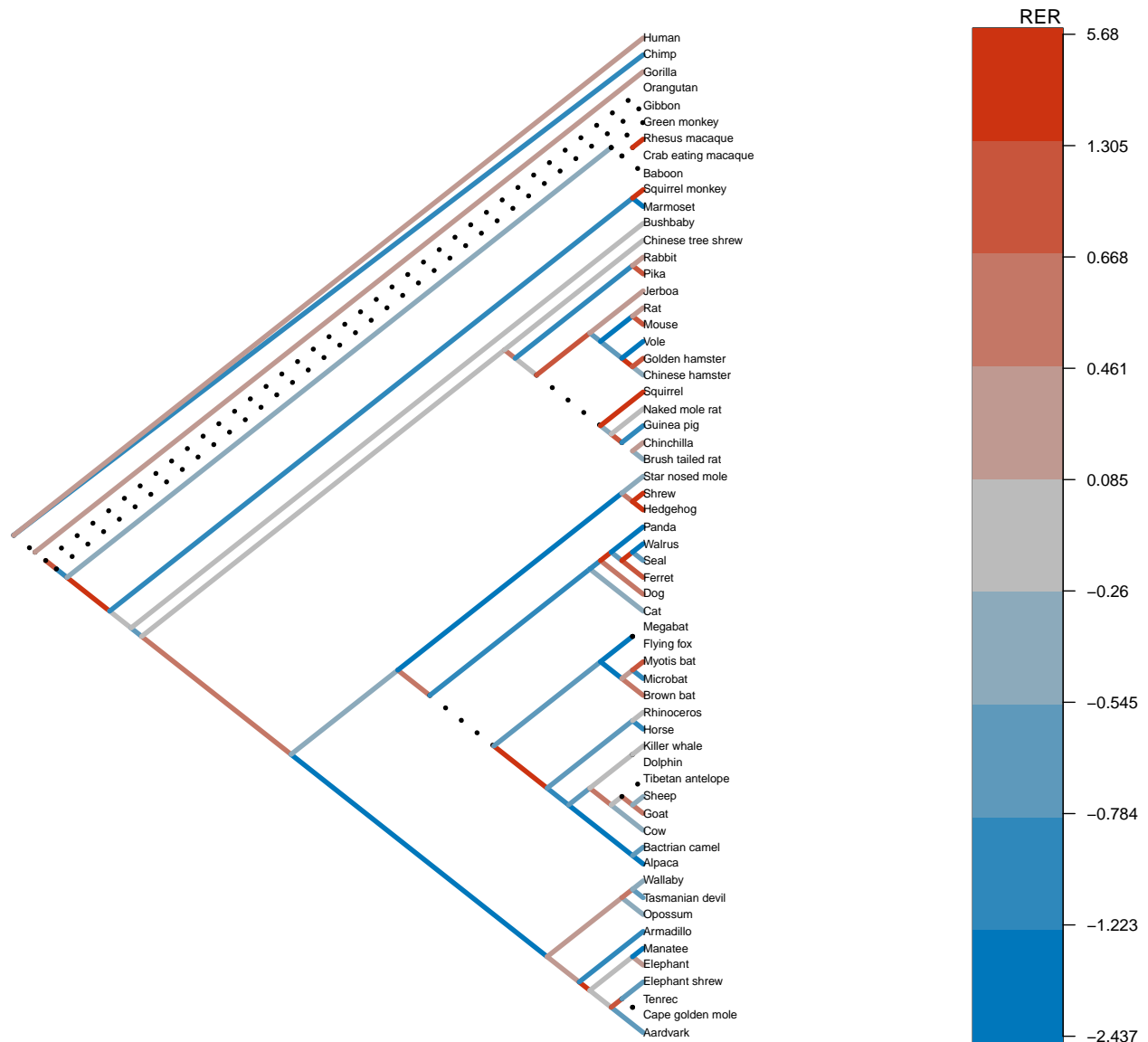
The heatmap has 9 colors, corresponding to `nlevels=9`. The breaks are determined by quantiles of the RER distribution, with separate quantiles computed for values below and above zero. Dashed lines indicate lineages that have been excluded (edge.length = NA). When using a different device, you may need to adjust the `figwid` variable to display the full phylogeny with optimal aesthetics. For more information on options that can be passed to this function, see the documentation for `treePlotNew`.

Binary Trait Analysis

Now we will associate variation in these RERs with variation in a **binary trait** across the tree. To do so, we first need to provide information about which branches of the tree have the trait of interest (**foreground branches**). There are several possible ways to do this:

1) Provide a binary trait tree file. This should be a file in Newick format with branch lengths zero for background branches and one (or weights between zero and one) for foreground branches. Setting any branches to NA will exclude these branches from the analysis. An example binary tree file is provided in *inst/extdata/MarineTreeBinCommonNames.txt*. This tree must have the same topology as the master tree or a subset of the master tree.

```
marineb=read.tree(paste(rerpath,"/extdata/MarineTreeBinCommonNames_noCGM.txt",sep=""))
marinebrooted = root(marineb,outgroup=noneutherians)
par(mfrow=c(1,2))
plot(marinebrooted, main="Trait tree from file (1)")
#alternative way of representing the tree
mb1 = marineb
mb1$edge.length = c(rep(1,length(mb1$edge.length)))
binplot1=plotTreeHighlightBranches(mb1, outgroup=noneutherians,
                                   hlspecies=which(marineb$edge.length==1), hlcols="blue",
                                   main="Foreground branches highlighted (1)")
```

Trait tree from file (1) **Foreground branches highlighted (1)**
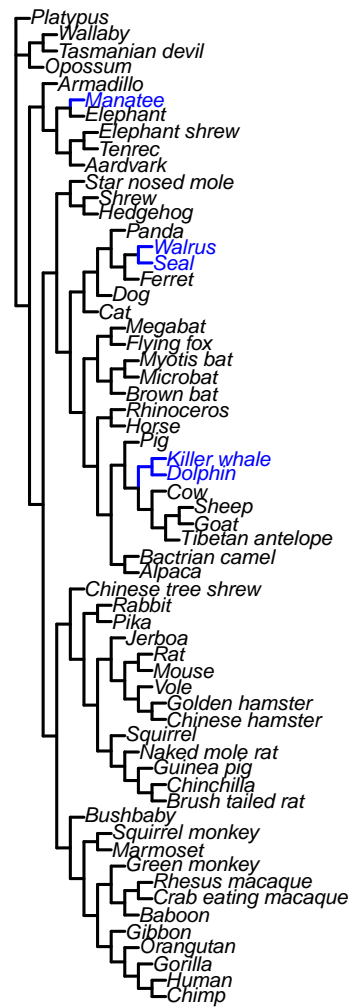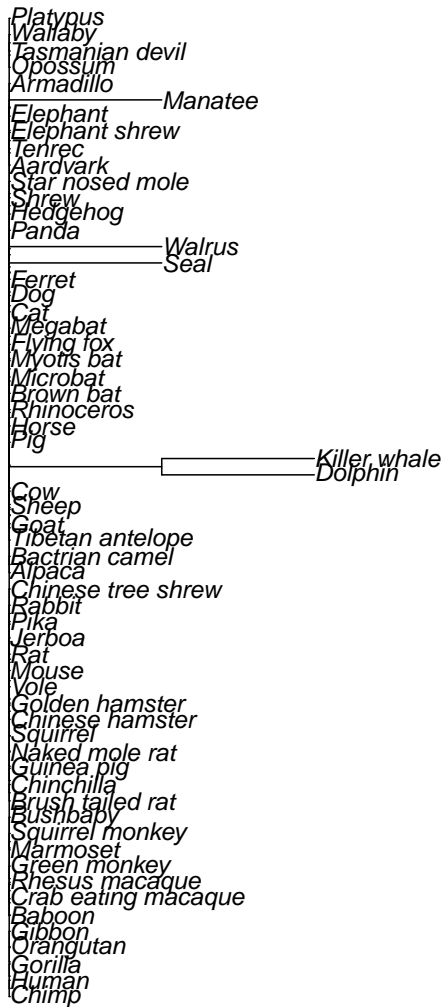
The plot on the left shows the tree you provided, with branch lengths 0 for all background lineages and branch lengths 1 for foreground lineages. The plot on the right displays the tree with all branch lengths 1 and the foreground lineages highlighted in blue. This binary tree represents the following as foreground lineages: all terminal branches leading to extant marine species, plus the branch leading to the common ancestor of the killer whale and the dolphin.

