phytools 2.0

An updated R ecosystem for phylogenetic comparative methods (and other things)

Liam J. Revell, University of Massachusetts Boston

24 February, 2023

Abstract

1. Phylogenetic comparative methods comprise the general endeavor of using an estimated phylogenetic tree (or set of trees) to make secondary inferences: about trait evolution, diversification, biogeography, community ecology, and a wide variety of other phenomena or processes.
2. Over the past ten years, the *phytools* R package (Revell, 2012) has grown to become one of the most important tools for phylogenetic comparative analysis.
3. *phytools* is a diverse contributed R library now consisting of hundreds of different functions covering a wide range of methods and purposes in phylogenetic biology.
4. As of the time of writing, *phytools* included functionality for fitting models of trait evolution, for reconstructing ancestral states, for studying diversification on trees, and for visualizing phylogenies, comparative data, and fitted models, as well as for a number of other things.
5. Here, I describe *some* of the major features of and recent updates to *phytools*, but I also illustrate several popular workflows of the *phytools* computational software.

Introduction

Phylogenetic trees are the directed graphs used to represent historical relationships among a set of operational taxa that are thought to have arisen via a process of descent with modification and branching. Operational taxa in a reconstructed phylogenetic tree might be gene copies, paralogous and orthologous members of a gene family, viral sequences, whole genomes, human cultural groups, or biological species. According to its broadest definition, the phylogenetic comparative method corresponds to the general activity of using a known or (most often) inferred phylogenetic tree to learn something *else* (apart from the relationships indicated by the tree) about the evolutionary process or past, the contemporary ecology, the biogeographic history, or the origins via diversification, of the particular operational taxa of our estimated tree (Harvey and Pagel, 1991; Harmon, 2018; Revell and Harmon, 2022).

Modern phylogenetic comparative methods are not new. Perhaps the most important article in the development of the phylogenetic approach to comparative biology (Felsenstein, 1985) was first authored nearly 40 years ago, and was even the subject of a relatively recent retrospective (Huey et al., 2019). Nonetheless, it's fair to say that phylogenetic comparative methods have seen a relatively impressive expansion and diversification over the past two decades. This has included the development of new approaches for studying the generating processes of trees (in other words, speciation and extinction), the relationship between phenotypic traits and species diversification, and a range of approaches for investigating heterogeneity in the evolutionary process over the tree of life. Phylogenetic comparative methods have also begun to be applied outside their traditional domain of evolutionary research. In particular, phylogenies and the comparative method have made recent appearances in studies on infectious disease epidemiology, virology, sociolinguistics, biological anthropology, molecular genetics, and community ecology.

The scientific computing environment R (R Core Team, 2023) is widely-used in biological research. One of the major advantages that R provides is that it empowers computational scientists and independent developers

to build functionality on top of the basic R computing platform. This functionality often takes the form of what are called *contributed R packages*: libraries of related functions built by individuals or research collaboratives not part of the core R development team. The growth of importance of R in phylogenetic biology stems entirely from contributed R package. Among these, the most important core function libraries are *ape* (Paradis et al., 2004; Popescu et al., 2012; Paradis and Schliep, 2019), *geiger* (Harmon et al., 2008; Pennell et al., 2014), *phangorn* (Schliep, 2011), and my package, *phytools* (Revell, 2012).

phytools is an R function library dedicated *primarily* to phylogenetic comparative analysis, but also including approaches and methodologies in a range of other domains of phylogenetic biology: not restricted to, but especially, visualization. The original article describing *phytools* is now more than 10 years old, and though I recently published a more comprehensive book on the subject of phylogenetic comparative methods in the R environment (Revell and Harmon, 2022), I nonetheless felt that it was time to provide a briefer (although this article is by no means brief) update of *phytools* (in particular) for the primary scientific literature. *phytools* has now grown to be *very large* – consisting of hundreds of functions, a documentation manual that's over 200 pages in length, and (literally) tens of thousands of lines of computer code. As such, I thought it would be most useful to rather briefly summarize some of the functionality of the *phytools* R package in a few different areas, and then provide a limited number of example workflows for the "2.0" version of the *phytools* package.

Overview

The *phytools* R package contains functionality in a wide diversity of different research areas of phylogenetics and phylogenetic biology. Rather than try to comprehensive survey this functionality, what I've elected to do instead, is briefly review a smaller number of methodological areas, and then illustrate each of these with one to three example analysis workflows, including the corresponding R code that can be used to reproduce the analysis and results presented here.

Installing and loading *phytools*

This article assumes that most readers already have some familiarity with the R computing environment, and have previously installed contributed R packages. Nonetheless, to get started using *phytools* for the first time, the easiest way to install the package locally is by using the R *base* function called `install.packages`, which will download and install *phytools* from its CRAN page. (CRAN is an acronym for Comprehensive R Archive Network: a network of mirror repositories used both to archive and distribute R and contributed R packages.) This can be done as follows.

```
install.packages("phytools")
```

Readers undertaking phylogenetic analysis in the R environment for the first time will note that several other R packages are installed automatically in addition to the one (*phytools*) that we'd requested. These are packages upon which *phytools depends* – meaning that *phytools* uses one or multiple functions exported by each of these packages in its own R code. More will be said later about the dependency relationship between *phytools* and other packages of the R and R phylogenetic ecosystems.

Having installed *phytools*, to proceed and use it in an interactive R session, we normally load it. (Loading an R package simply makes the names of the functions visible and available in our current R session.)

```
library(phytools)
```

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

```
## Loading required package: maps
```

Discrete characters

The *phytools* package now contains a wide range of different methods and models for the analysis of *discrete character evolution* on trees. In this section, I'll discuss just a few of these.

Stochastic character mapping

Perhaps the most important and widely-used discrete character method of *phytools* is a method called 'stochastic character mapping' (Huelsenbeck et al., 2003). Stochastic character mapping is an analytical technique in which we randomly sample discrete character histories ("stochastic maps") of our trait on the tree under a specified model. One example of a stochastic character mapping analysis would be to first fit (e.g., using Maximum Likelihood) the character transition model (a variant of the Mk discrete character evolution model of Lewis, 2001; also see Harmon, 2018), and then proceed to randomly sample a set of 1,000 (perhaps) discrete character histories based on this character. (Other workflows are also popular and possible to undertake within R. For instance, rather than use a model of character evolution that has been optimized using Maximum Likelihood, one can instead sample parameters of the evolutionary process from their joint posterior probability distribution using Bayesian MCMC.)

To illustrate this workflow, I'll use a discrete, ecological trait for a small phylogeny of centrarchid fishes from Revell and Collar (2009). Since the trait (which we'll refer to as "feeding mode") is binary, meaning it only takes two levels, there are a total of four possible discrete character (extended Mk) models: equal back-and-forth transitions between the two character values; different rates; and then the two different irreversible trait evolution models. *phytools* now allows us to fit each of these four different models, compare them, and then pass the model weights and fitted models directly to our stochastic mapping function. Our mapping function (called `simmap`) will then proceed to automatically sample stochastic character histories with probabilities that are proportional to each model weight.

For this example, our data have been packaged with the *phytools* library. so we can load them in an interactive R session using the function `data` as follows.

```
data(sunfish.tree)
data(sunfish.data)
```

For our Mk model-fitting function (which here will be the *phytools* function `fitMk`), and for many other character methods of the *phytools* R package, our input data typically takes the form of a character or factor vector. Personally, I prefer to use factors, because in that case we can easily access the levels assumed by the character through a call of the *base* R function `levels`. (In this example our input data consists of a data frame in which the `feeding.mode` column is code a factor. In general, however, had we read this same data from an input text file in, for example, comma-separated-value format, R would have created a character, rather than factor, formatted column by default. To adjust this we can set the argument `stringsAsFactors=TRUE` in our file-reading function, which, in that case, might be the *base* R function `read.csv`.)

```
feeding_mode<-setNames(sunfish.data$feeding.mode,
  rownames(sunfish.data))
levels(feeding_mode)
```

```
## [1] "non"  "pisc"
```

Here we see that our factor vector has two levels: `"non"` and `"pisc"`. These two character levels refer to non-piscivorous and piscivorous fish species in this group. Since R factors have no particular character limit on their levels, let's update our data according: once again using the function `levels`. (`levels` is an odd R function in that it can serve both as an *extractor* function, that pulls out the levels of a factor; as well as acting as an assignment or *replacement* function, in which the levels of the factor are updated. When we adjust our factor levels for `feeding_mode`, we're using `levels` in this latter fashion.)

```
levels(feeding_mode)<-c("non-piscivorous","piscivorous")
levels(feeding_mode)
```

```
## [1] "non-piscivorous" "piscivorous"
```

Now we're ready to proceed and fit our models. To do so, in this case, I'll use the *phytools* function `fitMk` and fit a total of four models, as previously indicated: `"ER"`, the equal rates model; `"ARD"`, the all-rates-different model in which we allow different forward and backward rates for our character; and, lastly, the two different irreversible models – one in which non-piscivory can evolve to piscivory, but not the reverse; and a second in which the opposite is true. For our irreversible models, we'll tell `fitMk` how to fit the model by creating and supplying a *design matrix* for each model. This design matrix should be of dimensionality $k \times k$, for k levels of the trait, with integer values in the positions of the matrix corresponding to allowed transitions and zeros otherwise. (We use different non-zero integer value for each rate that we want to permit to assume a different value.)

