corHMM 2.1: Generalized hidden Markov models

James D. Boyko and Jeremy M. Beaulieu

The vignette is comprised of three sections, where we demonstrate all new extensions as well as other new and useful features:

- **Introduction Some background information**
- **Section 1 Default use of corHMM**
  - 1.1: No hidden rate categores
  - 1.2: Any number of hidden rate categories
- **Section 2 How to make and interpret custom models**
  - 2.1: Creating and using custom rate matrices
    - 2.1.1: One rate category
    - 2.1.2: Any number of rate categories
  - 2.2: Some examples of "biologically informed" models
    - 2.2.1: Ordered habitat change
    - 2.2.2: Precursor model
    - 2.2.3: Ontological relationship of multiple characters
- **Section 3 Estimating models when node states are fixed**
  - 3.1: Fixing a single node
  - 3.2: Estimating rates under a parsimony reconstruction
  - 3.3: Fixing nodes when the model contains hidden states

Introduction

The original version of `corHMM` contained a number of distinct functions for conducting analyses of discrete morphological characters. This included the `corHMM()` function for fitting a hidden rates model, which uses "hidden" states as a means of allowing transition rates in a binary character to vary across a tree. In reality, the hidden rates model falls within a general class of models, hidden Markov models (HMM), that may also be applied to multistate characters. So, whether the focal trait is binary or contains multiple states, or whether the observed states represents a set of binary and multistate characters, hidden states can be applied as a means of allowing heterogeneity in the transition model. Choosing a model specific to your question is of utmost importance in any comparative method, and in this new version of `corHMM` we provide users with the tools to create their own hidden Markov models.

Before delving into this further it may be worth showing a little of what is underneath the hood. To begin, consider a single binary character with states *0* and *1*. If the question was to understand the asymmetry in the transition between these two states, the model, $\mathbf{Q}$, would be a simple 2x2 matrix,

$$Q = \begin{bmatrix} - & q_{0\to1} \\ q_{1\to0} & - \end{bmatrix}$$

This *transition rate matrix* is read as describing the transition rate *from* ROW *to* COLUMN. Thus, there are only two states, 0 and 1, and two transitions going from $0 \to 1$, and from $1 \to 0$. However, if we introduce a second binary character, the number of possible states you *could* observe is expanded to account for all the combination of states between two characters – that is, you could observe *00*, *01*, *10*, or *11*. To accommodate this, we need to expand our model such that it becomes a 4x4 matrix,

$$Q = \begin{bmatrix} - & q_{00\to01} & q_{00\to10} & q_{00\to11} \\ q_{01\to00} & - & q_{01\to10} & q_{01\to11} \\ q_{10\to00} & q_{10\to01} & - & q_{10\to11} \\ q_{11\to00} & q_{11\to01} & q_{11\to10} & - \end{bmatrix}$$

This model is considerably more complex, as the number of transitions in this rate matrix now goes from 2 to 12. However, with these models we often make a simplifying assumption that we do not allow for transitions in two states to occur at the same time. In other words, if a lineage is in state *00* it must first transition to either state *01* or *10*, before transitioning to state *11*. Therefore, we can simplify the matrix by removing these "dual" transitions from the model completely,

$$Q = \begin{bmatrix} - & q_{00\to01} & q_{00\to10} & - \\ q_{01\to00} & - & - & q_{01\to11} \\ q_{10\to00} & - & - & q_{10\to11} \\ - & q_{11\to01} & q_{11\to10} & - \end{bmatrix}$$

What we just described is the popular model of Pagel (1994), which tests for correlated evolution between two binary characters. But, one thing that is not obvious: the states in the model need not be represented as combinations of binary characters. For example, the focal character may be two characters, like say, flowers that are red with and without petals, and blue flowers with and without petals. One could just code it as a single multistate character, where *1*=red petals, *2*=red with no petals (i.e., sepals are red), *3*=blue petals, and *4*=blue with no petals (i.e., sepals are blue). The model would then be,

$$Q = \begin{bmatrix} - & q_{1\to2} & q_{1\to3} & q_{1\to4} \\ q_{2\to1} & - & q_{2\to3} & q_{2\to4} \\ q_{3\to1} & q_{3\to2} & - & q_{3\to4} \\ q_{4\to1} & q_{4\to2} & q_{4\to3} & - \end{bmatrix}$$

Notice it is the same as before, but the states are transformed from binary combinations to a multistate character. As before, we may assume that transitions in two states cannot occur at the same time and remove the "dual" transitions.

$$Q = \begin{bmatrix} - & q_{1\to2} & q_{1\to3} & - \\ q_{2\to1} & - & - & q_{2\to4} \\ q_{3\to1} & - & - & q_{3\to4} \\ - & q_{4\to2} & q_{4\to3} & - \end{bmatrix}$$

Again, exactly the same.

