Using corHMM 2.0

James D. Boyko and Jeremy M. Beaulieu

Introduction

The original version of the R-package `corHMM` contained a number of distinct functions for conducting various analyses of discrete morphological characters. This included the `corHMM()` function for fitting a hidden rates model for a binary character, which uses "hidden" states as a means of allowing transition rates to vary across a tree. In reality, the hidden rates model falls within a general class of models known as hidden Markov model (HMM), and it need not only be applied to binary characters. The use of hidden states is a way of conceptualizing a rate class. With multiple hidden states, you have multiple rate classes. 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, so to speak. 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, **Q**, would be a simple 2x2 matrix,

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

This *transition rate matrix* is read as describing the transition rate *from* ROW *to* COLUMN. And as you can see there are just two transitions going from 0 to 1, and from 1 to 0 because there are only two states possible in this model. Now, suppose we have a second character that is also binary. This means that 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 accomodate this, we need to expand our model now such that it becomes a 4x4 matrix,

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

The model is also 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: 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 cannot immediately transition to state *11*, rather, it must first transition either to state *01* or *10* before finally transitioning to state *11*. So, we can simplify the matrix by removing these "dual" transitions from the model completely,

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

What we just described is essentially 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 solely 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 no petals, *3*=blue petals, and *4*=blue, no petals. The model would then be,

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

Notice it is the same as before, but the states are transformed from binary combinations to a multistate character. And to make this point even clearer, we will drop the "dual" transitions,

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

Again, exactly the same.

Here we have modified `corHMM()` to transform any character or set of characters into a *single* multistate character. The model can then be expanded to accomodate an arbitrary number of hidden states. Thus, `corHMM()` now contains `rayDISC()` and `corDISC()` capabilities with the added bonus of allow for hidden states. This vignette is comprised of three sections, where we demonstrate all these extensions as well as other new features:

- **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: Precursor model
    - 2.2.2: Ordered habitat change
    - 2.2.3: Ontological relationship of multiple characters
  - 2.3: Estimating models when node states are fixed
- **Section 3 Unit tests of corHMM**
  - 3.1: rayDISC-*like* models in corHMM
  - 3.2: corHMM works for n States
  - 3.3: corHMMv2.0 is the same as previous versions

Section 1: Default use of corHMM

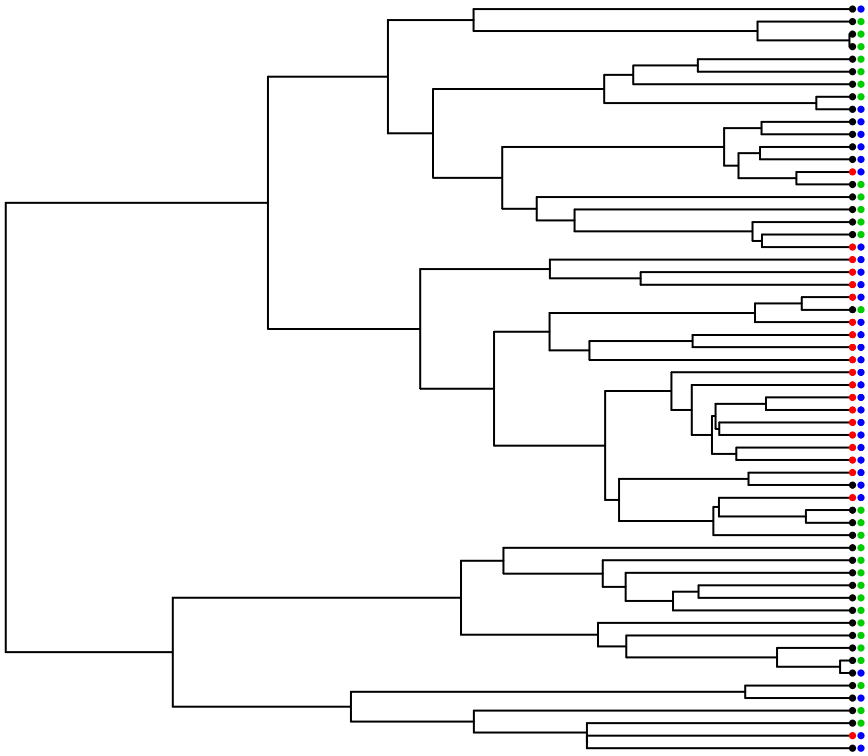
1.1: No hidden rate categories

We'll use the primate dataset that comes with corHMM.

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

```r
phy <- multi2di(phy)
data <- primates[[2]]
```

```r
plot(phy, show.tip.label = FALSE)
tiplabels(pch = 16, col = data[,2]+1, cex = 0.5)
tiplabels(pch = 16, col = data[,3]+3, cex = 0.5, offset = 0.5)
```



We have two characters each with two possible states. Trait 1 could either be presence (red) or absence of (black) tails. And trait 2 could be the presence (blue) or the absence (green) of coccyx.

The default use of corHMM only requires that you declare your *phylogeny*, your *dataset*, the number of *rate categories* (more detail about this later).