Since our $k = 2$, this is very easy; however, the same principle would apply to any value of k (Revell and Harmon, 2022).

```
ER.model<-fitMk(sunfish.tree,feeding_mode,
  model="ER")
ARD.model<-fitMk(sunfish.tree,feeding_mode,
  model="ARD")
Irr1.model<-fitMk(sunfish.tree,feeding_mode,
  model=matrix(c(0,1,0,0),2,2,byrow=TRUE))
Irr2.model<-fitMk(sunfish.tree,feeding_mode,
  model=matrix(c(0,0,1,0),2,2,byrow=TRUE))
```

Having fit our four models, we can also compare them to see which is best-supported by our data. I'll use a generic `anova` function call it pass it each of our fitted models as separate arguments in the function. `anova` will print the results of our model comparison; however, it's very important that we assign the *result* of `anova` to a new object (in my example, I'll call it `sunfish.aov`, but this is arbitrary) as follows.

```
sunfish.aov<-anova(ER.model,Irr1.model,Irr2.model,
  ARD.model)
```

```
##               log(L) d.f.      AIC    weight
## ER.model   -13.07453    1 28.14906 0.3486479
## Irr1.model -12.98820    1 27.97640 0.3800846
## Irr2.model -14.20032    1 30.40064 0.1130998
## ARD.model  -12.86494    2 29.72987 0.1581677
```

This table shows each of our fitted model names, their log-likelihoods, the number of parameters estimated for each model, a value of the Akaike Information Criterion (AIC), and the Akaike model weights. Smaller values of AIC indicate better support for the corresponding model, taking into account its parameter complexity. Model weights indicate the "weight of evidence" favoring each of our prior hypotheses.

With the result of our `anova` call in hand (as the `sunfish.aov` object), we're ready to pass it directly to the generic `simmap` method. By design, doing so will tell `simmap` to generate stochastic character maps under each of our four models with relative frequencies that are equal to the weight of evidence supporting of each model. (If we preferred, we could've just generated stochastic character maps for the best-supported of our four models. Using the `simmap` generic method, this would be done by setting the argument `weighted=FALSE`, or simply by passing our favored Mk model directly to the function.)

```
sunfish.simmap<-simmap(sunfish.aov,nsim=1000)
sunfish.simmap
```

```
## 1000 phylogenetic trees with mapped discrete characters
```

In spite of the significant number of stochastic simulations involved, this analysis should run fairly quickly (obviously, depending on the speed of our computer). In part this is because it *does not* involve estimating the Mk transition matrix, $\mathbf{Q}$, for each sampled model. An additional significant advantage of this approach

is that it has also permitted us to (partly) account for uncertainty in our model estimation that's due to uncertainty in model selection.

Figure 1 shows a set of six, randomly chosen stochastic character histories for our trait (feeding mode) on our input phylogeny. Readers should see that each of these are consistent with our observed value of the binary trait at the tips of the tree, but each different one from the other in the specific hypothesis of trait evolution that it represents.

To create my color palette for plotting I used a different contributed R package that we haven't seen yet called *viridisLite* by Garnier et al. (2022). To replicate this plot exactly, users should first install *viridisLite* from CRAN by running `install.packages("viridisLite")`, but do not need to load it.

```
cols<-setNames(viridisLite::viridis(n=2),
  levels(feeding_mode))
par(mfrow=c(2,3))
plot(sample(sunfish.simmap,6),ftype="i",fsize=0.6,
  colors=cols,offset=0.2)
```

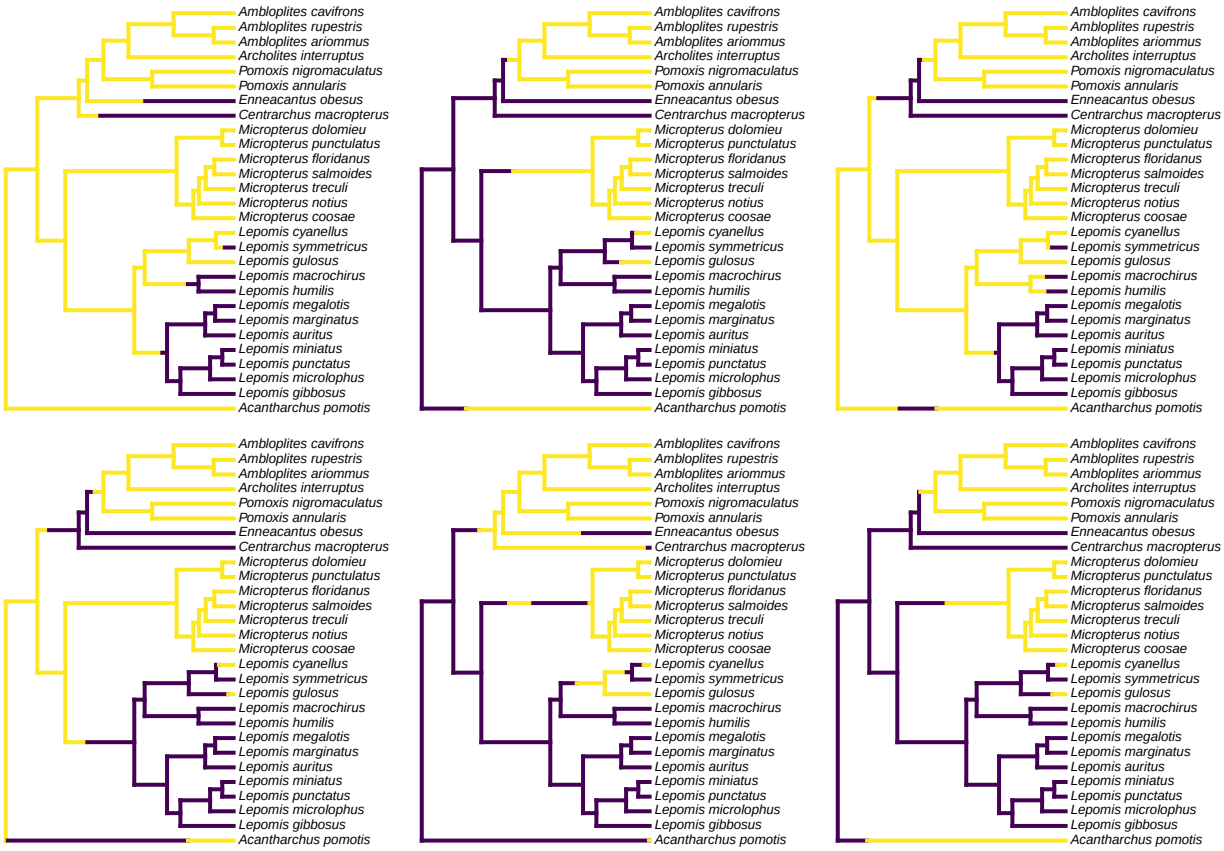

Figure 1: Six randomly chosen stochastic character maps of feeding mode (non-piscivorous vs. piscivorous) on a phylogeny of 28 centrarchid fish species. Stochastic character mapping involves randomly sampling character histories that are consistent with our tip data in proportion to their probability under a model. In this case, histories were sampled under the set of four alternative Mk models of a binary trait, in proportion to the weight of evidence supporting each model.

Although this already gives a sense of the uncertainty of our ancestral character history on the tree for our trait, most commonly we don't simply graph a subset (or all) of our stochastic mapped trees. Typically, instead, we first *summarize* our stochastic character maps (in multiple ways) before proceeding to plot or

analyze these summarized results.