2) Generate a binary tree from a vector of foreground species using `foreground2Tree`. This uses the following input variables (all the options set here are also the defaults):

11

- **clade**: which of the branches within a foreground clade to keep as foreground. Options are "ancestral" to keep only the inferred transition branch, "terminal" to keep only terminal branches, and "all" to keep all branches.
- **transition**: whether to allow only transitions to the foreground state ("unidirectional", the default) or both to and from the foreground state ("bidirectional"). Since we are considering transitions to a marine environment, which has only occurred in one direction within mammals, we use "unidirectional" (the default) here.
- **weighted**: whether to distribute the "weight" of the foreground specification across all branches within each independent clade (default: FALSE).
- **useSpecies**: a vector that can be used to specify a subset of species to use in the analysis. These should be the same species used to estimate RER.

We can visualize how several possible choices influence the output binary phenotype tree, as follows:

2a) `clade = "ancestral"`: Use maximum parsimony to infer where transitions from background to foreground occurred in the tree, and set those transition lineages to foreground.

```
marineextantforeground = c("Walrus","Seal","Killer_whale","Dolphin","Manatee")
marineb2a = foreground2Tree(marineextantforeground, toyTrees, clade="ancestral",
                            useSpecies=names(logAdultWeightcm))
```

```
## Species from master tree not present in useSpecies: Cape_golden_mole
```

Clade: ancestral
Transition: unidirectional
Weighted: FALSE

The output first indicates that there is one species in the master tree that is not in the list of species we requested to be included using `useSpecies`: the cape golden mole. This species will be excluded from our analysis.

The plot shows that the branch leading to the common ancestor of killer whale and dolphin, as well as the branch leading to the common ancestor of walrus and seal, are foreground, along with the terminal branch leading to the manatee.

2b) `clade = "terminal"`: Set only terminal lineages leading to foreground species as foreground.

```
marineb2b = foreground2Tree(marineextantforeground, toyTrees, clade="terminal",
                            useSpecies=names(logAdultWeightcm))
```

```
## Species from master tree not present in useSpecies: Cape_golden_mole
```

Clade: terminal
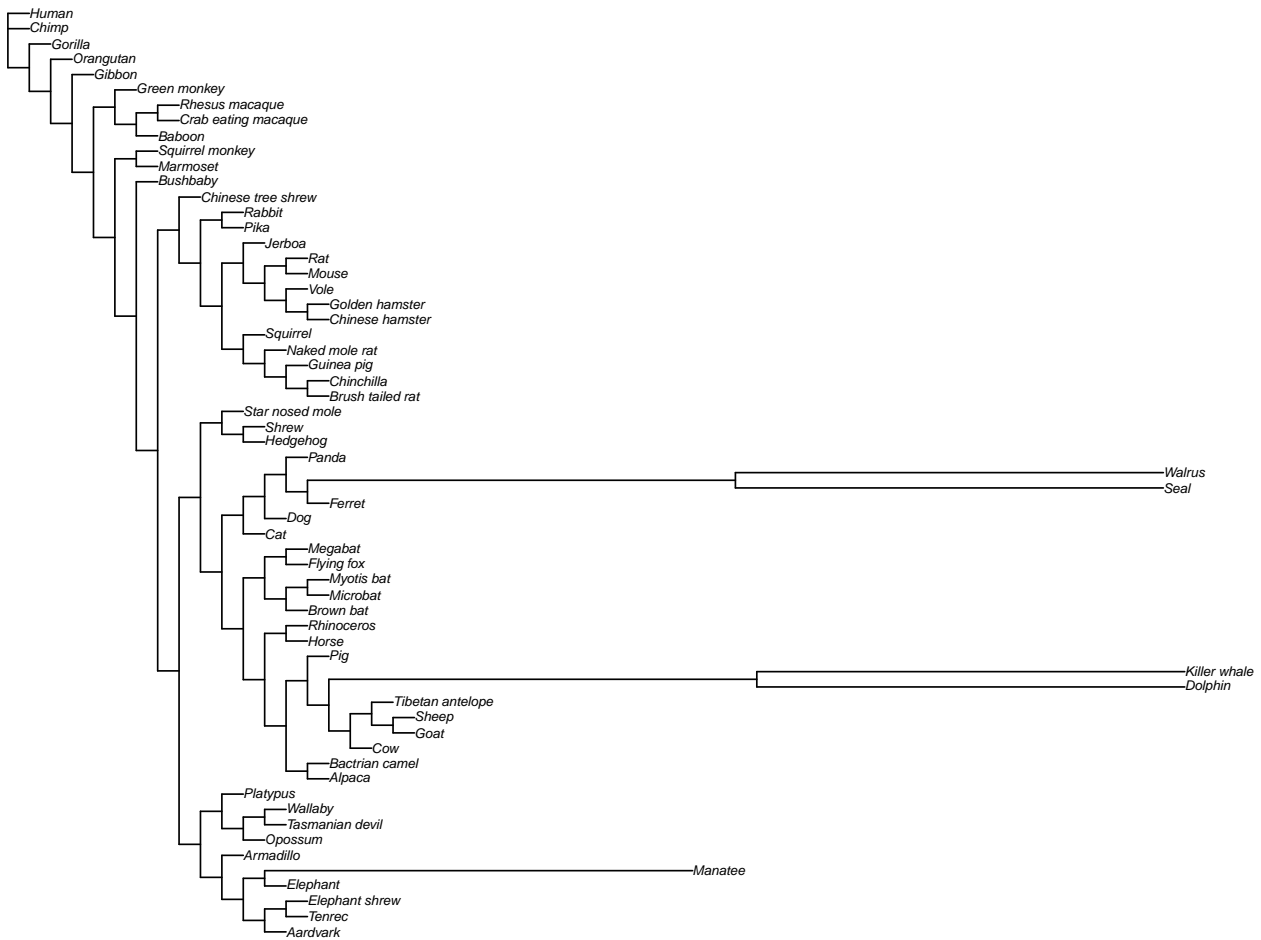**Transition: unidirectional**
Weighted: FALSE
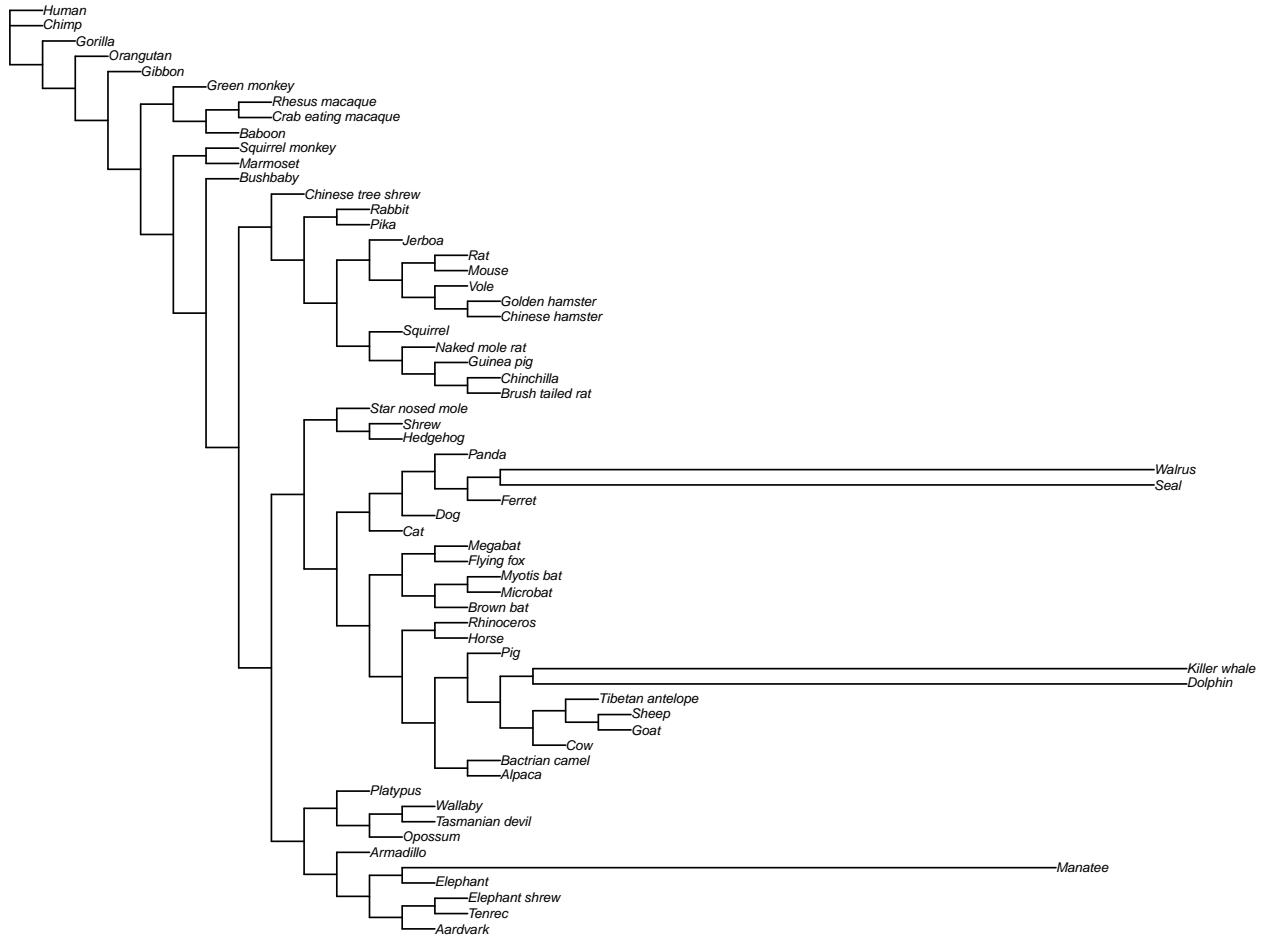
Here the each terminal branch leading to a marine species is foreground, but no internal branches are foreground.