The updated version of `corHMM()` now lets users transform a set of characters into a *single* multistate character. This means that two characters need not have the same number of character states – that is, one trait could have four states, and the other could be binary. We also allow any model to be expanded to accomodate an arbitrary number of hidden states. Thus, `corHMM()` is completely flexible and naturally contains `rayDISC()` and `corDISC()` capabilities - standalone functions in previous versions of `corHMM` that may have been mistaken as different "methods." As this vignette will show, they are indeed nested within a broader class of HMMs.

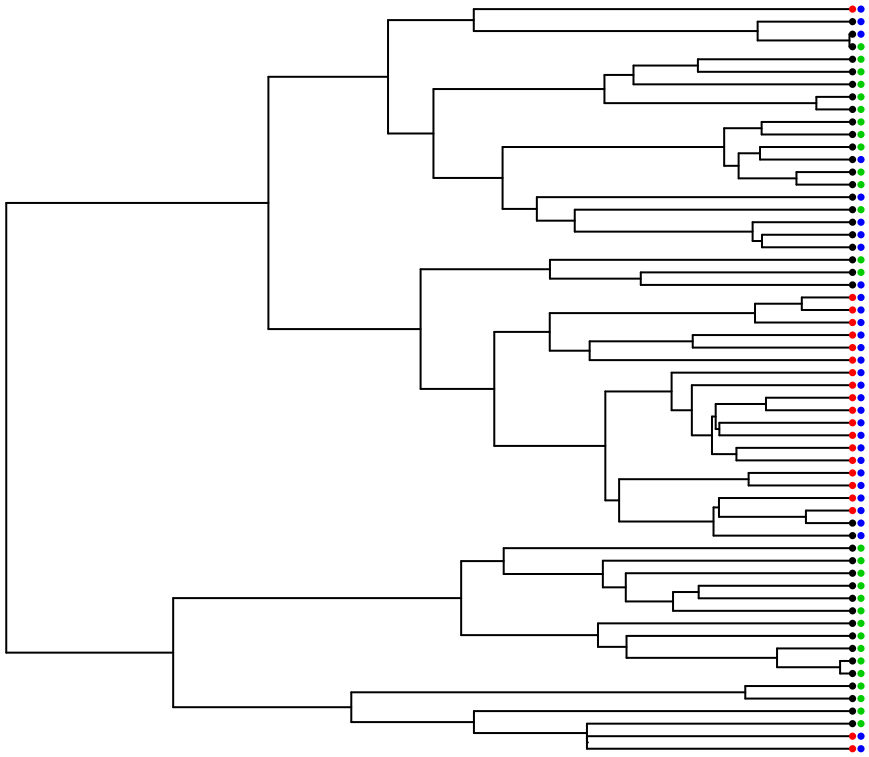
Section 1: Default use of corHMM

1.1: No hidden rate categories

To start, we'll use the primate dataset from Pagel and Meade (2006) that comes with `corHMM`:

```
set.seed(1985)
require(ape)
require(expm)
require(corHMM)
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
```

```
plot(phy, show.tip.label = FALSE)
data.sort <- data.frame(data[, 2], data[, 3], row.names = data[, 1])
data.sort <- data.sort[phy$tip.label, ]
tiplabels(pch = 16, col = data.sort[, 1] + 1, cex = 0.5)
tiplabels(pch = 16, col = data.sort[, 2] + 3, cex = 0.5, offset = 0.5)
```



We have two characters each with two possible states: trait 1 is the absence (black) or presence (red) of estrus advertisement in females, and trait 2 is single male (green) or multimale (blue) mating system in primates.