Perhaps most often, *phytools* users undertaking stochastic character mapping want to compute the posterior probabilities of each value of the character trait at each internal node of the tree. These values are calculated using the generic `summary` method for our object class, which can then be plotted using a generic `plot` function call as follows.

```
plot(summary(sunfish.simmap),ftype="i",fsize=0.7,
  colors=cols)
legend("topleft",levels(feeding_mode),pch=21,
  pt.cex=2,pt.bg=cols,bty="n")
```

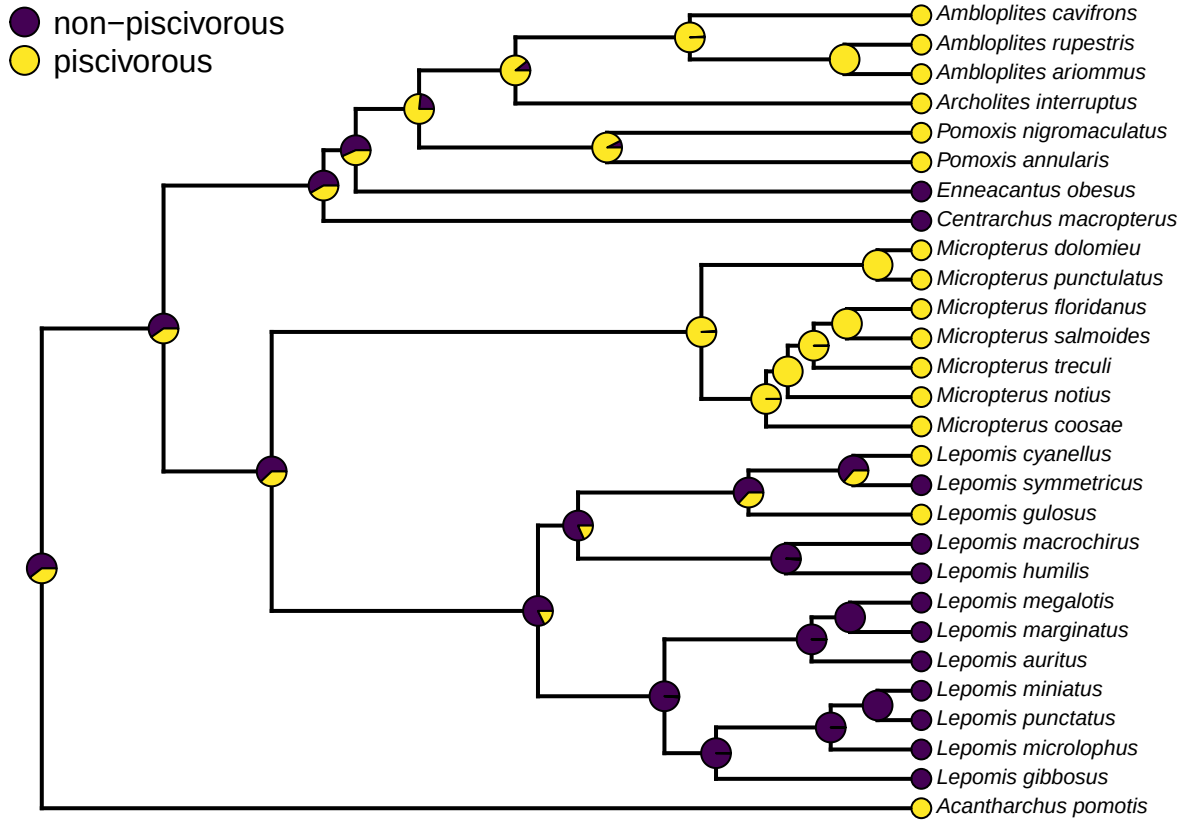

Figure 2: Posterior probabilities at each ancestral node of the centrarchid tree of Figure 1 from stochastic character mapping using model weights to sample across four different extended Mk trait evolution models.

A correct interpretation of the graph of Figure 2 shows the observed discrete character states (at the tips of the tree) and the posterior probabilities from stochastic mapping that each internal node is in each state, in which transition models were sampled in proportion to the weight of evidence in support of each model.

In addition to node probabilities, *phytools* users undertaking a stochastic character mapping analysis are often interested in the number of changes of each type that is implied by the evolutionary process and our data. Stochastic mapping samples character histories and thus can be used to produce an estimate of the posterior probability distribution of the *number* of character changes of each type on the tree, under our sampled model or models.

To obtain this, we'll first call the generic function `density` which, when given an object from stochastic mapping, computes the relative frequency distribution of changes of each type. We can then graph these distributions (remember, our character is binary, to there are only two types of character state change:

non-piscivorous → piscivorous, and the reverse) using a different generic `plot` method.

To recreate this analysis completely, readers will also have to install an additional contributed R package called *coda* (Plummer et al., 2006) in the same way that we installed *phytools* earlier.

```
sunfish.density<-density(sunfish.simmap)
```

```
## Loading required package: coda
```

```
par(mfrow=c(1,2),las=1,cex.axis=0.8,cex.lab=0.9)
COLS<-setNames(cols,sunfish.density$trans)
plot(sunfish.density,ylim=c(0,0.6),transition=names(COLS)[1],
    colors=COLS[1],main="")
mtext("a) transitions to piscivory",line=1,
  adj=0,cex=0.9)
plot(sunfish.density,ylim=c(0,0.6),transition=names(COLS)[2],
  colors=COLS[2],main="")
mtext("b) transitions to non-piscivory",line=1,
  adj=0,cex=0.9)
```

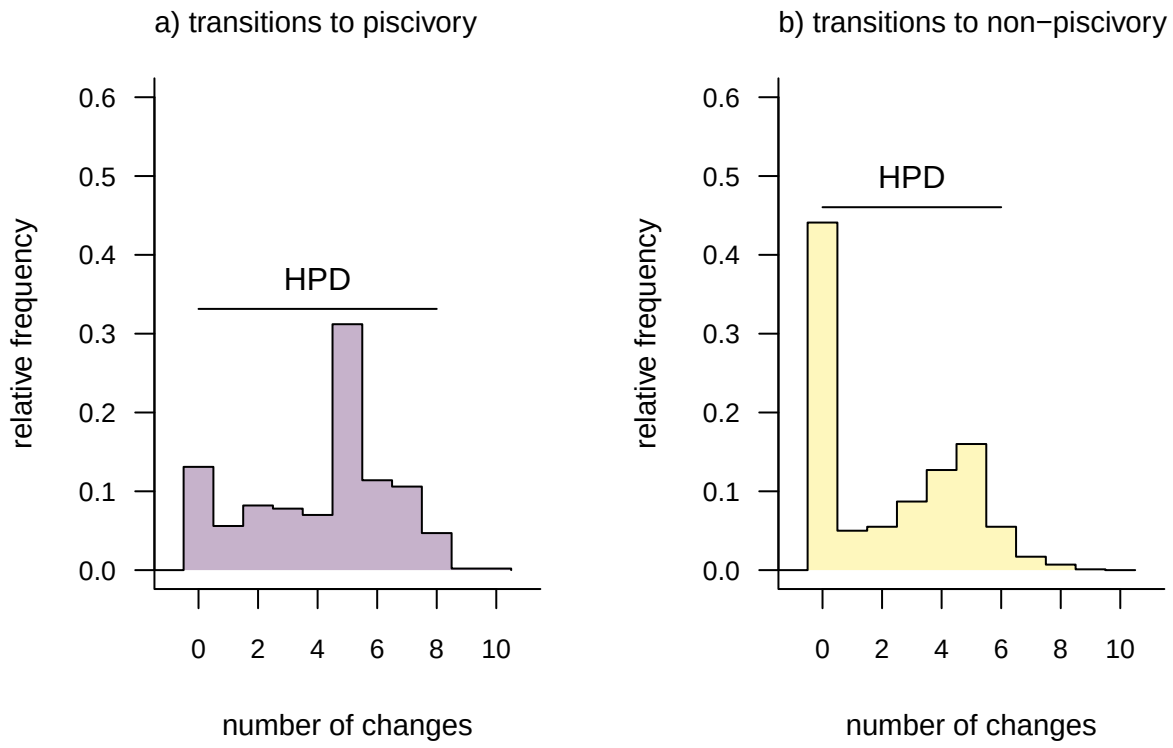

Figure 3: Posterior probability distributions of changes either (a) from non-piscivory to piscivory, or (b) from piscivory to non-piscivory, obtained from an analysis of stochastic mapping. See main text for additional details.

The distributions of Figure 3 show the relative frequencies of changes of each type across our set of mapped histories, as well as Bayesian 95% high probability density intervals from *coda*. An interesting attribute of these posterior distributions is that they are both *bi-modal*. This is due in part to our procedure of model-averaging in which we sampled both reversible and *irreversible* character evolution models in proportion

to their weights. (The weight of evidence was highly similar between our equal-rates model and the irreversible model in which piscivory is acquired from piscivory, but never the reverse.)