2c) `clade = "all"`: Use maximum parsimony to infer where transitions from background to foreground occurred in the tree, and set those transition lineages, along with all daughter lineages, to foreground.

```
marineb2c = foreground2Tree(marineextantforeground, toyTrees, clade="all",
                            useSpecies=names(logAdultWeightcm))
```

```
## Species from master tree not present in useSpecies: Cape_golden_mole
```

Here the foreground branches are all those inferred to be transitional from 2a, as well as the terminal branches. If we had a case in which some branches daughter to the transition branches were not terminal, those would be included in the foreground as well.

2d) `clade = "all"` and `weighted = TRUE`: Infer transition and daughter branches as in 2c, but spread a weight of 1 evenly across all branches within each independent foreground clade.

```
marineb2d = foreground2Tree(marineextantforeground, toyTrees, clade="all", weighted = TRUE,
                        useSpecies=names(logAdultWeightcm))
```

```
## Species from master tree not present in useSpecies: Cape_golden_mole
```

Clade: all
**Transition: unidirectional**
Weighted: TRUE
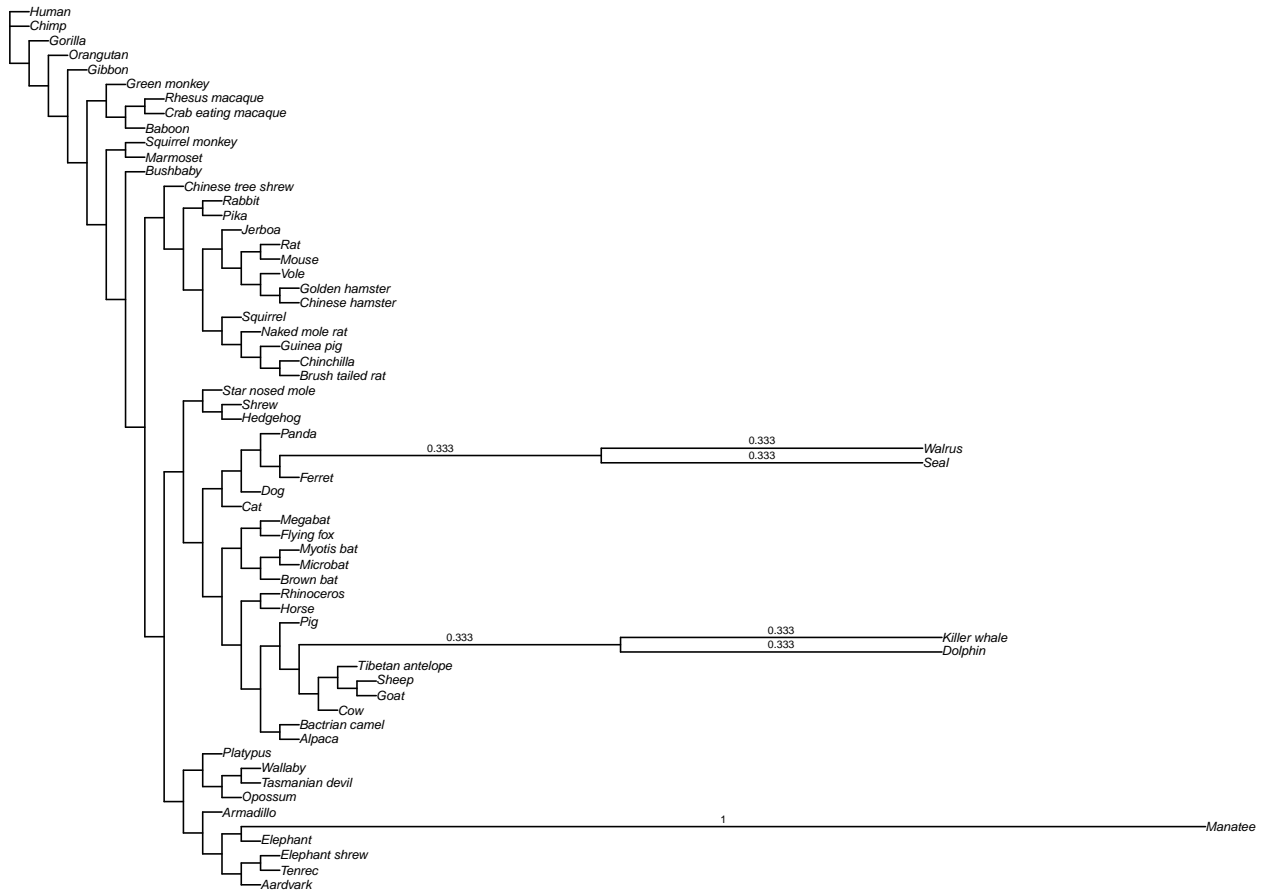
Here all branches in the cetacean and pinniped clades have length 1/3, whereas the terminal branch leading to the manatee has length 1. This is a way of distributing the weight given to each independent convergence event evenly across clades, rather than across lineages.