We have updated corHMMv2.0 to handle different types of input data. Previously, the data could only contain two columns: [,1] a column of species names, and [,2] a column of state values. As of corHMMv2.0, 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 an independently evolving character. Because of this, dual transitions are automatically disallowed. In our example, we expect that a species cannot go from absence of tail and coccyx to presence of tail and coccyx in one step.

```r
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.
```

By default, a marginal ancestral state reconstruction will be preformed.

```
MK_3state
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -41.91508 91.83016 92.55743        1   60
##
## Rates
##            (1,R1)     (2,R1)     (3,R1)
## (1,R1)         NA 0.01752732         NA
## (2,R1) 0.05439964         NA 0.02555553
## (3,R1)         NA 0.01551701         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 negative 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). Below *Fit* is the real meat of the model: *Rates*.

The rates describe transitions between states and are organized as a matrix. This *transition rate matrix* can be read as describing the transition rate **from** ROW **to** COLUMN. For example, if you were interested in the transition rate from State 1 (absence of tails & coccyx) to State 2 (absence of tails and presence of coccyx) 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 might be wondering how we knew that State 1 is absence of tails & coccyx and State 2 is absence of tails & presence of coccyx. After all, the data we input to corHMM was comprised of several columns. The reason the conversion occurs is to make the results more readable and to ensure consistency across all datasets given to corHMM. There are two ways to know which states represent which trait combinations. The first is to use the corHMM results object itself.

```
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
```

This will provide an augmented dataframe. It takes the initial user data and adds a column that corresponds to how corHMM treated each species. Alternatively, a user can supply their dataset to getRateMat4Dat which as one of its output provides a legend consistent with the corHMM function. The other output is an index matrix (or rate matrix) which describes which rates are to be estimated in corHMM (we'll talk more of the index matrix later).

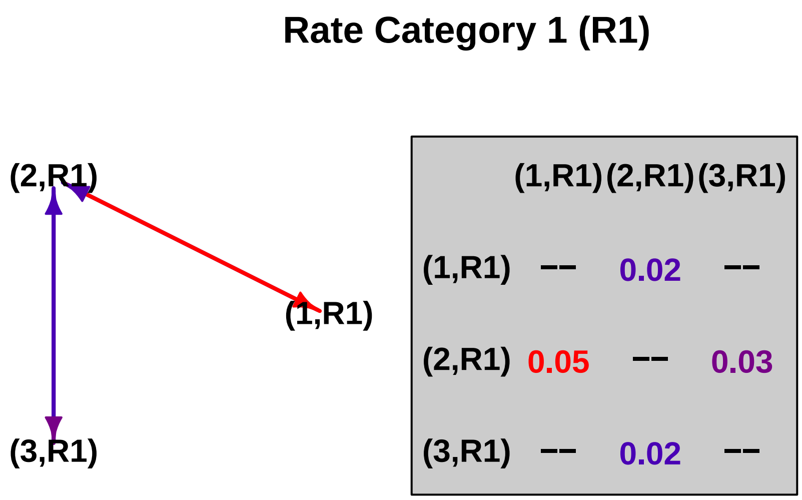
```
getRateMat4Dat(data)
```

```
## $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 use the more complex Hidden Markov models (which is done by setting rate.cat > 1). To help users we have implemented a new plotting function.

```r
plotMKmodel(MK_3state)
```



This function can take a corHMM object (which is the result of running corHMM) or a custom rate matrix (discussed in a later section) and plot the model in two parts. On the left is a ball and stick diagram depicting the transitions between the states. On the right is a simplified rate matrix (a rounded version of the solution output of corHMM). The colors of the arrows match the rates.

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

The major difference between corHMMv2.0 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. We will use the data from above and assign 2 rate categories.

```r
HMM_3state <- corHMM(phy = phy, data = data, rate.cat = 2, model = "SYM")
```

```
##
## 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.
```

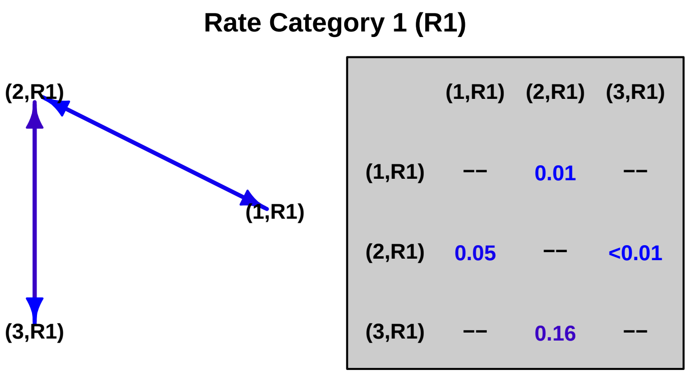
Models with more states take longer to run. Hidden rate models are no exception, and we have increased the number of parameters being estimated from 6 (in a 3-state one rate category Markov model) to 14 by adding a hidden rate category. In section 1.1 we left our parameters unconstrained. We estimated 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 said that the transition rates between any two states are equal. If we set model = "ER", then we would have constrained all transition rates between states to be the same. Finally, if model = "ARD" (the default), then all transition rates are independently estimated. Although "ER" and "SYM are common restrictions, it is often more useful to manually restrict your model to match a builogical hypothesis (which is described in the next section).