Lastly, in addition to these analyses, *phytools* also makes it easy to visualize the posterior probabilities of each of the two trait conditions not only at nodes, but also also along the branches of the phylogeny. This can be done using the function `densityMap` (Revell, 2013), which creates a graph showing the probability *density* of stochastic history in each of our mapped states. By design, in *phytools* this object can be first created, using the `densityMap` function, updated, using the function `setMap` to adjust the color palette for plotting, and then graphed, using a generic plot method designed for the object class.

```
sunfish.densityMap<-densityMap(sunfish.simmap,
  plot=FALSE,res=1000)
sunfish.densityMap<-setMap(sunfish.densityMap,
  viridisLite::viridis(n=10))
plot(sunfish.densityMap,lwd=3,outline=TRUE,fsize=0.8,
  legend=0.1)
```

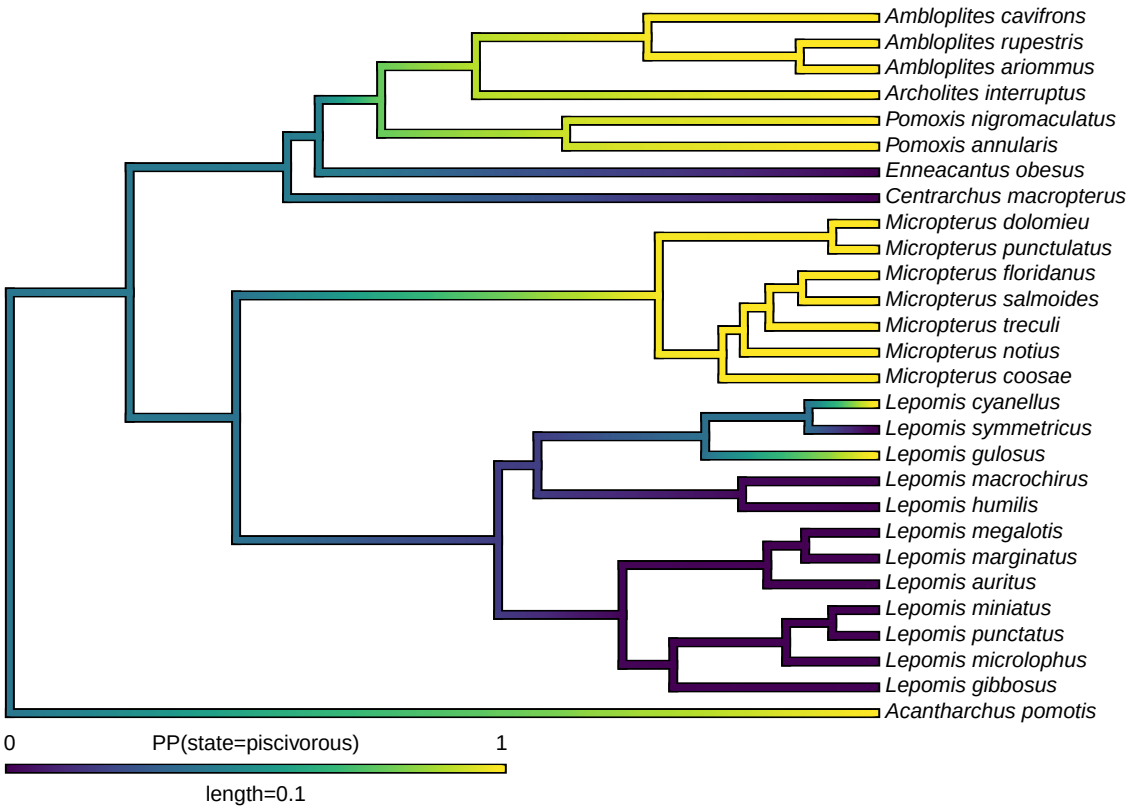

Figure 4: Posterior probability density of each of the two character levels, piscivory and non-piscivory, mapped along the edges of a tree of centrarchid fishes. See main text for more details.

The polymorphic trait evolution model

Another important, but much more recently-added, tool in the *phytools* R package is a method (denominated `fitpolyMk`) designed to fit a discrete character evolution model to trait data containing intraspecific polymorphism. In this case, the model is one in which an evolutionary transition from (say) character state a to character state b must first pass through the intermediate polymorphic condition of $a + b$. This

model becomes increasingly complicated for multi-state data in which we must also consider whether our characters are evolving in an ordered or unordered fashion. Figure 5 shows the general structure of an *ordered* polymorphic trait evolution model (in panel a) and an *unordered* model (in panel b), both for the same number of monomorphic conditions of our trait.

```
par(mfrow=c(1,2))
graph.polyMk(k=4,ordered=TRUE,states=0:3,mar=rep(0.1,4))
mtext("a) ordered polymorphic model",line=-1,adj=0.2)
graph.polyMk(k=4,ordered=FALSE,states=letters[1:4],
  mar=rep(0.1,4),spacer=0.15)
mtext("b) unordered polymorphic model",line=-1,adj=0.2)
```

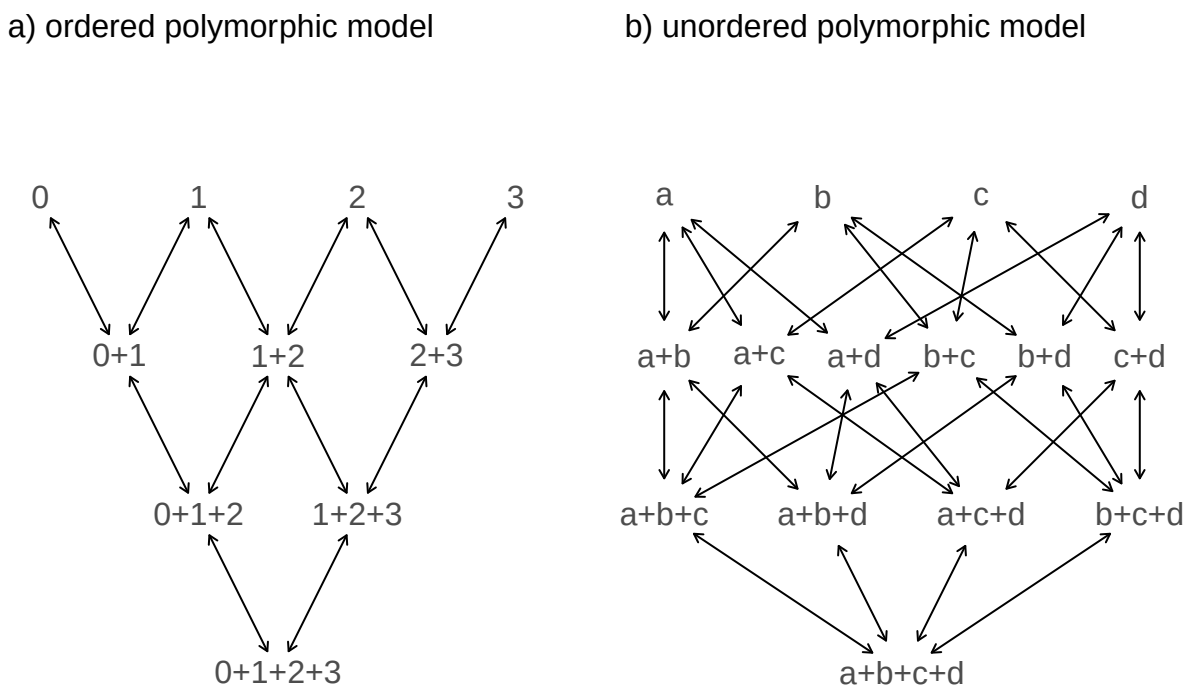

Figure 5: Example structures of two alternative polymorphic trait evolution models for characters with four monomorphic conditions: a) an ordered model; b) an unordered model. See main text for additional details.

Obviously, the potential parameter complexity of the unordered polymorphic trait evolution model is *higher* than the ordered model. Since the unordered model, has all ordered models as a special case, ordered and unordered models can also be compared using likelihood-ratio tests or information criteria.

To try out our polymorphic trait evolution model, let's use an excellent, recently-published dataset from Halali et al. (2020) consisting of a phylogenetic tree containing 287 Mycalesina butterfly species and data for habitat use. Habitat use was coded as a polymorphic trait in which, for example, a species using both "forest" and forest "fringe" habitat would be coded as `"forest+fringe"`. In this case, our polymorphic trait evolution will assume that to evolve from forest specialization to fringe specialization, a species must first (at least transiently) evolve through the polymorphic state of using both habitats at once. This seems logical.

Our data are packaged with the *phytools* library, so can be loaded using the `data` function as follows.

```
data(butterfly.tree)
data(butterfly.data)
```

Let's extract our habitat use data as a vector and print the levels that it assumes.

```
butterfly.habitat<-setNames(butterfly.data$habitat,
  rownames(butterfly.data))
print(levels(butterfly.habitat))
```

```
## [1] "forest"             "forest+fringe"      "forest+fringe+open"
## [4] "fringe"             "fringe+open"        "open"
```

Let's proceed to fit our discrete character, polymorphic trait evolution model to these data. In this instance, I'll fit a grand total of six different models. (This is not a comprehensive set of the conceivable models for polymorphic data with these levels, but it seemed like a reasonable selection for illustrative purposes.)

The first three of these models suppose that the evolution of my discrete character is totally unordered. Among these, we'll imagine, first, equal transition rates between all monomorphic states or polymorphic conditions. In our second model, we'll permit all possible transition rates between states or state combinations to assume different values. Finally, for our third model we'll assume that the acquisition of polymorphism (or its increase) occurs with one rate, whereas the loss (or decrease) of polymorphism occurs with another, separate rate. We refer to this model as the "transient model" following Revell and Harmon (2022). This comes from the general notion that if the rate of loss exceeds the rate of gain, then polymorphism will typically be *transient* in nature. Since polymorphism tends to be less frequently observed in the type of data that typifies many phylogenetic comparative study, including this model in our set seems sensible.