If you plot all the resulting trees, you can see how the different choices for `clade` influence the resulting branch lengths.

```
par(mfrow=c(2,2))
plot(marineb2a,cex=0.6,main="ancestral")
plot(marineb2b,cex=0.6,main="terminal")
plot(marineb2c,cex=0.6,main="all unweighted", x.lim=c(0,2.5))
labs2c = round(marineb2c$edge.length,3)
labs2c[labs2c==0] = NA
edgelabels(labs2c, col = 'black', bg = 'transparent', adj = c(0.5,-0.5),cex = 0.4,frame='n')
plot(marineb2d,cex=0.6,main="all weighted", x.lim=c(0,2.5))
labs2d = round(marineb2d$edge.length,3)
labs2d[labs2d==0] = NA
edgelabels(labs2d, col = 'black', bg = 'transparent', adj = c(0.5,-0.5),cex = 0.4,frame='n')
```
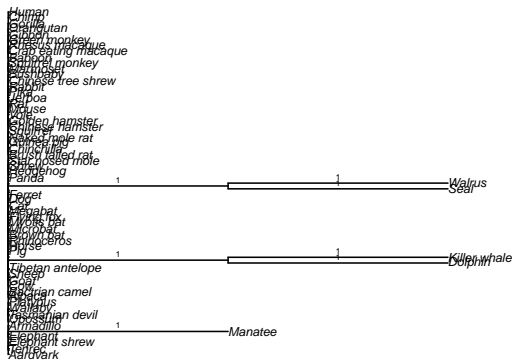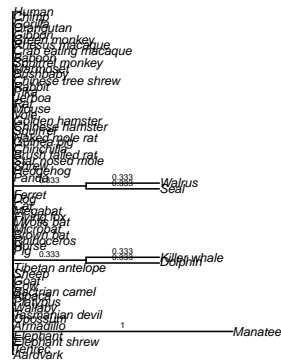
ancestral
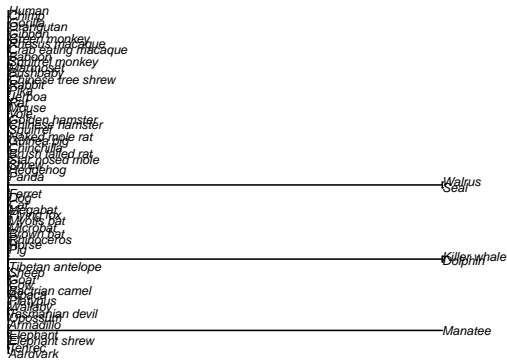
terminal
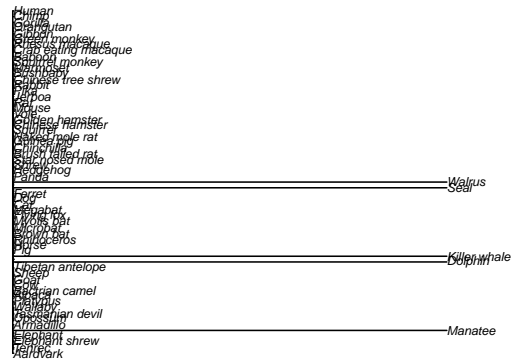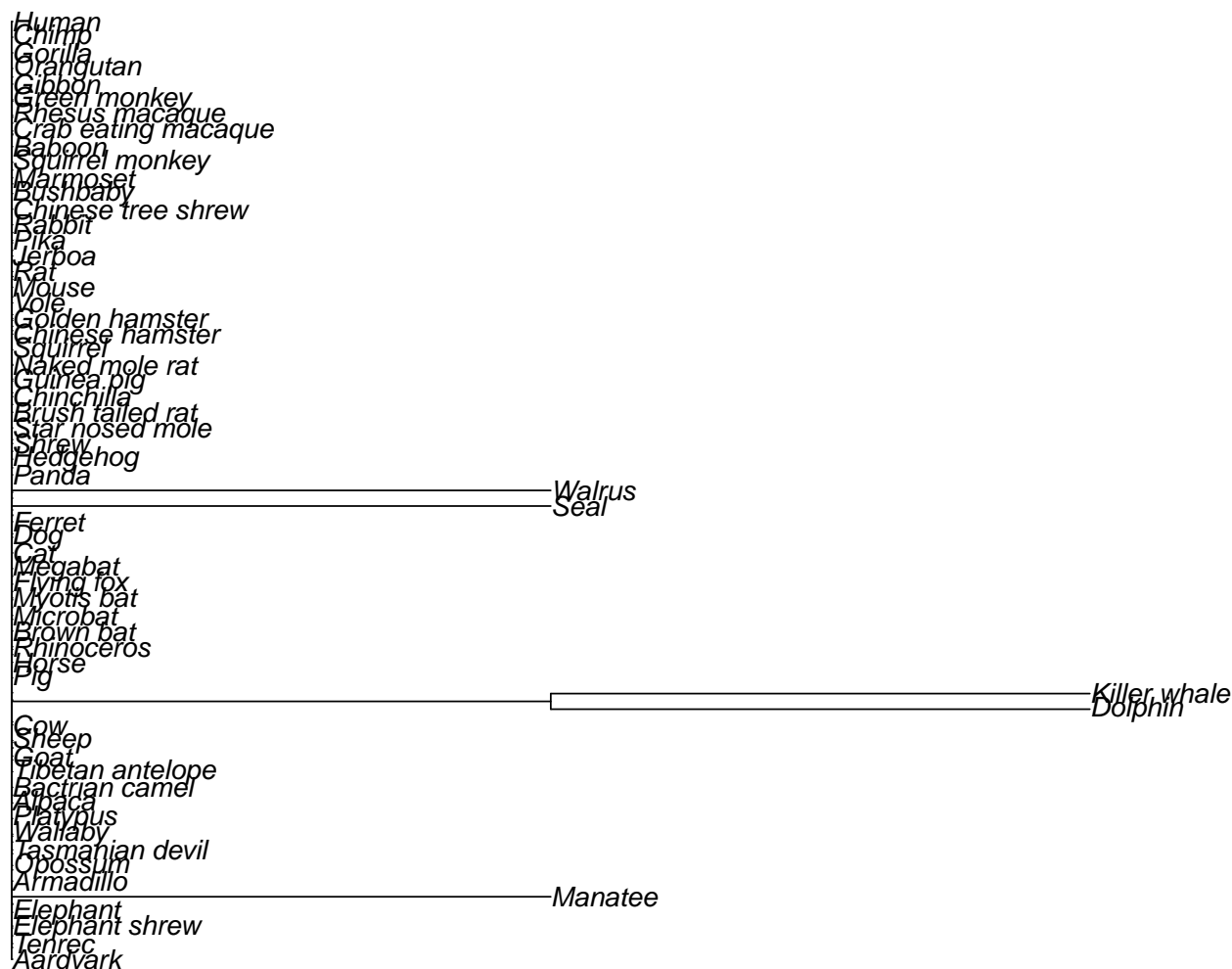

all unweighted

all weighted

None of these are the same as the binary tree that you specified in (1)!

```
plot(marineb,main="Manually specified binary tree (1)")
```

Manually specified binary tree (1)

Note that, in this tree, the branch representing the ancestor of the killer whale and the dolphin has a branch length of 1, whereas the branch representing the ancestor of seal and walrus has a branch length of 0. This shows that providing a binary trait tree file (1) allows you to specify which branches should be foreground with more flexibility than providing a set of foreground species to `foreground2Tree`.

3) Use the interactive branch selection tool. The following should open a plot of the master tree. When the GUI opens, select the marine foreground branches (Walrus, Seal, Killer whale, Dolphin, Killer whale-Dolphin ancestor, and Manatee), and click 'End selection.' Do not click 'New category'.

```
marineb3=click_select_foreground_branches(toyTrees$masterTree)
```

You can double check that the tree you created by manual selection (3) matches the binary tree you provided in (1) (it should) using the following code:

```
marineb3rooted=root(marineb3,outgroup=c("Platypus", "Wallaby","Tasmanian_devil","Opossum"))
par(mfrow=c(1,2))
plot(marinebrooted, main = "Provided binary tree (1)")
plot(marineb3rooted, main = "Trait tree from manual selection (3)")
```

Generating paths using `tree2Paths` or `foreground2Paths`

Some of the genes (like BEND3 above) may not have data for all species, meaning that their phylogeny will be a subset of the full phylogeny. To plot RERs for these genes and to correlate them with trait evolution, we run one of two functions that determine how the trait would evolve along all/many possible subsets of the full phylogeny, generating a set of **paths**. The function `tree2Paths` takes a binary tree as input, and the function `foreground2Paths` takes a set of foreground species as input. `foreground2Paths` has the same arguments as `foreground2Tree` described above (see option 2 in *Reading in or generating trait trees*).

Important note: If you are using a binary tree to create paths, it **must** have the same topology as the **master tree** or a subset of the **master tree** (see previous section for how this is generated).

```
phenvMarine=tree2Paths(marineb, toyTrees)
phenvMarine2=foreground2Paths(marineextantforeground, toyTrees, clade="all")
phenvMarine2b=tree2Paths(marineb2b, toyTrees)
```

Correlating gene evolution with binary trait evolution using `correlateWithBinaryPhenotype`

Now that we have estimates for the RERs for all genes of interest, as well as a representation of how the trait of interest evolves across the tree, we can use `correlateWithBinaryPhenotype` to test for an association between relative evolutionary rate and trait across all branches of the tree.

This uses the following input variables (all the options set here are also the defaults):

- `min.sp`: the minimum number of species in the gene tree for that gene to be included in the analysis. The default is 10, but you may wish to modify it depending upon the number of species in your master tree.
- `min.pos`: the minimum number of independent foreground (non-zero) lineages represented in the gene tree for that gene to be included in the analysis. The default is 2, requiring at least two foreground lineages to be present in the gene tree.
- `weighted`: whether to perform a weighted correlation where branch lengths represent weights. This can be used with the `weighted=TRUE` option in `foreground2Tree` or `foreground2Paths` to distribute phenotype weights across multiple lineages in a clade. The default is "auto", which will use weighted correlation if any branch lengths are between 0 and 1, and will use standard correlation otherwise.

```
corMarine=correlateWithBinaryPhenotype(mamRERw, phenvMarine, min.sp=10, min.pos=2,
                                        weighted="auto")
```

The text displayed shows which correlation method is used to test for association. Here it uses the default for binary traits: the Kendall rank correlation coefficient, or Tau.