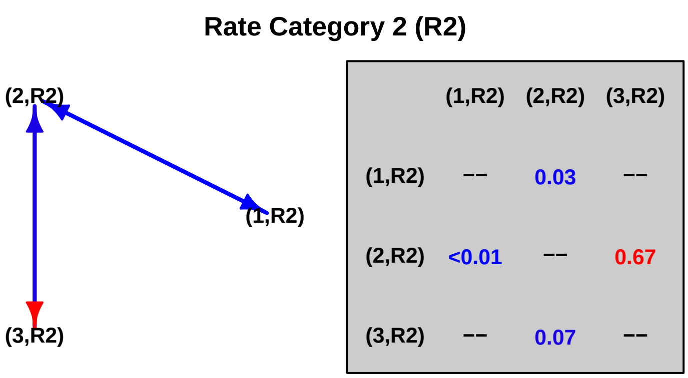
```r
round(HMM_3state$solution,3)
```

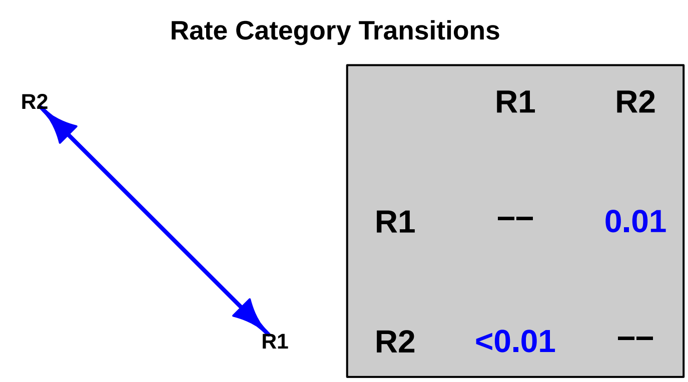
```
##        (1,R1) (2,R1) (3,R1) (1,R2) (2,R2) (3,R2)
## (1,R1)     NA  0.008     NA  0.011     NA     NA
## (2,R1)  0.046     NA      0     NA  0.011     NA
## (3,R1)     NA  0.156     NA     NA     NA  0.011
## (1,R2)  0.000     NA     NA     NA  0.035     NA
## (2,R2)     NA  0.000     NA  0.000     NA  0.667
## (3,R2)     NA     NA      0     NA  0.068     NA
```

Looking at the solution of this hidden Markov model is intimidating, but the same principles of interpreting the transition rate matrices apply. You still read rates from row to column. However, we have added different rate categories (as represented by R1 and R2. Each rate category contains the same model we had in section 1.1. That is to say, (1,R1),(2,R1), and (3,R1) are a Markov model of transitions between states 1, 2, and 3. We can plot the HMM using the plotMKmodel function. It will plot the underlying structure of model in discrete parts. The first 2 plots are descriptions of how observed states transition, whereas the final plot describes how these rate classes transition between one another.

```r
plotMKmodel(HMM_3state)
```







If you're confused, don't worry. This becomes much easier to interpret once you've learned how to create custom corHMM models.

Section 2: How to make and interpret custom models

2.1: Creating and using custom rate matrices

2.1.1: One rate category

A custom rate matrix allows you to specify how you think traits are expected to evolve. If you believe that traits evolve in a certain order, that there are different rates of character evolution in different clades, or that there are hidden precursors before a state can evolve, then a custom model is the best way to specify that particular hypothesis.

At its core, the purpose of a rate matrix (rate.mat) is to indicate to corHMM which parameters are being estimated. A rate.mat does not actually contain *rates*. It would more appropriately named an index matrix because it specifies to corHMM which rates in the matrix are being estimated and if any of them are expected to be identical.

Let's start by using the getRateMat4Dat function, but this time we'll focus on the rate.mat.

```
LegendAndRateMat <- getRateMat4Dat(data)
RateMat <- LegendAndRateMat$rate.mat
RateMat
```

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

This is the rate.mat that was internally called when we ran our model in section 1.1. The numbers in this matrix are not rates, they are used to index the parameters within 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) to (2) & (2) to (1) are equal AND that (3) to (2) and (2) to (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.

```
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
```

We used equateStateMatPars whose first argument is *the rate matrix being modified* and second argument is *list of the parameters to be equated* to recreate the "SYM" model.

```
pars2equal
```

```
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 3 4
```

One thing to note is that you must have the appropriate number of rate categories. A user rate matrix will not be duplicated or changed by corHMM. 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 give this rate.mat to corHMM, and it will estimate this constrained model.

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

```
##
## 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.
```

```
round(MK_3state_customSYM$solution, 3)
```

```
##        (1,R1) (2,R1) (3,R1)
## (1,R1)     NA  0.026     NA
## (2,R1)  0.026     NA   0.02
## (3,R1)     NA  0.020     NA
```

As you can see, transitions between, from, and to states are equal, as we had hoped.