To get our remaining three models, for each of the three listed above in which character evolution is unordered, we'll add a second *ordered* model where we assume that character evolution for our three monomorphic conditions tends to proceed as follows: *forest* ↔ *fringe* ↔ *open*, not forgetting the intermediate polymorphic conditions in between each of these monomorphic states!

To fit these three different models in R, we can use the function `fitpolyMk` from the *phytools* package as follows.

```
butterfly.ER_unordered<-fitpolyMk(butterfly.tree,butterfly.habitat,
  model="ER")
```

```
##
## This is the design matrix of the fitted model. Does it make sense?
##
##                    forest fringe open forest+fringe forest+open fringe+open
## forest                  0      0    0             1           1           0
## fringe                  0      0    0             1           0           1
## open                    0      0    0             0           1           1
## forest+fringe           1      1    0             0           0           0
## forest+open             1      0    1             0           0           0
## fringe+open             0      1    1             0           0           0
## forest+fringe+open      0      0    0             1           1           1
##                    forest+fringe+open
## forest                              0
## fringe                              0
## open                                0
## forest+fringe                       1
## forest+open                         1
## fringe+open                         1
## forest+fringe+open                  0
```

By default, `fitpolyMk` prints out the *design matrix* of the model that it's about to fit. This is helpful,

because we should find that it corresponds with the design matrix that was discussed under the simpler Mk model of the previous section, as well as with the graphed models of Figure 5.

Nonetheless, we can turn off this message by simply setting `quiet=TRUE`. Let's do that for our remaining two unordered models.

```
butterfly.ARD_unordered<-fitpolyMk(butterfly.tree,butterfly.habitat,
  model="ARD",quiet=TRUE)
butterfly.transient_unordered<-fitpolyMk(butterfly.tree,
  butterfly.habitat,model="transient",quiet=TRUE)
```

We're not done fitting models yet, but to see how we're doing so far, we can compare our three fitted models using the generic `anova` method of their object class, as follows.

```
anova(butterfly.ER_unordered,butterfly.ARD_unordered,
  butterfly.transient_unordered)
```

```
##                                  log(L) d.f.      AIC       weight
## butterfly.ER_unordered        -355.8122    1 713.6244 5.048749e-16
## butterfly.ARD_unordered       -303.5900   18 643.1800 1.000000e+00
## butterfly.transient_unordered -353.4496    2 710.8991 1.972328e-15
```

This tells us that among the models we've seen so far, the best-supported is our parameter-rich all-rates-different (`"ARD"`) model.

Now we can proceed to do the same thing, but this time setting the argument `ordered=TRUE`.

Here we should also specify the order level using the argument `order`. If `order` isn't indicated, `fitpolyMk` will simply assume that our characters are ordered alphanumerically – but this is very rarely likely to be correct! (Readers may notice that it happens to be in this case. I set the argument `order` anyway!)

```
levs<-c("forest","fringe","open")
```

```
butterfly.ER_ordered<-fitpolyMk(butterfly.tree,butterfly.habitat,
  model="ER",ordered=TRUE,order=levs,quiet=TRUE)
butterfly.ARD_ordered<-fitpolyMk(butterfly.tree,butterfly.habitat,
  model="ARD",ordered=TRUE,order=levs,quiet=TRUE)
butterfly.transient_ordered<-fitpolyMk(butterfly.tree,
  butterfly.habitat,model="transient",ordered=TRUE,
  order=levs,quiet=TRUE)
```

Now, with all six models in hand, let's compare them using a second `anova` call as follows. This time I'll save my results to the object `butterfly.aov`.

```
butterfly.aov<-anova(butterfly.ER_ordered,butterfly.ER_unordered,
  butterfly.transient_ordered,butterfly.transient_unordered,
  butterfly.ARD_ordered,butterfly.ARD_unordered)
```

```
##                                  log(L) d.f.      AIC       weight
## butterfly.ER_ordered          -329.0390    1 660.0779 1.639410e-09
## butterfly.ER_unordered        -355.8122    1 713.6244 3.865519e-21
## butterfly.transient_ordered   -329.0205    2 662.0409 6.143598e-10
## butterfly.transient_unordered -353.4496    2 710.8991 1.510091e-20
## butterfly.ARD_ordered         -297.8100   12 619.6201 9.999923e-01
## butterfly.ARD_unordered       -303.5900   18 643.1800 7.656389e-06
```

Our model comparison shows that (among the models in our set) the best supported by far is the ordered, all-rates-different model. *phytools* has a function to graph this model, so let's proceed and do so.

```
plot(butterfly.ARD_ordered,asp=0.7,mar=rep(0.1,4))
legend("bottomleft",legend=c(
  paste("log(L) =",round(logLik(butterfly.ARD_ordered),2)),
  paste("AIC =",round(AIC(butterfly.ARD_ordered),2))),bty="n")
```

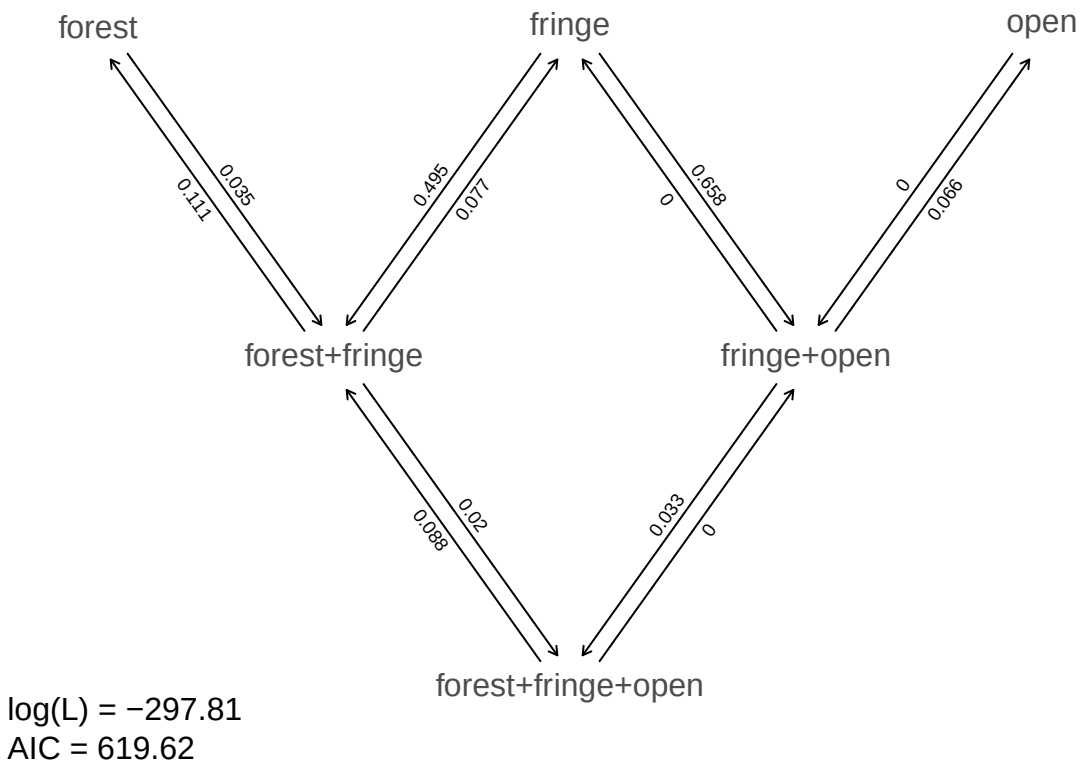

Figure 6: Best-fitting polymorphic trait evolution model for the evolution of habitat use in Mycalesina butterflies.

Just as with our fitted Mk models from the prior section, we can *also* pass this fitted model object our generic stochastic character mapping method, `simmap`. If we do, it will automatically generate a set of 100 stochastic character maps under our fitted model. (We could've likewise passed `simmap` our `anova` results, but in this case nearly all the weight of evidence fell on one model: our ordered, all-rates-different model.)

```
butterfly.simmap<-simmap(butterfly.ARD_ordered)
butterfly.simmap
```

```
## 100 phylogenetic trees with mapped discrete characters
```

Let's compute a summary of this set of stochastically mapped character histories.

```
butterfly.summary<-summary(butterfly.simmap)
```

JUst as we saw before, the object from our generic `summary` call can be plotted.