The `correlateWithBinaryPhenotype` function generates a table with the following output for each gene:

1) Rho: the correlation between relative evolutionary rate and trait across all branches

2) N: the number of branches in the gene tree

3) P: an estimate of the P-value for association between relative evolutionary rate and trait.

4) p.adj: an estimate of the P-value adjusted for multiple comparisons using the Benjamini-Hochberg procedure (i.e., an estimate of the false discovery rate, or FDR).

Let's take a look at some of the top genes within this set.

```
head(corMarine[order(corMarine$P),])
```

```
##                 Rho   N           P      p.adj
## ANO2      0.2604946 106 0.001138448 0.1575810
## AK124326 -0.3064213  68 0.002292048 0.1575810
## BDH1      0.2423160 102 0.002997042 0.1575810
## BMP10    -0.2367746 103 0.003561152 0.1575810
```

```
## ATP2A1    0.2312257 101 0.004832241 0.1710613
## ASB15     0.2136137 107 0.007338788 0.1977725
```

Because we might expect different sets of genes to be in the positively or negatively correlated groups, it can be helpful to sort the table by the log-transformed p-values that carry the sign of Rho. Examining the extreme top or bottom of this sorting shows top positively and negatively correlated genes. This can be done as below in which we examine the top positively correlated genes; for negative correlations change decreasing = FALSE.

```
  corMarine$stat = -log10(corMarine$P) * sign(corMarine$Rho)
  head(corMarine[order(corMarine$stat, decreasing = TRUE),])
```

```
##                Rho   N           P      p.adj      stat
## ANO2    0.2604946 106 0.001138448 0.1575810 2.943687
## BDH1    0.2423160 102 0.002997042 0.1575810 2.523307
## ATP2A1  0.2312257 101 0.004832241 0.1710613 2.315851
## ASB15   0.2136137 107 0.007338788 0.1977725 2.134376
## TTN     0.1778351 119 0.018469789 0.2971957 1.733538
## ABCB4   0.1658190 107 0.037416063 0.3595490 1.426942
```

ANO2 and *BMP10* are two of the top genes in the positive and negative directions, respectively. Let's examine their RER plots.
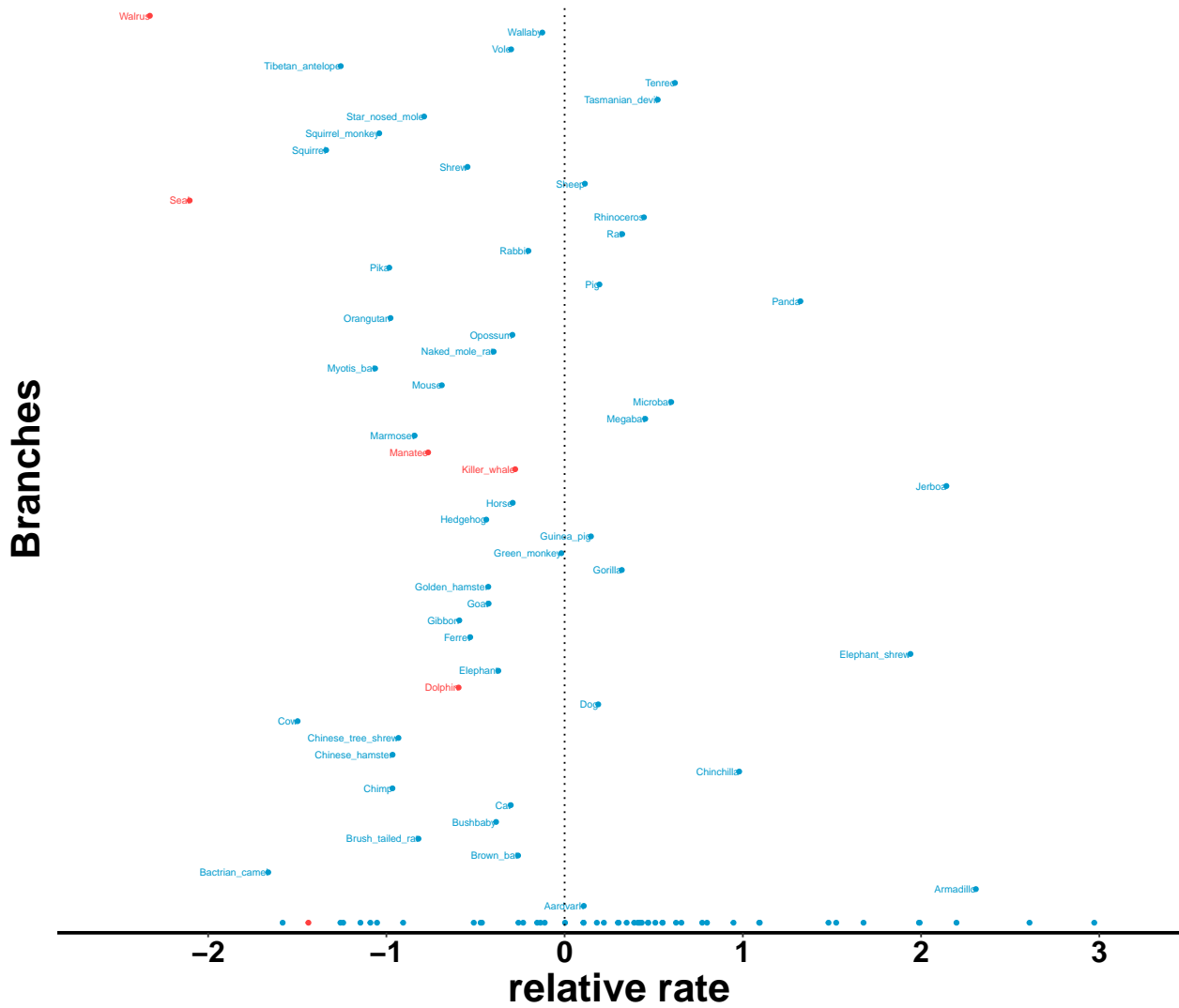
```
plotRers(mamRERw,"ANO2",phenv=phenvMarine)
```

ANO2: rho = 0.2605, p = 0.0011
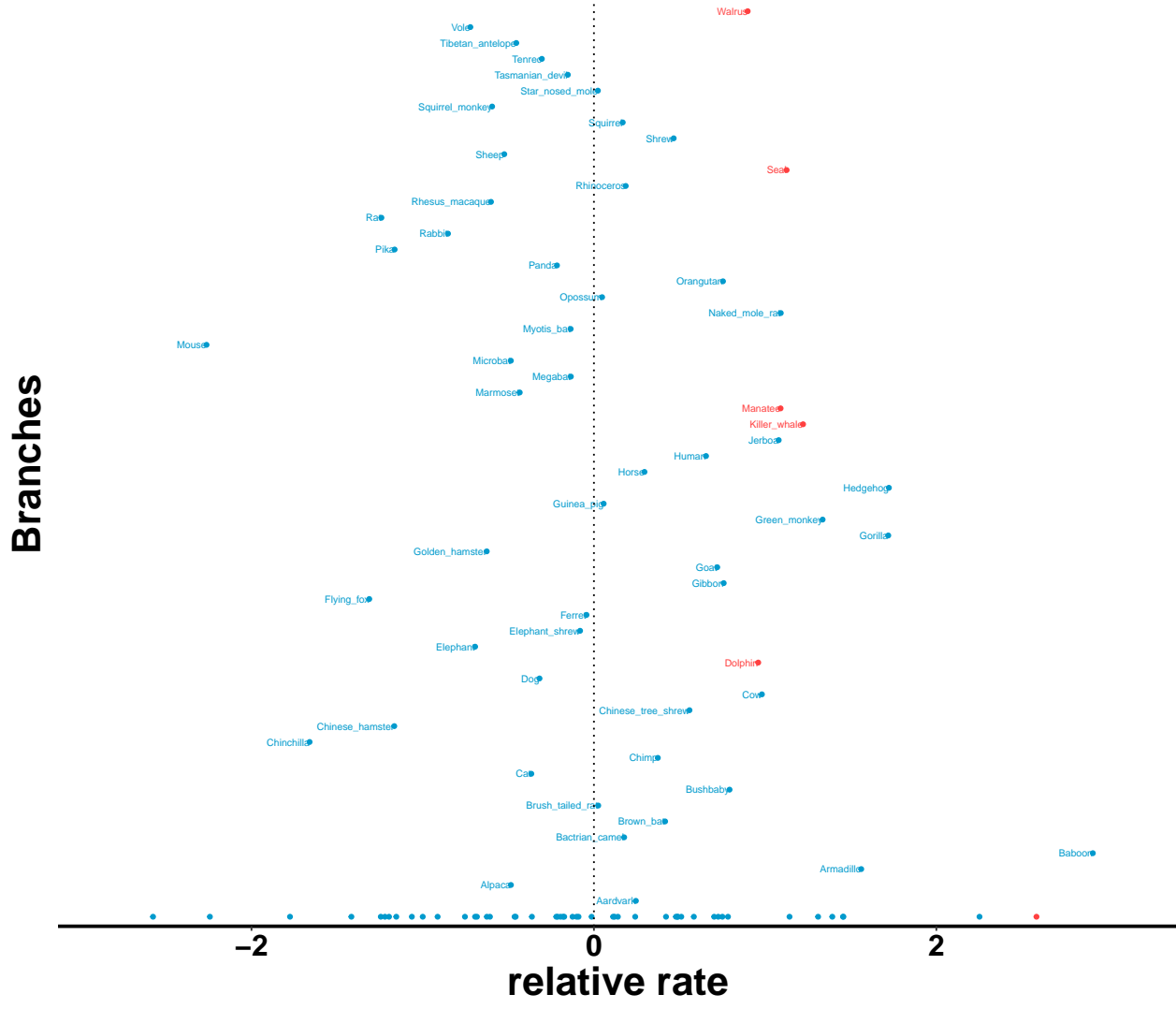
```
plotRers(mamRERw,"BMP10",phenv=phenvMarine)
```