2.1.2: Any number of rate categories

If you wanted to add hidden rate categories you need to know 2 things:

First, you need to know the dynamics within each rate category. In our case we might expect that at times, the whole process undergoes drift, with no strong trends. We can describe that in R1 with a model where all rates are equal. However, we may also think that for some species once they gain tails, they never lose them. We will describe that process in R2.

```
RateCat1 <- getRateMat4Dat(data)$rate.mat # R1
RateCat1 <- equateStateMatPars(RateCat1, c(1:4)) # set all rates to be equal
RateCat1
```

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

```
RateCat2 <- getRateMat4Dat(data)$rate.mat # R2
RateCat2 <- dropStateMatPars(RateCat2, 3) # once you have tails, you don't lose them (unless you went b
RateCat2
```

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

Second, you need to know how R1 and R2 relate to one another. This is an optional step. By default, corHMM will assume that all RateClasses can be transitioned between independently. If you do decide to specify how the rate categories relate to one another, your RateClassMat will have as many states as there are rate categories. I.e. the RateClassMat doesn't care about how many observed states you have. R1 and R2 describe how our three observed states change, but the RateClassMat describes how species change between R1 and R2. R1 and R2 could temporate or tropical, island or mainland, presence or absence of a necessary mutation. It is everything and anything that can influence the evolution of your observed characters.

In this case, we'll specify that transitions from R1 to R2 is the same as R2 to R1. This also introduces a new function, getStateMat. This simple function will create an index matrix of size nState by nState specified by the user.

```
RateClassMat <- getStateMat(2) # the size of this state matrix is the no. of rate classes
RateClassMat <- equateStateMatPars(RateClassMat, c(1,2))
RateClassMat
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    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)
```

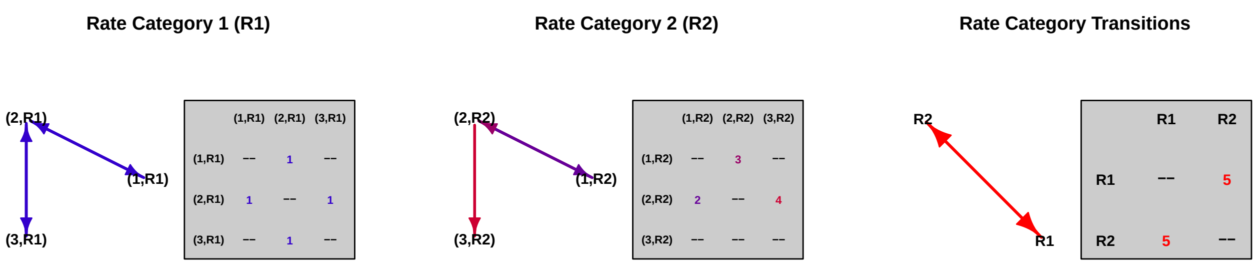
With that, we have all components necessary to create our model. We put it all together with getFullMat. getFullmat requires that the first input be a list of the rate class matrices and the second argument be how they are related to one another.

```
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 its individuals components we may not be sure it's exactly what we want. We can use plotMKmodel to also plot an index matrix. This makes it easy to make sure the custom you model you created is the one you want.

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



The first two plots are transitions between our observed states 1,2,3. If we focus just on those we can see the same general model structure that was present in section 1.1. As we intended, the dynamics of R1 differ from R2. R1 is an equal rates model, whereas R2 disallows transitions from State 3. The 3rd plot (Rate Category Transition matrix) describes how species transition between R1 and R2.

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 = "no
```

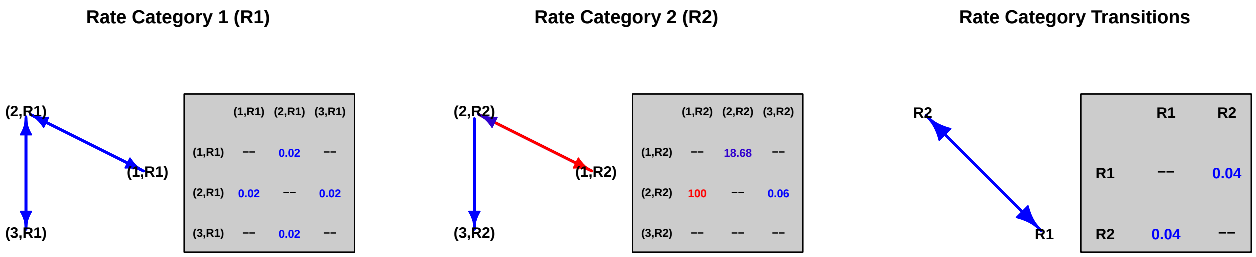
```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##       1       2      NA       3
## "0 & 0" "0 & 1" "1 & 0" "1 & 1"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in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