Using the *base* graphics function `rgb`, I'll try to select colors for plotting that are evenly spaced in a red-green-blue color space in which the "corners" (red, green, and blue) correspond to the three monomorphic states of our data.

```
hab.cols<-setNames(c(rgb(0,1,0),rgb(0,0.5,0.5),rgb(1/3,1/3,1/3),
    rgb(0,0,1),rgb(0.5,0.5,0),rgb(1,0,0)),levels(butterfly.habitat))
plot(butterfly.summary,type="fan",ftype="off",colors=hab.cols,
  cex=c(0.4,0.2),part=0.5,lwd=1)
legend("topleft",names(hab.cols),pch=21,pt.bg=hab.cols,pt.cex=1.5,
  cex=0.8,bty="n")
```

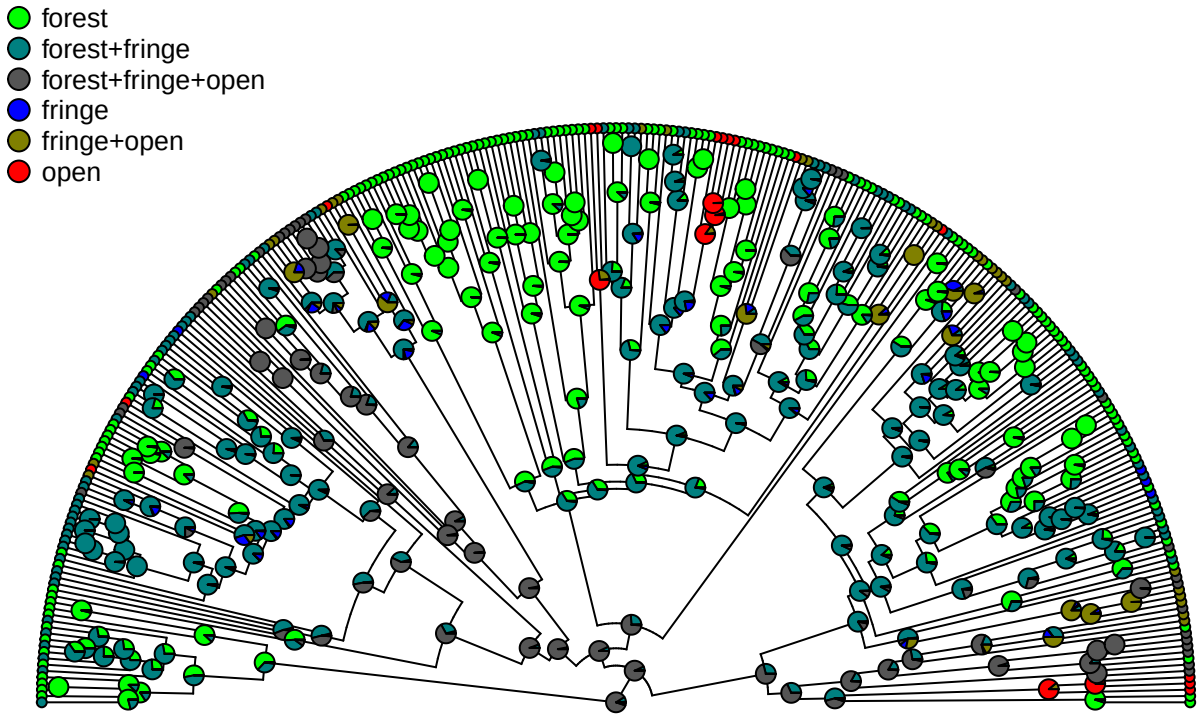

Figure 7: Posterior probabilities of monomorphic or polymorphic conditions at internal nodes from stochastic mapping under an ordered, ARD model of trait evolution. See main text for additional details.

Lastly, let's graph the posterior distribution of the *accumulation* of lineages in each state over time, using the *phytools* function `ltt` as follows. We can do that using the same color palette as in Figure 7.

```
butterfly.ltt<-ltt(butterfly.simmap)
par(mar=c(5.1,4.1,1.1,1.1))
plot(butterfly.ltt,show.total=FALSE,bty="n",las=1,cex.axis=0.8,
  colors=hab.cols)
```

Continuous characters

Numerous continuous trait methods exist in *phytools*. Perhaps the most popular, is the (somewhat dubious) measurement of phylogenetic signal (Revell et al., 2008). In addition to phylogenetic signal, *phytools* contains methodology for multivariate and multi-rate Brownian motion evolution, and for ancestral state reconstruction. Here I'll start by illustrating the measurement of phylogenetic signal, I'll show how to fit a variable-correlation multivariate Brownian trait evolution model, I'll demonstrate Bayesian ancestral state estimation, and, finally,

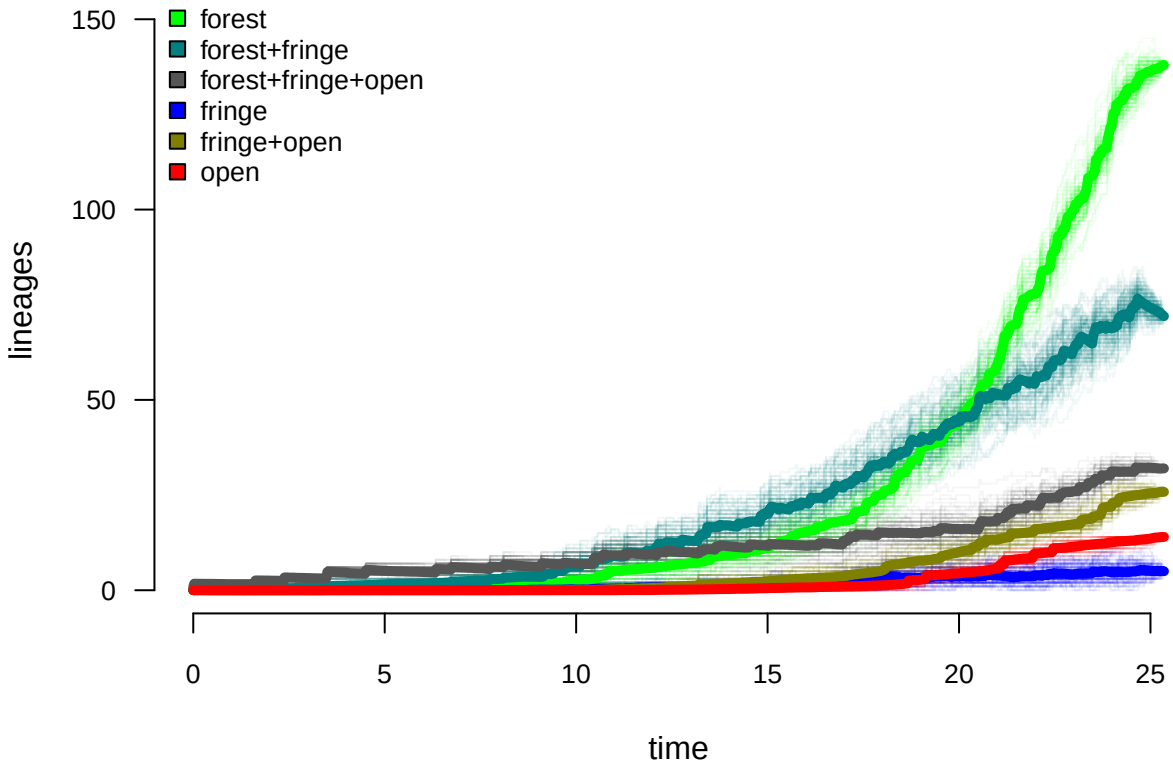

Figure 8: Lineage-through-time plot showing the reconstructed accumulation of lineages in each polymorphic condition or monomorphic state over time, from 100 stochastic character maps.

I'll show a relatively new multi-rate trait evolution model that uses the estimation technique of penalized likelihood. This is by no means comprehensive of the range of continuous character methods implemented in the *phytools* R package; however, it nonetheless provides a relatively broad sample.

Phylogenetic signal

Perhaps the *simplest* phylogenetic comparative analysis that we might choose to undertake for a continuous trait in R is the measurement of phylogenetic signal (Revell et al., 2008). Phylogenetic signal has been defined in various ways, but could be considered to be the tendency of more closely related species to bear more similarity, one to the other, than they do with more distant taxa. Phylogenetic signal can be measured in multiple ways but perhaps the two most popular metrics are Blomberg et al.'s (2003) K statistic, and Pagel's (1999) λ. Conveniently, both can be calculated using the *phytools* package.