The default use of `corHMM()` only requires that you declare your *phylogeny*, your *dataset*, and the number of *rate categories* (more detail about this later). We have updated `corHMM()` to handle different types of input data. Now to use `corHMM()`, the first column must be species names (as in the previous version), but there can be any number of data columns. If your dataset does have 2 or more columns of trait information, each column is taken to describe a separate character. Note that when the `corHMM()` call is used, the function automatically determines all the unique character combinations *observed* in the data set. In our primate example only 3 of the 4 possible combinations are observed, and so the model is constructed accordingly. Also, dual transitions are automatically disallowed. In other words, we expect that a species cannot go directly from estrus advertisement being absent in a single male mating system to having estrus advertisement in a multimale mating system. They must first evolve either estrus advertisement or multimale mating system.

Let's give this a try:

```
MK_3state <- corHMM(phy = phy, data = data, rate.cat = 1)
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
MK_3state
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -41.90867 91.81734 92.54461        1   60
##
## Rates
##            (1,R1)     (2,R1)    (3,R1)
## (1,R1)         NA 0.01900010        NA
## (2,R1) 0.05664305         NA 0.0262821
## (3,R1)         NA 0.01610568        NA
##
## Arrived at a reliable solution
```

When you run your `corHMM` object you are greeted with a summary of the model. Your model fit is described by the log likelihood (lnL), Akaike information criterion (AIC), and sample size corrected Akaike information criterion (AICc). You are also given the number of rate categories (Rate.cat) and number of taxa (ntax).

The *Rates* section of the output describes transition rates between states and are organized as a matrix. Again, the *transition rate matrix* is read as the transition rate **from** ROW **to** COLUMN. For example, if you were interested in the transition rate from State 1 (i.e., absence of estrus advertisement in a single male mating system) to State 2 (i.e., absence of estrus advertisement in a multimale mating system) you would be looking at the Row 1, Column 2, entry. For a time calibrated ultrametric tree, these rates will depend on the age of your phylogeny.

You may also notice that `corHMM()` printed a state legend to the screen. Thus, you can obtain the exact coding for each species in an augmented dataframe provided by the `corHMM()` results object itself. This dataframe uses the initial user data to create columns that corresponds to how each species was represented in `corHMM()`:

```
head(MK_3state$data.legend)
```

```
##                  Genus_sp T1 T2 legend
## 1    Cercocebus_torquatus  1  1      3
## 2  Cercopithecus_aethiops  0  1      2
## 3      Cercopithecus_mona  0  0      1
## 4 Cercopithecus_nictitans  0  0      1
## 5      Colobus_angolensis  0  1      2
## 6         Colobus_guereza  0  0      1
```

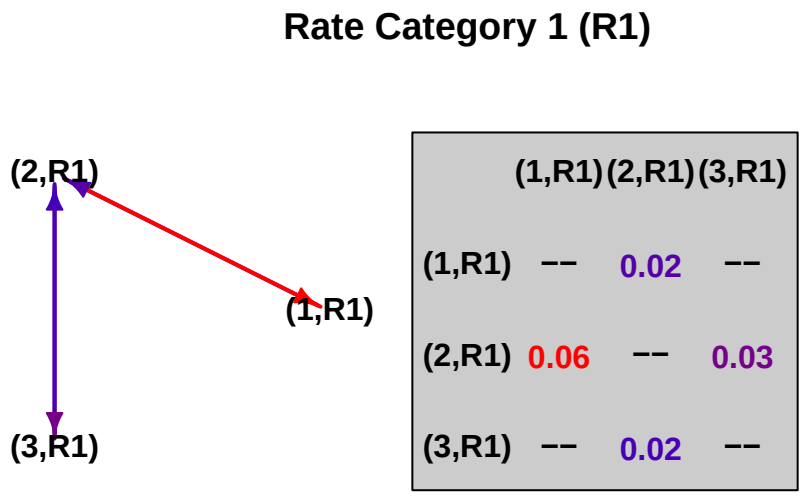
Alternatively, a user can supply their dataset to getStateMat4Dat, which outputs a legend that is consistent with the `corHMM()` function. The other output is an index matrix (or rate matrix) that describes which rates are to be estimated in `corHMM()`. We provide an in-depth discussion about this part of the index matrix later:

```
getStateMat4Dat(data)
```

```
## $legend
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   4
## (3)   0   3   0
```