```
round(HMM_3state_custom$solution, 3)
```

```
##        (1,R1) (2,R1) (3,R1)  (1,R2) (2,R2) (3,R2)
## (1,R1)     NA  0.018     NA   0.038     NA     NA
## (2,R1)  0.018     NA  0.018      NA  0.038     NA
## (3,R1)     NA  0.018     NA      NA     NA  0.038
## (1,R2)  0.038     NA     NA      NA 18.678     NA
## (2,R2)     NA  0.038     NA 100.000     NA  0.056
## (3,R2)     NA     NA  0.038      NA     NA     NA
```

And now we can plot the HMM with rates instead of indices.

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



2.2: Biological examples

2.2.1: The precursor model

The precursor from Marazzi et al. (2012) is a good example to start with. They were interested in locating putative evolutionary precursors of plant extrafloral nectaries (EFNs). There are 2 states, absence (0) and presence (1) of extrafloral nectaries. However, they proposed that only species with a precursor could gain EFNs. Unfortunately, this precursor is not observed. Here is how we could code this model in corHMM using custom rate matrices.

We'll start by loading a simulated dataset consistent with the presence and absence of extrafloral nectaries.

```
phy <- read.tree("randomBD.tree")
load("simulatedData.Rsave")
head(Precur_Dat)
```

```
##      sp d
## s7   s7 0
```

```
## s14 s14 0
## s16 s16 0
## s17 s17 0
## s18 s18 1
## s21 s21 0
```

Let's get a starting rate matrix based on our dataset.

```
Precur_LegendAndMat <- getRateMat4Dat(Precur_Dat)
Precur_LegendAndMat
```

```
## $legend
##   1   2
## "0" "1"
##
## $rate.mat
##     (1) (2)
## (1)   0   2
## (2)   1   0
```

This legend tells us that the absence of EFNs will be State 1 in corHMM and the presence of EFNs will be State 2. The rate matrix tells us how these observed states are allowed to transition between one another. As of now, the rate of gain and rate of loss will differ. However, what if we wanted to model an unobserved state that influences our observed character? We can code this hidden state using different rate classes. The precursor is expected to be an unobserved character without which it is impossible to gain an EFN. Once we have the precursor however, transitions from absence to presence of EFN will be allowed. Now that we know how the hidden state influences our observed character we can make this using different rate classes.

The first rate class will represent how our observed character changes in the absence of the precursor. In this rate class, it will be impossible to gain an EFN.

```
Precur_R1 <- Precur_LegendAndMat$rate.mat
Precur_R1 <- dropStateMatPars(Precur_R1, 2)
Precur_R1
```

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

Next, we'll create a rate class consistent with the idea of a precursor. In this rate class we expect that species can either gain or lose EFNs. This is the same as the matrix produced by getRateMat4Dat.

```
Precur_R2 <- Precur_LegendAndMat$rate.mat
Precur_R2
```

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

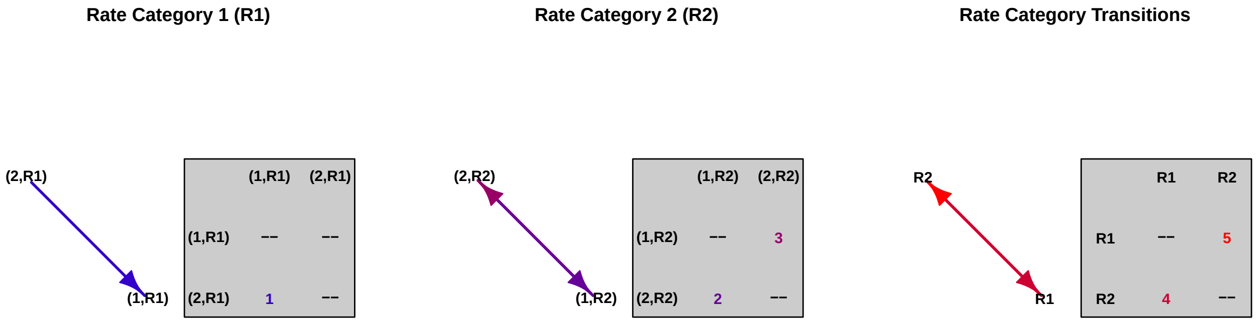
Put the rate classes together.

```
Precur_FullMat <- getFullMat(list(Precur_R1, Precur_R2))
Precur_FullMat
```

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

If you're unsure that the model is correct, you can plot the index matrix.