To get started in our undertaking, let's load some data from *phytools* consisting of a phylogenetic tree of *Anolis* lizards and a data frame of morphological traits. Both derive from an article by Mahler et al. (2010).

```
data(anoletree)
data(anole.data)
```

Having loaded these data, we can next extract one variable from our data array. Phylogenetic signal can be measured for any continuous trait, but we'll use overall body size: here represented by "snout-to-vent-length" (SVL), here given on a log scale.

```
anole.bodysize<-setNames(anole.data$SVL,rownames(anole.data))
```

We can next compute our K value of Blomberg et al. (2003) using the *phytools* function `phylosig`. `phylosig` calculates K by default, but if I add the argument value `test=TRUE` it will also conduct a statistical test of the measured value of K by comparing it to a null distribution for K obtained by permuting our observed trait values randomly across the tips of the phylogeny.

```
anole.Blomberg_K<-phylosig(anoletree,anole.bodysize,test=TRUE)
anole.Blomberg_K
```

```
##
## Phylogenetic signal K : 1.69526
## P-value (based on 1000 randomizations) : 0.001
```

K has an expected value of 1.0 under Brownian motion (Blomberg et al., 2003). Lower values would thus indicate less phylogenetic signal than expected under Brownian evolution; whereas higher values indicate that phylogenetic signal is larger than expected. This result shows us that our measured phylogenetic signal of body size in *Anolis* lizards is greater than Brownian motion; however, it *does not* indicate whether it is significantly greater than Brownian motion. Our statistical test, remember, is compared to random permutations of our data vector!

phytools also can be used to estimate Pagel's (1999) λ statistic. In this case, we scalar multiply the correlation of related species by λ, and use Maximum Likelihood to find the value of λ that makes our data most probable. Since we can also compute a likelihood for $\lambda = 0$, we can test for *significant* phylogenetic signal by computing a likelihood ratio.

```
anole.Pagel_lambda<-phylosig(anoletree,anole.bodysize,
  method="lambda",test=TRUE)
anole.Pagel_lambda
```

```
##
## Phylogenetic signal lambda : 0.999934
## logL(lambda) : 5.25344
## LR(lambda=0) : 97.8981
## P-value (based on LR test) : 4.40483e-23
```

Unsurprisingly, this tells us that we've found significant phylogenetic signal in our trait by both measures!

phytools also contains methods to visualize our phylogenetic signal analyses. In particular, for Blomberg et al.'s K we can plot the permutation distribution of K as well as our observed measure; whereas for Pagel's λ we can plot the likelihood surface and our measure solution. Both of these plots are illustrated in Figure @ref{phylosig} for our *Anolis* body size data.

```
par(mfrow=c(1,2),cex=0.9)
plot(anole.Blomberg_K,las=1,cex.axis=0.9)
mtext("a)",adj=0,line=1)
plot(anole.Pagel_lambda,bty="n",las=1,
  cex.axis=0.9,xlim=c(0,1.1))
mtext("b)",adj=0,line=1)
```

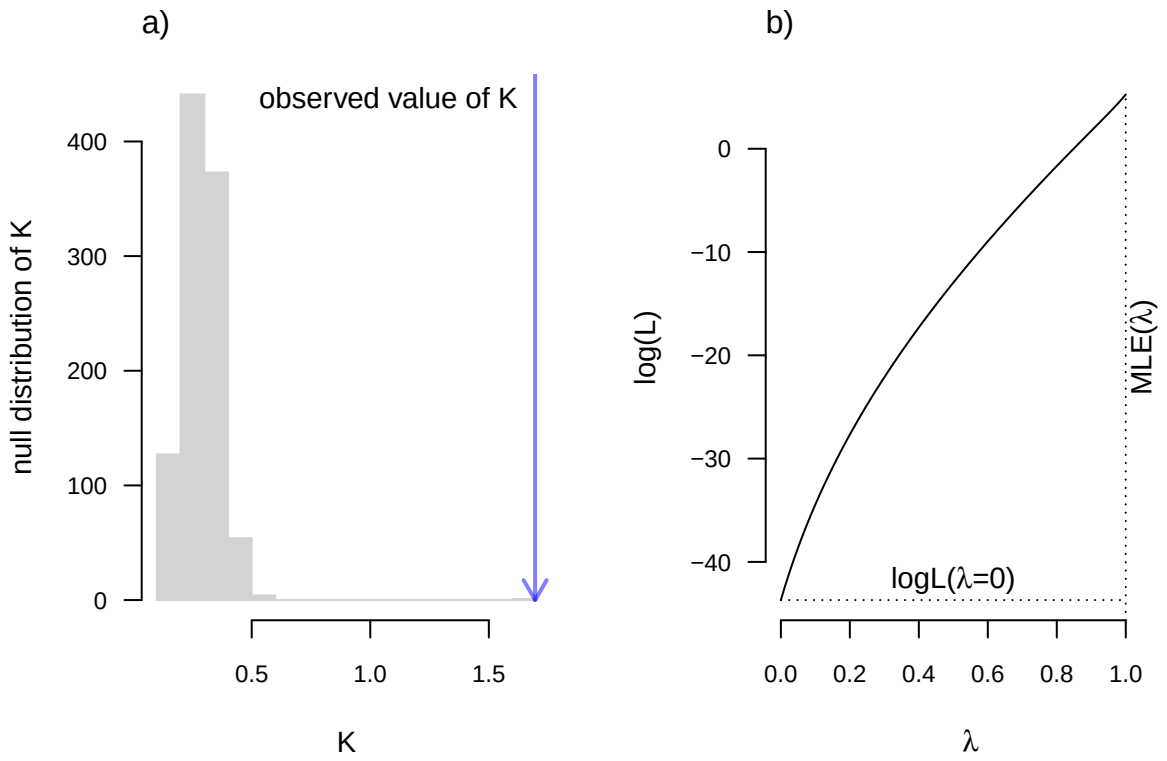

Figure 9: a) Blomberg et al. (2003) measured value of the K statistic for phylogenetic signal, compared to a null distribution of K obtained via randomization. b) Pagel's (1999) λ statistic for phylogenetic signal, also showing the likelihood surface.

Bayesian ancestral state estimation

phytools contains multiple functions for discrete and continuous character ancestral state estimation. One such method is the function `anc.Bayes` which, as its name would suggest, performs ancestral state estimation using Bayesian MCMC. The implementation of this method allows us to include prior information about the states at internal nodes.

To illustrate `anc.Bayes` I'll load a *phytools* dataset consisting of a phylogeny of cordylid lizards.

```
data(cordylid.tree)
data(cordylid.data)
```

```
cordylid.pc1<-setNames(cordylid.data$pPC1,rownames(cordylid.data))
```

```
cordylid.mcmc<-anc.Bayes(cordylid.tree,cordylid.pc1,
  ngen=100000)
```

```
## Control parameters (set by user or default):
```

```
## List of 7
##  $ sig2   : num 0.713
##  $ a      : num [1, 1] 0.000422
##  $ y      : num [1:26] 0.000422 0.000422 0.000422 0.000422 0.000422 ...
##  $ pr.mean: num [1:28] 1000 0 0 0 0 0 0 0 0 0 ...
##  $ pr.var : num [1:28] 1e+06 1e+03 1e+03 1e+03 1e+03 1e+03 1e+03 1e+03 1e+03 1e+03 ...
##  $ prop   : num [1:28] 0.00713 0.00713 0.00713 0.00713 0.00713 ...
##  $ sample : num 100
```

```
## Starting MCMC...
```

```
## Done MCMC.
```

```
cordylid.ace<-summary(cordylid.mcmc)
```

```
##
## Object of class "anc.Bayes" consisting of a posterior
##    sample from a Bayesian ancestral state analysis:
##
## Mean ancestral states from posterior distribution:
##        29        30        31        32        33        34        35        36
##  0.167331 -0.054595 -0.086632  0.042983  0.095234  0.111434  0.132066  0.217701
##        37        38        39        40        41        42        43        44
##  0.421489  0.504685 -0.066590 -0.065794  0.418919  0.393942  0.333837  0.182598
##        45        46        47        48        49        50        51        52
## -1.419904 -1.826914 -0.089010 -0.482568 -0.842082 -1.018801 -1.068330  0.327760
##        53        54        55
##  0.444237  0.228196  0.090203
##
## Based on a burn-in of 20000 generations.
```

```
plot(setMap(contMap(cordylid.tree,cordylid.pc1,
  anc.states=cordylid.ace,plot=FALSE),
  viridisLite::viridis(n=10)),ftype="i",
  leg.txt="PC 1",fsize=c(0.7,0.8))
nodelabels(frame="circle",bg="white",cex=0.7)
```

```
plot(density(cordylid.mcmc,what=49),las=1,bty="n",
  main="",cex.lab=0.9,cex.axis=0.8)
```

Variable rate Brownian motion

```
data(mammal.tree)
data(mammal.data)
```

```
mammal.lnMass<-setNames(log(mammal.data$bodyMass),rownames(mammal.data))
```