BMP10: rho = −0.2368, p = 0.0036

In these RER plots, the marine lineages specified in the foreground are highlighted in red. The terminal branches are listed in alphabetical order, and internal branches are displayed at the bottom; note the red point at the bottom of each plot indicating the RER for killer whale-dolphin ancestral branch.

In the *ANO2* tree, marine lineages have high RER, leading to a positive Rho and a low p-value. In contrast, in the *BMP10* tree, marine lineages have low RER. This also yields a low p-value, but with a negative Rho.

To see what the overall pattern of association is across all genes in the set, we can plot a p-value histogram.

```
hist(corMarine$P, breaks=15, xlab="Kendall P-value",
     main="P-values for correlation between 200 genes and marine environment")
```

P−values for correlation between 200 genes and marine environment
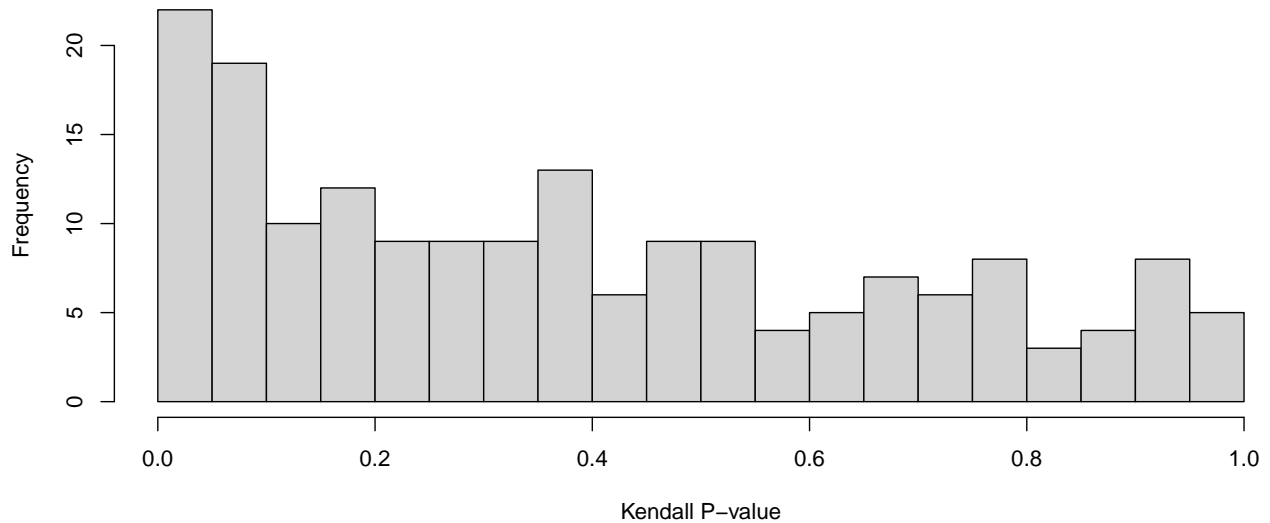
There appears to be a slight enrichment of low p-values, but since we have only evaluated the first 200 genes from our ~19,000 gene genome-wide set, we should hold off on drawing conclusions from this.

Continuous Trait Analysis

In addition to supporting binary trait analyses, RERconverge can also calculate correlations between rates of evolution of genes and the change in a *continuous* trait. To perform a continuous trait analysis, start with a named vector in R. Vector names must match the names in the trees read in previously. Here are the first few entries in the vector we will use for continuous trait analysis:

```
head(logAdultWeightcm)
```

```
##           Alpaca          Dolphin Chinese_tree_shrew         Tree_shrew
##        15.919981        17.609640           7.643856           7.643856
##          Manatee              Pig
##        18.296701        16.988152
```

We must convert the trait vector to paths comparable to the paths in the RER matrix. To do that, we can use the function 'char2Paths' as shown here:

```
charpaths=char2Paths(logAdultWeightcm, toyTrees)
```

```
## using metric diff, with filtering constant -1
```

```
## Species not present: Cape_golden_mole
```

We are using `metric diff`, which means that branch lengths are assigned to the trait tree based on the difference in trait values on the nodes connected to that branch.

The function tells us that there is one species in the master tree that is not present in the trait tree: the cape golden mole.

The `char2Paths` function creates a paths vector with length equal to the number of columns in the RER matrix. The phylogenetic relationships represented in the "char2Paths" output are the same as those represented in the RER matrix.

Finally, we can perform our ultimate analysis to find correlations between the rate of evolution of a genomic element (encoded in the RER matrix) and the rate of change of a phenotype (encoded in charpaths) using `correlateWithContinuousPhenotype`. The final output is the list of input genes with relevant statistics.

As input, we provide the RER matrix and trait path. `correlateWithContinuousPhenotype` is a wrapper function for continuous trait analysis using the `getAllCor` function. By default it performs Pearson correlation. To perfom a rank-based correlation (Spearman) use the `getAllCor` function with `method="s"`. For Spearman, Winsorizing extreme values is not necessary.

This function uses the following input variables (all the options set here are also the defaults):

- `min.sp`: the minimum number of species in the gene tree for that gene to be included in the analysis. The default is 10, but you may wish to modify it depending upon the number of species in your master tree.
- `winsorizeRER`/`winsorizetrait`: pulls the most extreme N values (default N=3) in both the positive and negative tails to the value of the N+1 most extreme value. This process mitigates the effect of extreme outliers before calculating correlations.

```
res=correlateWithContinuousPhenotype(mamRERw, charpaths, min.sp = 10,
                                     winsorizeRER = 3, winsorizetrait = 3)
head(res[order(res$P),])
```

```
##                   Rho   N           P      p.adj
## ANKRD18B  -0.3434618   80 0.001813809 0.2693473
## TTN        0.2721813  119 0.002748442 0.2693473
## ABCB4      0.2507133  107 0.009196076 0.4935868
## ADH7       0.2825531   79 0.011636013 0.4935868
## ATP10A    -0.2419800  105 0.012884245 0.4935868
## B3GALT1    0.4021305   34 0.018393504 0.4935868
```

In these results, Rho is the standard statistic for a Pearson correlation, N is the number of branches included in the analysis, and P is the uncorrected correlation p-value. Since huge numbers of statistical tests are being performed in these analyses, it is essential to correct p-values using a method such as the Benjamini-Hochberg correction (p.adj).