```
plotMKmodel(Precur_FullMat, 2, display = "row", text.scale = 0.7)
```



Since, it looks good we can now run corHMM making sure to specify that we have 2 rate categories (or rate classes or hidden states - it's all the same).

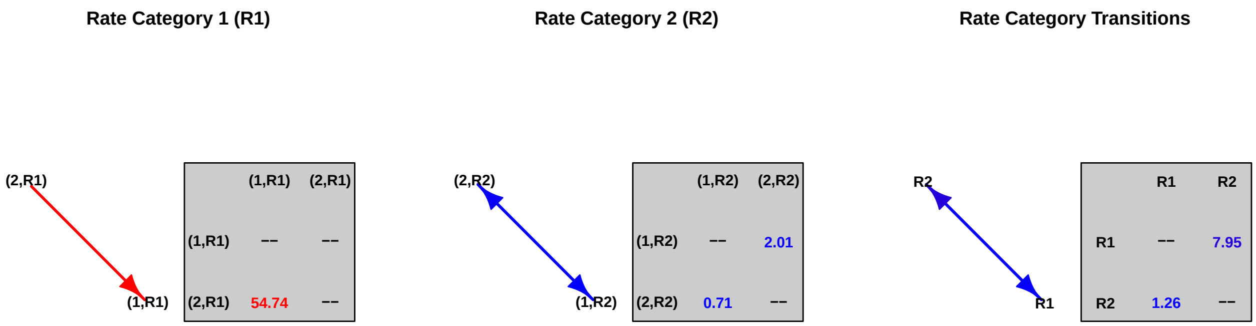
```
Precur_res.corHMM <- corHMM(phy = phy, data = Precur_Dat, rate.cat = 2, rate.mat = Precur_FullMat)

## State distribution in data:
## States:  1   2
## Counts:  57  43
## Beginning thorough optimization search -- performing 0 random restarts
## Finished. Inferring ancestral states using marginal reconstruction.
```

```
Precur_res.corHMM

##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -63.03259 136.0652 136.7035        2  100
##
## Rates
##             (1,R1)    (2,R1)    (1,R2)    (2,R2)
## (1,R1)         NA        NA 7.9507665        NA
## (2,R1) 54.739454        NA        NA 7.950766
## (1,R2)  1.259083        NA        NA 2.013322
## (2,R2)         NA 1.259083 0.7122854        NA
##
## Arrived at a reliable solution
```

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



In addition to plotting this model, let's look at each entry (row #, col #) and interpret the biological meaning.

```r
round(Precur_res.corHMM$solution, 3)
```

```
##        (1,R1) (2,R1) (1,R2) (2,R2)
## (1,R1)     NA     NA  7.951     NA
## (2,R1) 54.739     NA     NA  7.951
## (1,R2)  1.259     NA     NA  2.013
## (2,R2)     NA  1.259  0.712     NA
```

- Entry (2,1) is the rate of loss of extrafloral nectaries when the precursor is absent.
- Entries (3,1) and (4,1) are the rates at which the precursor is lost.
- Entries (1,3) and (1,4) are the rates at which the precursor is gained (remember we constrained that the rate of gain and loss were the same, hence the parameter estimates being the same).
- Entry (3,4) is the rate of gain of extrafloral nectaries when the precursor is present.
- Entry (4,3) is the rate of loss of extrafloral nectaries when the precursor is present.

2.2.2: Ordered habitat change

I'm working on a project concerned with the ancestral habitat during primary endosymbiosis. The possible habitats are marine, freshwater, and terrestrial. The phylogeny contains many species with a diverse range of life histories. Cyanobacteria can move freely between all of these states. But, some species may move between terrestrial and marine through freshwater. Finally, some species may move freely between aquatic states, but once they become terrestrial they are stuck there. In this section I will demonstrate how to create a custom hidden Markov model which satisfies all of these requirements. First I'm going to need 3 state matrices.

Start by simulating a dataset consistent with 3 states.

```r
Q <- matrix(abs(rnorm(9)), 3, 3)
diag(Q) <- 0
diag(Q) <- -rowSums(Q)
load("simulatedData.Rsave")
head(MFT_dat)
```

```
##    sp           d
## 1  s7  Freshwater
## 2 s14      Marine
## 3 s16      Marine
## 4 s17 Terrestrial
## 5 s18 Terrestrial
## 6 s21      Marine
```

```r
summary(as.factor(MFT_dat[,2])) # how many of each state do we have?
```

```
##  Freshwater      Marine Terrestrial
##           7          14          79
```

Start off by getting a legend and rate matrix consistent with this dataset.

```r
MFT_LegendAndRate <- getRateMat4Dat(MFT_dat)
MFT_LegendAndRate
```

```
## $legend
##             1             2             3
##  "Freshwater"      "Marine" "Terrestrial"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   3   5
```

```
## (2)   1   0   6
## (3)   2   4   0
```

In corHMM, freshwater habitat will be State 1, marine habitat will be State 2,and terrestrial habitat will be State 3. Now, we need to create 3 different rate classes that are consistent with our hypotheses of how habitat changes occurs. We'll say that Rate Class 1 is one in which linneages cannot leave a terrestrial habitat, Rate Class 2 will allow linneages to transition between marine & terrestrial only through freshwater, and Rate Class 3 will be unrestricted movement between the habitats.

For Rate Class 1 we need terrestrial to be a sink state. That means disallowing transitions out of terrestrial. Since 1 = Fresh, 2 = Marine, and 3 = Terra, that means removing from (3) to (1) and from (3) to (2).

```r
MFT_R1 <- dropStateMatPars(MFT_LegendAndRate$rate.mat, c(2,4))
MFT_R1
```

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

For Rate Class 2, we need to disallow transitions between terrestrial and marine. We disallow the positions (1,3) and (3,1) in the rate matrix. In this case any linneage can move into freshwater and move out of freshwater, but they are not allowed to transition directly between terrestrial and marine habitats.