Finally, interpreting a Markov matrix can be difficult, especially when you're just starting out. This problem is compounded when users begin to apply the more complex hidden Markov models (i.e. setting rate.cat > 1). To help users, we have implemented a new plotting function:

```
plotMKmodel(MK_3state)
```



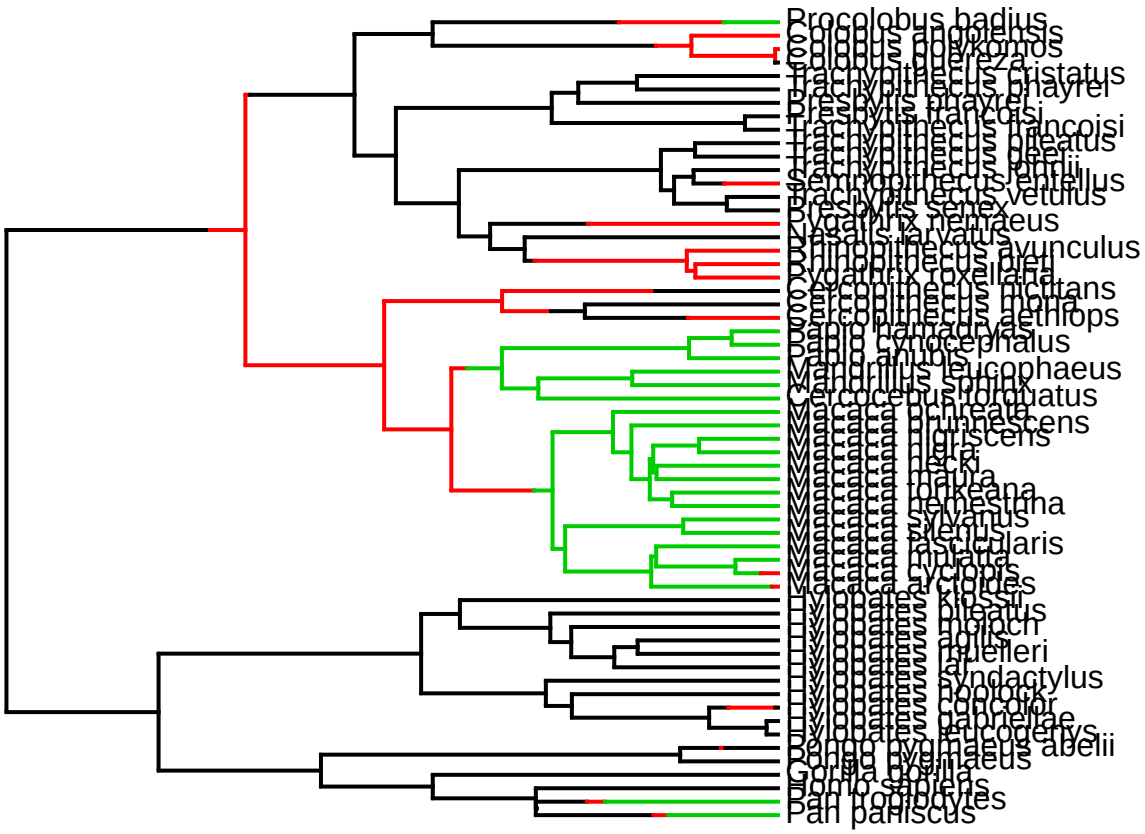
This function uses a `corHMM` object (which is the result of running `corHMM()`) or a custom rate matrix (discussed in a later section) to plot the model in two parts. On the left is a ball and stick diagram that depicts the state transitions. On the right is a simplified rate matrix that contains rounded values from the solution output of `corHMM()`. The colors of the arrows correspond to the magnitude of the rates.

The final new plotting tool we have made available to users is a stochastic character mapping function, `makeSimmap` (Bollback, 2006). We can use `makeSimmap` to create a character history for any `corHMM` model and then use plotSimmap (from the popular R-package, `phytools`) to plot the output.

```
phy = MK_3state$phy
data = MK_3state$data
model = MK_3state$solution
model[is.na(model)] <- 0
diag(model) <- -rowSums(model)
states = MK_3state$states
tip.states = MK_3state$tip.states
# run get simmap (can be plotted using phytools)
simmap <- makeSimmap(tree = phy, tip.states = tip.states, states = states, model = model,
    nSim = 1, nCores = 1)
```

```
# we import phytools plotSimmap for plotting
phytools::plotSimmap(simmap[[1]])
```

```
## no colors provided. using the following legend:
##        1        2        3
##  "black"    "red" "green3"
```