Because we might expect different sets of genes to be in the positively or negatively correlated groups, it can be helpful to sort the table by the log-transformed p-values that carry the sign of Rho. Examining the extreme top and bottom of this sorting shows top positively and negatively correlated genes. This can be done as below in which we examine the top positively correlated genes; for negative correlations change `decreasing = FALSE`.

```
  res$stat = -log10(res$P) * sign(res$Rho)
  head(res[order(res$stat, decreasing = TRUE),])
```

```
##                  Rho   N           P      p.adj     stat
## TTN        0.2721813  119 0.002748442 0.2693473 2.560913
## ABCB4      0.2507133  107 0.009196076 0.4935868 2.036397
## ADH7       0.2825531   79 0.011636013 0.4935868 1.934196
## B3GALT1    0.4021305   34 0.018393504 0.4935868 1.735336
## BDH1       0.2153956  102 0.029692553 0.4938339 1.527352
## ALG3       0.1951868   97 0.055375586 0.5819799 1.256682
```

We can also plot a distribution of our uncorrected p-values to allow us to speculate if they will remain significant after correction. A mostly uniform distribution with an elevated frequency of low p-values indicates the presence of genomic elements whose rates of evolution are correlated with the phenotype (note that this trend will also be increasingly distinct when larger numbers of genomic elements are considered).

```
hist(res$P, breaks=15)
```

Histogram of res$P
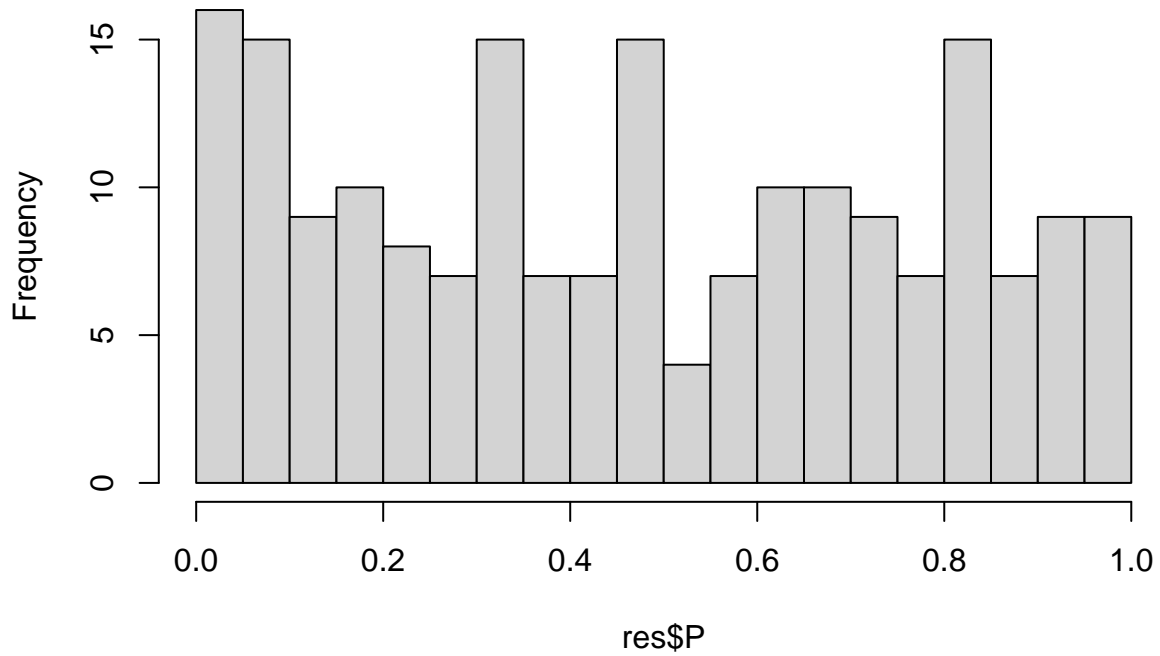
One important consideration of the results is the impact of one or a few species on the overall correlation. To assess this risk, we can examine individual correlation plots as follows:

```r
x=charpaths
y=mamRERw['TTN',]
pathnames=namePathsWSpecies(toyTrees$masterTree)
names(y)=pathnames
plot(x,y, cex.axis=1, cex.lab=1, cex.main=1, xlab="Weight Change",
     ylab="Evolutionary Rate", main="Gene TTN Pearson Correlation",
     pch=19, cex=1, xlim=c(-2,2))
text(x,y, labels=names(y), pos=4)
abline(lm(y~x), col='red',lwd=3)
```

Gene TTN Pearson Correlation
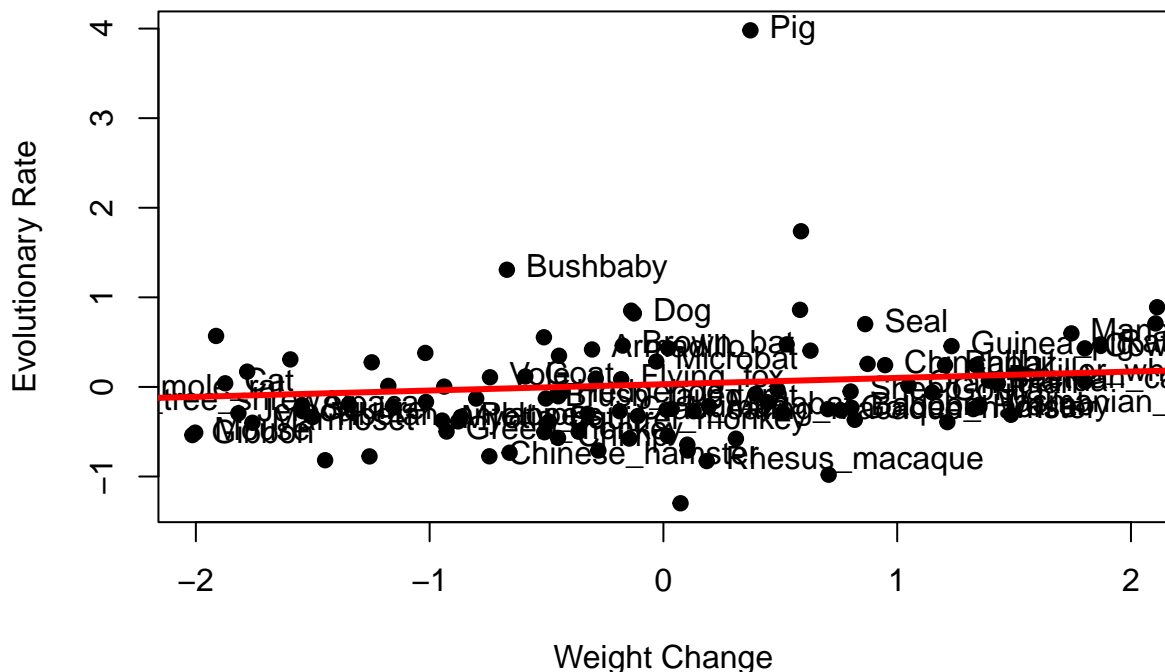
In this case, we see that the positive correlation is driven by all species and not just a single clade. Note that labelled points are terminal nodes in the phylogeny and unlabelled points are internal nodes.

Further analyses could include using functional enrichment detection methods to find functionally-related genomic elements that are experiencing convergent evolutionary rates as a group (see walkthrough below) and using branch-site models to determine if fast-evolving genes are experiencing relaxation of constraint or undergoing directional selection.

Enrichment Walkthrough

This section describes how to run pathway enrichment analysis using RERconverge output and functions included with the RERconverge package. Enrichment analysis detects groups of genes that are evolving faster or slower with a phenotype of interest. In the RERconverge package, the enrichment function is implemented as a Wilcoxon Rank-Sum Test on a list of genes ranked based on their correlation statistics. It detects distribution shifts in groups of genes compared to all genes, and it thereby bypasses the need to set a foreground significance cutoff like some other enrichment methods.

Input to the enrichment function is the output from RERconverge correlation functions and pathways of interest with gene symbols (gene names).

Output is enrichment statistics for each pathway, including genes in the pathways and their ranks.

Extract Results from RERconverge Correlation Analysis

Enrichment analysis starts with the results from the *correlateWithContinuousPhenotype*, *correlateWithBinaryPhenotype*, or *getAllCor* functions in the RERconverge package. These results include Rho, p-value, and the Benjamini-Hochberg corrected p-value for the correlation between the relative evolutionary rates of each gene and the phenotype provided. These statistics are used to calculate enrichments.