```r
MFT_R2 <- dropStateMatPars(MFT_LegendAndRate$rate.mat, c(4,6))
MFT_R2
```

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

The free-moving matrix is already provided to us by getRateMat4Dat.

```r
MFT_R3 <- MFT_LegendAndRate$rate.mat
```

Let's put all these matrices in a list.

```r
MFT_ObsStateClasses <- list(MFT_R1, MFT_R2, MFT_R3)
MFT_ObsStateClasses
```

```
## [[1]]
##     (1) (2) (3)
## (1)   0   2   3
## (2)   1   0   4
## (3)   0   0   0
##
## [[2]]
##     (1) (2) (3)
## (1)   0   3   4
## (2)   1   0   0
## (3)   2   0   0
##
## [[3]]
##     (1) (2) (3)
## (1)   0   3   5
## (2)   1   0   6
## (3)   2   4   0
```

Since we only have 100 species let's constrain our parameters a bit further and say transitions between rate classes occur at the same rate.

```
MFT_RateClassMat <- getStateMat(3) # we have 3 rate classes
MFT_RateClassMat <- equateStateMatPars(MFT_RateClassMat, 1:6)
```

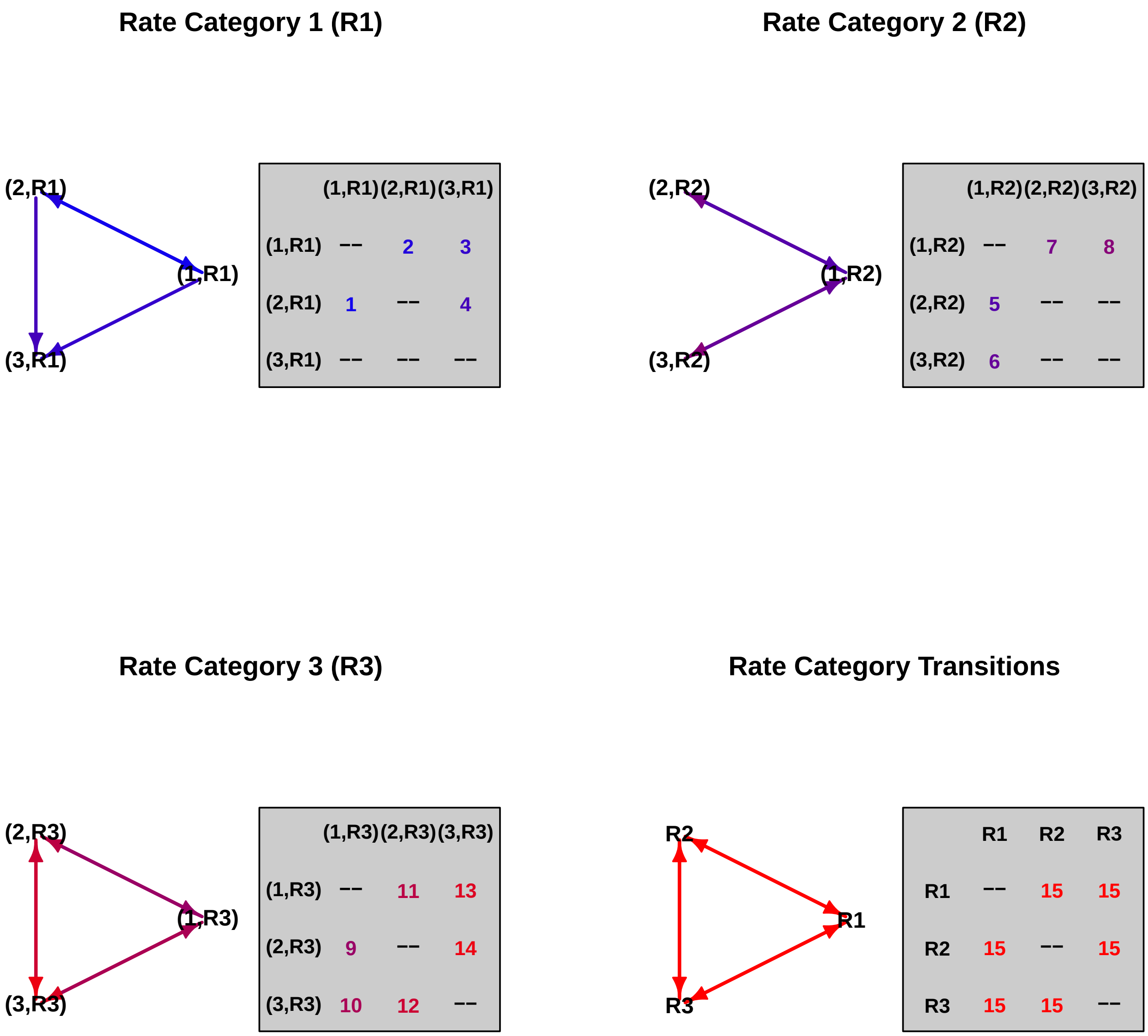
And we put it all together into a corHMM compatible rate.mat.