1.2: A trait with any number of states and any number of hidden rate categories

The major difference between this version of `corHMM` and previous versions is allowing models of any number of states and any number of hidden rate categories (*hidden rate categories will be explained in more depth in section 2*). Running a hidden Markov model (HMM) only requires assigning a value greater than 1 to the rate.cat input. Below, we have assigned 2 rate categories to the data from above:

```
HMM_3state <- corHMM(phy = phy, data = data, rate.cat = 2, model = "SYM", get.tip.states = TRUE)
```

```
## You specified 'fixed.nodes=FALSE' but included a phy object with node labels. These node labels have
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
```

```
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

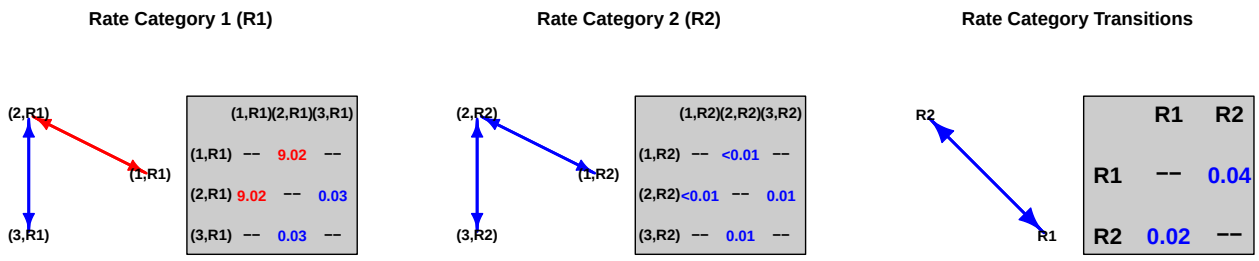
```
HMM_3state
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -41.54999 95.09998 96.68489        2   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)      (1,R2)      (2,R2)     (3,R2)
## (1,R1)         NA 9.01973679         NA 0.038573708          NA         NA
## (2,R1) 9.01973679         NA 0.02982121          NA 0.038573708         NA
## (3,R1)         NA 0.02982121         NA          NA          NA 0.03857371
## (1,R2) 0.01733009         NA         NA          NA 0.000000001         NA
## (2,R2)         NA 0.01733009         NA 0.000000001          NA 0.01204654
## (3,R2)         NA         NA 0.01733009          NA 0.012046545         NA
##
## Arrived at a reliable solution
```

Models with more states (larger state space) take longer to estimate because the number of transition rates increases. Hidden rate models further expand state space. For example, adding a second rate category incerases the number of transition rates from 4 to 10 (if the model is left as the default "ARD"). In section 1.1 we left our parameters unconstrained. We estimated all transisions as independent and allowed for transitions from all states to any other state. However, we can constrain a model in `corHMM` in two different ways. The easiest way is to set the model to either "SYM" or "ER". This is what we've done for the HMM_3state model above. By setting model = "SYM", we have forced the transition rates between any two states to be equal. In comparison, model = "ER" constrains all transition rates between states to be the same. Finally, model = "ARD" (the default) allows all transition rates to be independently estimated. Although "ER" and "SYM are common restrictions, it is often more useful to manually restrict your model to match a biological hypothesis (which is described in the next section). Finally, we set get.tip.states to be true because it is necessary for simmaps.

Interpreting the estimated rate matrix for this hidden Markov model is intimidating. But, the same principles of interpreting the transition rate matrices apply – that is, you still read rates from row to column. However, there is the added complexity of transitions among the different rate categories (as represented by R1 and R2). `plotMKmodel()` will plot the underlying structure of model in discrete parts. In the following example, the first 2 panels show how observed states transition within each rate category, and the last panel shows transitions among the different rate classes:

```
plotMKmodel(HMM_3state, display = "row")
```



And again we can plot the simmap of this `corHMM` result. It is important to note that a character history not only generates hypotheses about ancestral states, but is an effective way to visualize the tempo of evolution. This is particularly important for HMMs where rates of evolution can vary drastically across the tree.

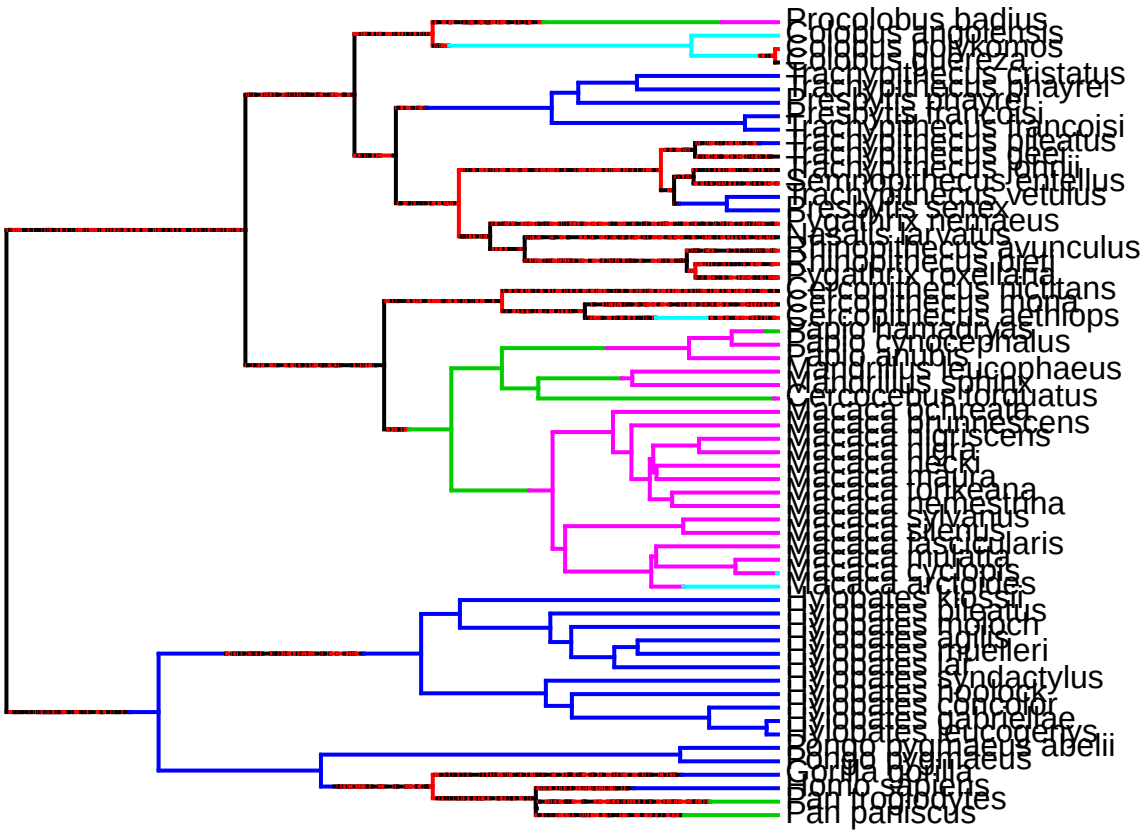
```
# get simmap inputs from corhmm outputs
phy = HMM_3state$phy
```

```
data = HMM_3state$data
model = HMM_3state$solution
model[is.na(model)] <- 0
diag(model) <- -rowSums(model)
states = HMM_3state$states
tip.states = HMM_3state$tip.states

# run get simmap (can be plotted using phytools)
simmap <- makeSimmap(tree = phy, tip.states = tip.states, states = states, model = model,
    nSim = 1, nCores = 1)

# we import phytools plotSimmap for plotting
phytools::plotSimmap(simmap[[1]])
```

```
## no colors provided. using the following legend:
##         1         2         3         4         5         6
##   "black"     "red"  "green3"    "blue"    "cyan" "magenta"
```



Section 2: How to make and interpret custom models

2.1: Creating and using custom rate matrices

2.1.1: One rate category

At its core, the purpose of a rate matrix (i.e., `rate.mat`) is to indicate to `corHMM` which parameters are being estimated. It specifies to `corHMM()` which rates in the matrix are being estimated and if any of them are expected to be identical.

A custom rate matrix allows you to specify explicit hypotheses. For example, such an approach allows for tests of evolution of traits in a particular order, tests of different rates of evolution in different clades, or tests of the presence of hidden precursors before a state can evolve.

Let's start by using the `getStateMat4Dat()` function to get a generic `rate.mat` object:

```
LegendAndRateMat <- getStateMat4Dat(data)
RateMat <- LegendAndRateMat$rate.mat
RateMat
```

```
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   4
## (3)   0   3   0
```

The numbers in this matrix are not rates, they are used to index the unique parameters to be estimated by `corHMM()`. Each distinct number is a parameter to be estimated independently from all others. Let's manually create the symmetric model we used in secion 1.2. In the symmetric model we want transitions *to* a state to be the same as *from* that state. This means that (1) → (2) & (2) → (1) are equal AND that (3) →(2) and (2) → (3) are equal. In other words, based on the `rate.mat` above, we want parameters 1 & 2 to be equal and we want parameters 3 & 4 to be equal as shown below:

```
pars2equal <- list(c(1, 2), c(3, 4))
StateMatA_constrained <- equateStateMatPars(RateMat, pars2equal)
StateMatA_constrained
```

```
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   2
## (3)   0   2   0
```

To manually create a symmetric model, we used the equateStateMatPars() function, in which the first argument is the rate matrix being modified (i.e., rate.mat object) and second argument is list of the parameters to be equated. One thing to note is that you must have the appropriate number of rate categories since a user rate matrix is not duplicated or changed by `corHMM()`. Thus, this custom model can only be used if we set rate.cat=1 since that is the appropriate number of rate categories. We can now provide this customized `rate.mat` to `corHMM()`:

```
MK_3state_customSYM <- corHMM(phy = phy, data = data, rate.cat = 1, rate.mat = StateMatA_constrained)
```

```
## You specified 'fixed.nodes=FALSE' but included a phy object with node labels. These node labels have
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
```

```
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
MK_3state_customSYM
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -44.36714 92.73429 92.94482        1   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)
## (1,R1)         NA 0.02569184         NA
## (2,R1) 0.02569184         NA 0.01969303
## (3,R1)         NA 0.01969303         NA
##
## Arrived at a reliable solution
```

2.1.2: Any number of rate categories

From a technical standpoint, hidden Markov models have a hierarchical structure that can be broken down into two components: a "state-dependent process" and an unobserved "parameter process" (Zucchini et al. 2017). In comparative biology, the standard "state-dependent process" model is a continuous-time Markov chain. The observed states could be any discretized trait such as presence or absence of extrafloral nectaries (Marazzi et al. 2012), woody or herbaceous growth habit (Beaulieu et al. 2013), or diet state across all animals (Roman-Palacios et al. 2019). However, a simple Markov process alone that assumes homogeneity through time and across taxa is often not adequate to capture the variation of real datasets (e.g. Beaulieu et al. 2013). One option is to say that the observed data is the product of several processes occurring in different parts of a phylogeny. The parameter process describes how several state-dependent processes relate to one another. Thus, observations are generated by a given state-dependent process depending on the state of the parameter process. It is initially unknown what the parameter process corresponds to biologically, hence the moniker "hidden" state.

If you wanted to add hidden rate categories, you need to know: (1) the dynamics *within* each rate category (state-dependent processes), and (2) transitions *between* the different rate classes (parameter process). We begin by constructing two *within* rate category `rate.mat` objects (R1 and R2). In R1, we assume a drift-like hypothesis where all transition rates are equal. In R2, we assume that the combination of estrus advertisement and multimale mating systems are not lost once they evolve:

```
RateCat1 <- getStateMat4Dat(data)$rate.mat  # R1
RateCat1 <- equateStateMatPars(RateCat1, c(1:4))
RateCat1
```

```
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   1
## (3)   0   1   0
```

```
RateCat2 <- getStateMat4Dat(data)$rate.mat  # R2
RateCat2 <- dropStateMatPars(RateCat2, 3)
RateCat2
```

```
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   3
## (3)   0   0   0
```

With respect to transitions *among* the different rate classes, we have implemented a separate matrix generator, `getRateCatMat()`. By default, this function will assume that all transitions among the specified number of rate classes occur independently. In our example, we will generate a matrix that specifies how transitions between R1 and R2 occur. Note that R1 and R2 could represent a biologically-relevant, but unmeasured factor, such as, say, temperate or tropical environments, island or mainland, presence or absence of a third trait. Basically, it is everything and anything that can influence the evolution of your observed characters.

For illustrative purposes, we will specify that the transition rate from R1 to R2 is the same as the rate from R2 to R1:

```
RateClassMat <- getRateCatMat(2)  #
RateClassMat <- equateStateMatPars(RateClassMat, c(1, 2))
RateClassMat
```

```
##    R1 R2
## R1  0  1
## R2  1  0
```

We now group all of our rate classes together in a list. The first element of the list corresponds to R1, the second to R2, etc.