In this case, we will start with the data from the continuous trait analysis described above that used adult mass as the phenotype of interest in mammalian species. This walkthrough assumes that you have already installed

RERconverge, and it uses output from the *correlateWithContinuousPhenotype* function ("RERresults") that is included with the RERconverge package.

```
library(RERconverge)
data("RERresults")
```

Deriving a ranked gene list

We will perform our enrichment on Rho-signed negative log p-values from our correlation results. The *getStat* function converts our correlation results to these values, removes NA values, and returns a named numeric vector where names correspond to rownames from the correlation results (in this case, gene names).

```
library(RERconverge)
stats=getStat(RERresults)
```

Import Pathway Annotations

Now that we have our gene statistics, we need pathway annotations. Download all curated gene sets, gene symbols (c2.all.v6.2.symbols.gmt) from GSEA-MSigDB as gmtfile.gmt. You must register for an account prior to downloading. The rest of the vignette expects "gmtfile.gmt" to be in the current working directory.

With the file in the appropriate place, simply use the *read.gmt* function to read in the annotation data.

```
annots=read.gmt("gmtfile.gmt")
```

Format Pathway Annotations
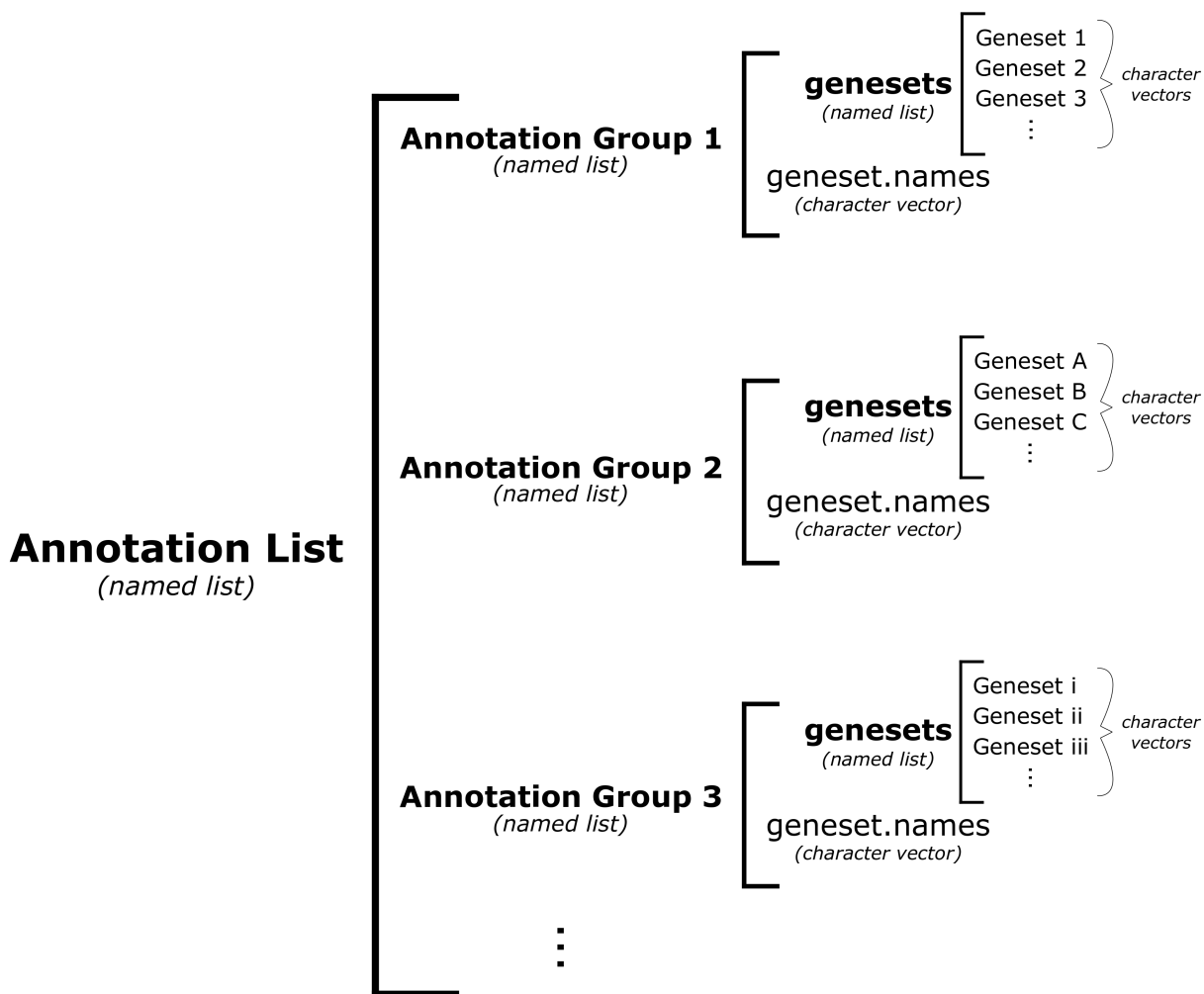
RERconverge enrichment functions expect pathways to be in named pathway-group lists contained within a list (see diagram above). A pathway-group is a group of similar pathways stored together (for example KEGG pathways might be one pathway-group and MGI pathways might be another pathway-group). Each pathway-group list contains a named list called *genesets* with each element of the list a character vector of gene names in a particular pathway, and the names of the elements the names of the pathways. The names of the genesets are also contained as a character vector in the second element of the pathway-group list named *geneset.names*. As an example of correctly-formatted annotations, you would have a list called *annotations* that contains another list called *canonicalpathways*. The *canonicalpathways* list contains a list named *genesets* and a vector named *geneset.names*. Each element of the *genesets* list is a character vector of gene names corresponding to a particular pathway, and the names of the elements in *genesets* are the names of the pathways. The *geneset.names* vector contains the names of the elements in *genesets*.

To convert our gmt file to the correct format, we simply need to put it inside another list; the *read.gmt* function automatically reads in gmt files in the format described above.

```
annotlist=list(annots)
names(annotlist)="MSigDBpathways"
```

Calculate Enrichment Using *fastwilcoxGMTall*

We can now use *annotlist* and *stats* to calculate pathway enrichment. We will use the function *fastwilcoxGM-TAll*, which calculates the estimated Wilcoxon Rank-Sum statistics based on the ranks of the genes in *stats*. The test essentially measures the distribution shift in ranks of genes of interest (each pathway) compared to the background rank distribution (ranks of all other genes included in the pathway-group annotations that are not part of the pathway of interest). We set *outputGeneVals* to true so the function returns names of the genes in the pathway and their ranks. *num.g* specifies the minimum number of genes required to be present to run the enrichment (10 by default).

```
enrichment=fastwilcoxGMTall(stats, annotlist, outputGeneVals=T, num.g=10)
```

```
## 25 results for annotation set MSigDBpathways
```

Note that these results contain statistics for all pathways with the minimum number of genes specified. Very few pathways have a sufficient number of genes because we only calculated correlation statistics for 200 total genes.

Further Analysis and Visualization

You may visualize your pathway results in a variety of ways, including using **PathView** to overlay correlation colors on KEGG pathways and **igraph** to make pathway networks based on gene similarity among networks.

You may also calculate permutation p-values based on permuting the phenotype vector and rerunning the correlation and enrichment analyses many times to filter pathways that always tend to have significant enrichment values regardless of the phenotype input.

Conclusion

We've now walked through the basic workflow for RERConverge. For more information about these methods and some results relevant to marine and subterranean adaptation, see (Chikina, Robinson, and Clark 2016) and (Partha et al. 2017).

References

Chikina, Maria, Joseph D Robinson, and Nathan L Clark. 2016. "Hundreds of Genes Experienced Convergent Shifts in Selective Pressure in Marine Mammals." *Molecular Biology and Evolution* 33 (9): 2182–92. https://doi.org/10.1093/molbev/msw112.

Partha, Raghavendran, Bharesh K Chauhan, Zelia Ferreira, Joseph D Robinson, Kira Lathrop, Ken K Nischal, Maria Chikina, and Nathan L Clark. 2017. "Subterranean mammals show convergent regression in ocular genes and enhancers, along with adaptation to tunneling." *eLife* 6 (October): e25884. https://doi.org/10.7554/eLife.25884.

Schliep, Klaus Peter. 2011. "phangorn: Phylogenetic analysis in R." *Bioinformatics* 27 (4): 592–93. https://doi.org/10.1093/bioinformatics/btq706.