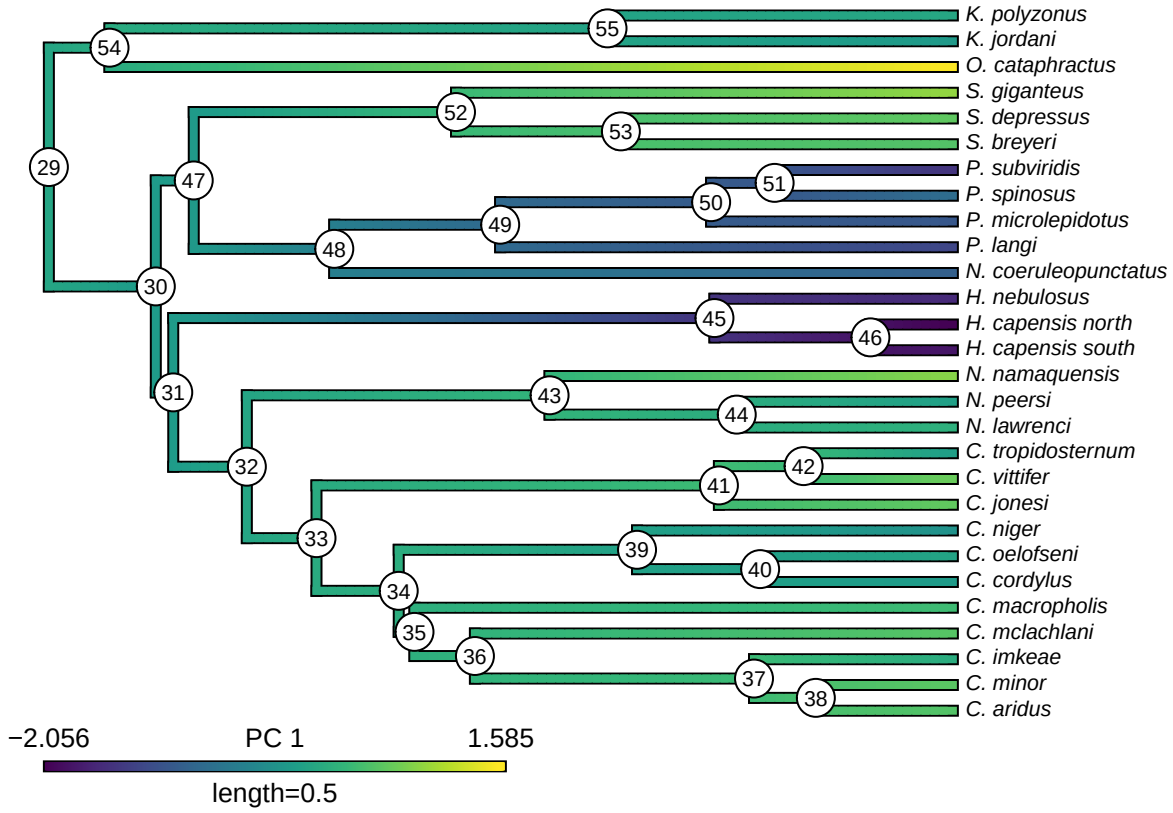

Figure 10: Reconstructed ancestral values from Bayesian MCMC projected onto the nodes and edges of the tree.

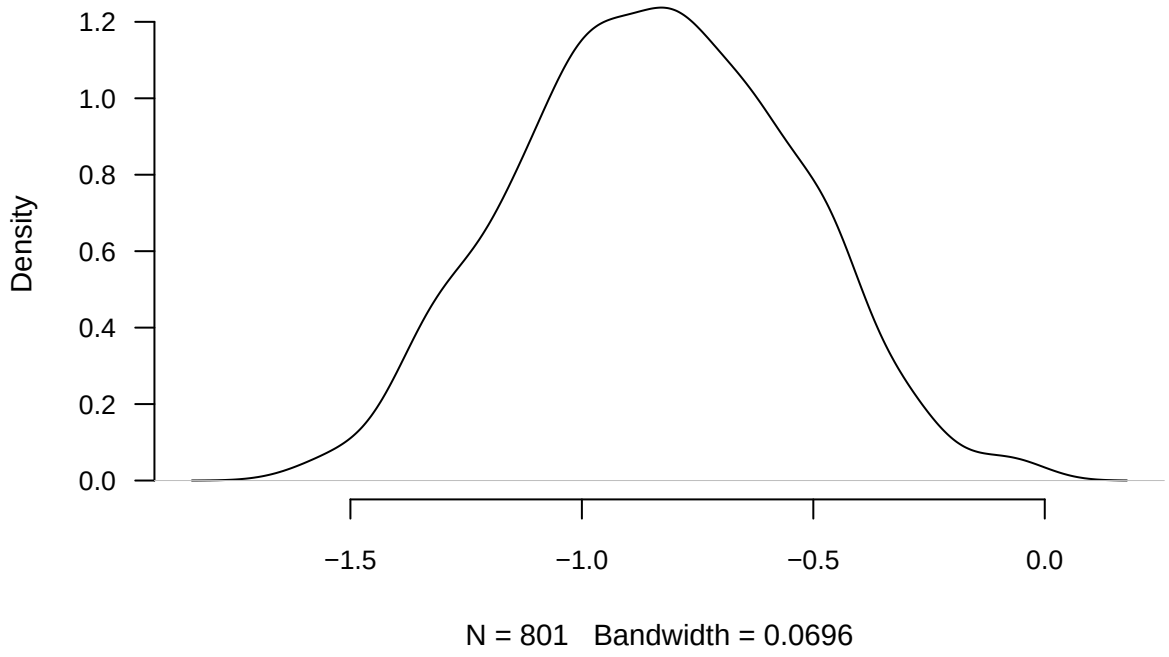

Figure 11: Posterior probability density at node 49 of Figure 10from Bayesian MCMC ancestral state reconstruction of PC 1 from a morphological analysis on a phylogenetic tree of cordylid lizards.

```
mammal.multirateBM<-multirateBM(mammal.tree,mammal.lnMass,lamba=1,
  parallel=TRUE)
```

```
## Beginning optimization....
## Using socket cluster with 16 nodes on host 'localhost'.
## Optimization iteration 1. Using "L-BFGS-B" (parallel) optimization method.
## Best (penalized) log-likelihood so far: -230.967
## Done optimization.
```

```
plot(mammal.multirateBM,fsize=0.6,lwd=4,outline=TRUE,ftype="i")
```

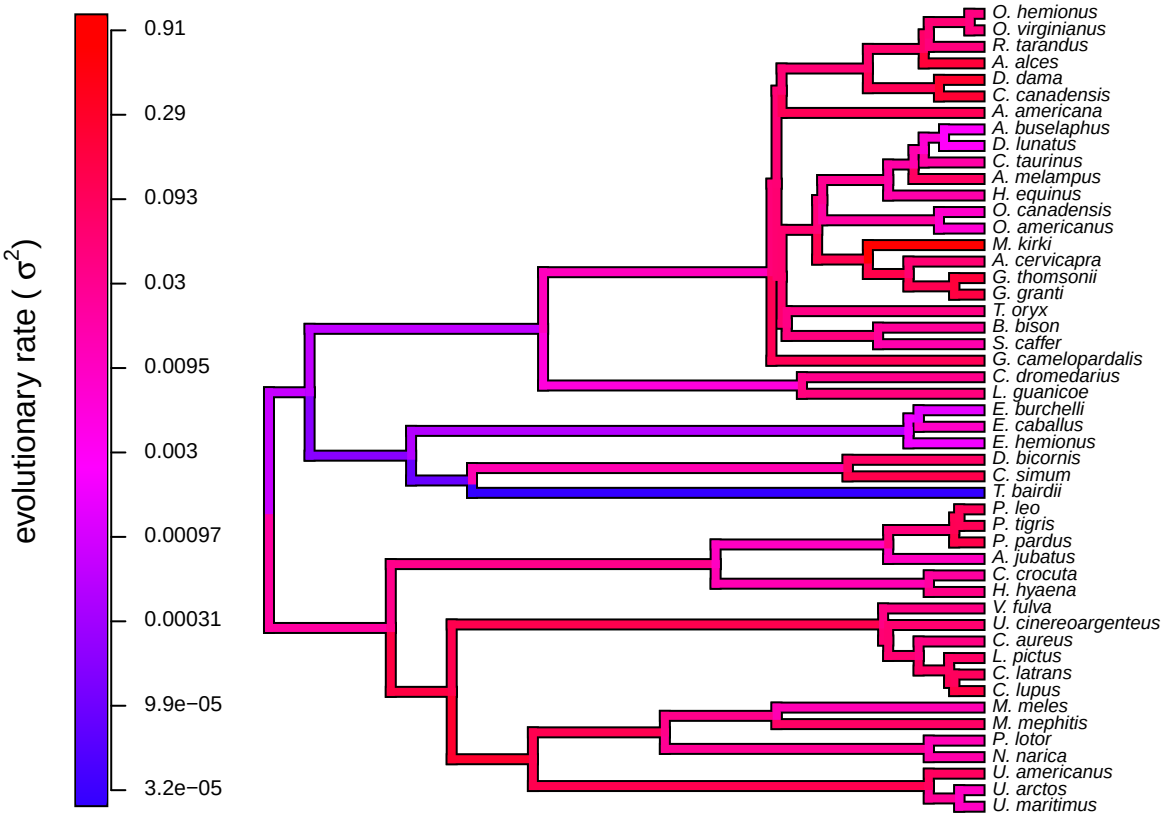

Figure 12: Estimated rates of log(body mass) evolution under a variable-rate Brownian evolution model.

Diversification

Visualization