```
StateMats <- list(RateCat1, RateCat2)
StateMats
```

```
## [[1]]
##     (1) (2) (3)
## (1)   0   1   0
## (2)   1   0   1
## (3)   0   1   0
##
## [[2]]
##     (1) (2) (3)
## (1)   0   2   0
## (2)   1   0   3
## (3)   0   0   0
```

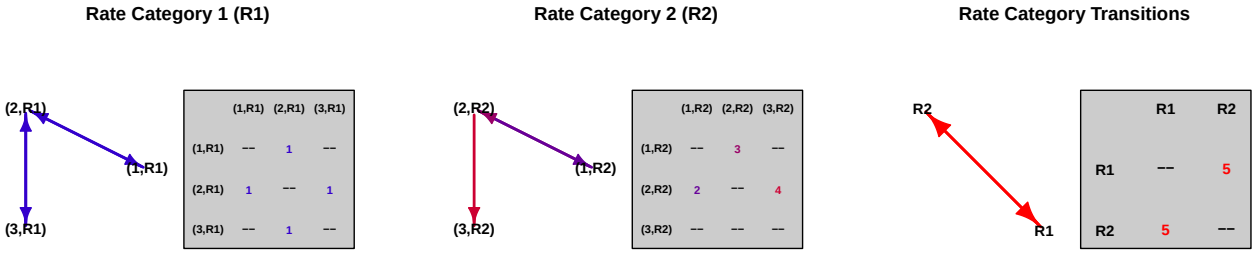
We now have all the components necessary to create the full model using the `getFullMat()` function. This function requires that the first input be a list of the within rate class matrices and the second argument be the among rate class matrices:

```
FullMat <- getFullMat(StateMats, RateClassMat)
FullMat
```

```
##        (1,R1) (2,R1) (3,R1) (1,R2) (2,R2) (3,R2)
## (1,R1)      0      1      0      5      0      0
## (2,R1)      1      0      1      0      5      0
## (3,R1)      0      1      0      0      0      5
## (1,R2)      5      0      0      0      3      0
## (2,R2)      0      5      0      2      0      4
## (3,R2)      0      0      5      0      0      0
```

Even though we created this larger index matrix from individuals components, we may not be sure it's exactly what we want. We can use `plotMKmodel()` to take a look at the model setup *before* running the analysis:

```
plotMKmodel(FullMat, rate.cat = 2, display = "row", text.scale = 0.7)
```



Since this is the model we intended on making, we can run `corHMM()` with our custom matrix:

```
HMM_3state_custom <- corHMM(phy = phy, data = data, rate.cat = 2, rate.mat = FullMat,
    node.states = "none")
```

```
## You specified 'fixed.nodes=FALSE' but included a phy object with node labels. These node labels have
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  29  10  21
## Beginning thorough optimization search -- performing 0 random restarts
```

```
HMM_3state_custom
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -40.62204 91.24408 92.35519        2   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)       (1,R2)      (2,R2)
## (1,R1)         NA 0.02362049         NA   0.04132034          NA
## (2,R1) 0.02362049         NA 0.02362049           NA  0.04132034
## (3,R1)         NA 0.02362049         NA           NA          NA
## (1,R2) 0.04132034         NA         NA           NA 17.29499644
## (2,R2)         NA 0.04132034         NA 100.00000000          NA
## (3,R2)         NA         NA 0.04132034           NA          NA
##            (3,R2)
## (1,R1)         NA
## (2,R1)         NA
## (3,R1) 0.04132034
## (1,R2)         NA
## (2,R2) 0.03925606
## (3,R2)         NA
##
## Arrived at a reliable solution
```

We may plot the resulting parameter estimates as before:

```
plotMKmodel(HMM_3state_custom, display = "row", text.scale = 0.7)
```