```
MFT_FullMat <- getFullMat(MFT_ObsStateClasses, MFT_RateClassMat)
MFT_FullMat
```

```
##        (1,R1) (2,R1) (3,R1) (1,R2) (2,R2) (3,R2) (1,R3) (2,R3) (3,R3)
## (1,R1)      0      2      3     15      0      0     15      0      0
## (2,R1)      1      0      4      0     15      0      0     15      0
## (3,R1)      0      0      0      0      0     15      0      0     15
## (1,R2)     15      0      0      0      7      8     15      0      0
## (2,R2)      0     15      0      5      0      0      0     15      0
## (3,R2)      0      0     15      6      0      0      0      0     15
## (1,R3)     15      0      0     15      0      0      0     11     13
## (2,R3)      0     15      0      0     15      0      9      0     14
## (3,R3)      0      0     15      0      0     15     10     12      0
```

That's kind of difficult to interpret, so let's plot it out and see if it's what we wanted.

```
plotMKmodel(pp = MFT_FullMat, rate.cat = 3, display = "square", text.scale = 0.9)
```



And it is. To run this model, we would only need to specify the data, the phylogeny, this matrix, and that this matrix has 3 rate categories.

```
MFT_res.corHMM <- corHMM(phy = phy, data = MFT_dat, rate.cat = 3, rate.mat = MFT_FullMat, node.states =
```

```
## State distribution in data:
## States:  1   2   3
## Counts:  7   14  79
## Beginning thorough optimization search -- performing 0 random restarts
```

```
MFT_res.corHMM
```

```
##
## Fit
##       -lnL      AIC     AICc Rate.cat ntax
##  -56.61096 143.2219 148.9362        3  100
##
## Rates
##              (1,R1)       (2,R1)       (3,R1)       (1,R2)     (2,R2)
## (1,R1)           NA 2.078508e-09 2.061154e-09 5.351547e+00         NA
```

```
## (2,R1) 2.061154e-09           NA 1.133523e+01           NA  5.351547
## (3,R1)           NA           NA           NA           NA        NA
## (1,R2) 5.351547e+00           NA           NA           NA 86.040044
## (2,R2)           NA 5.351547e+00           NA 2.061154e-09        NA
## (3,R2)           NA           NA 5.351547e+00 2.061154e-09        NA
## (1,R3) 5.351547e+00           NA           NA 5.351547e+00        NA
## (2,R3)           NA 5.351547e+00           NA           NA  5.351547
## (3,R3)           NA           NA 5.351547e+00           NA        NA
##              (3,R2)    (1,R3)       (2,R3)       (3,R3)
## (1,R1)           NA 5.3515469           NA           NA
## (2,R1)           NA        NA 5.351547e+00           NA
## (3,R1) 5.351547e+00        NA           NA 5.351547e+00
## (1,R2) 2.061154e-09 5.3515469           NA           NA
## (2,R2)           NA        NA 5.351547e+00           NA
## (3,R2)           NA        NA           NA 5.351547e+00
## (1,R3)           NA        NA 2.061154e-09 2.061154e-09
## (2,R3)           NA 0.3892012           NA 5.716380e-01
## (3,R3) 5.351547e+00 1.2256540 1.942339e-01           NA
##
## Arrived at a reliable solution
```

2.2.3: Ontological relationship of multiple characters

Lets say we had a dataset with multiple characters: presence or absence of limbs, presence or absence of fingers, corporeal or incorporeal form. It could look something like this…

```
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
Limbs <- c("Limbs", "noLimbs")[data[,2]+1]
Fings <- vector("numeric", length(phy$tip.label))
Fings[which(Limbs == "Limbs")] <- round(runif(length(which(Limbs == "Limbs")), 0, 1))
Corpo <- rep("corporeal", length(phy$tip.label))
Ont_Dat <- data.frame(sp = phy$tip.label, limbs = Limbs, fings = Fings, corp = Corpo)
head(Ont_Dat)
```

```
##                     sp   limbs fings      corp
## 1         Homo_sapiens noLimbs     0 corporeal
## 2         Pan_paniscus   Limbs     0 corporeal
## 3      Pan_troglodytes   Limbs     0 corporeal
## 4      Gorilla_gorilla   Limbs     0 corporeal
## 5        Pongo_pygmaeus   Limbs     0 corporeal
## 6 Pongo_pygmaeus_abelii   Limbs     1 corporeal
```

Previously, the user would have had to convert this dataset into something corHMM could use. This would mean taking all possible unique combinations and creating a corHMM specific dataset. Now, corHMM will internally convert this dataset and provide users with a legend in the results section for aiding the interpretation of the results. In addition, corHMM will remove any double transitions.

```
Ont_LegendAndMat <- getRateMat4Dat(Ont_Dat)
Ont_LegendAndMat
```

```
## $legend
##                        1                        2
##   "Limbs & 0 & corporeal"   "Limbs & 1 & corporeal"
```

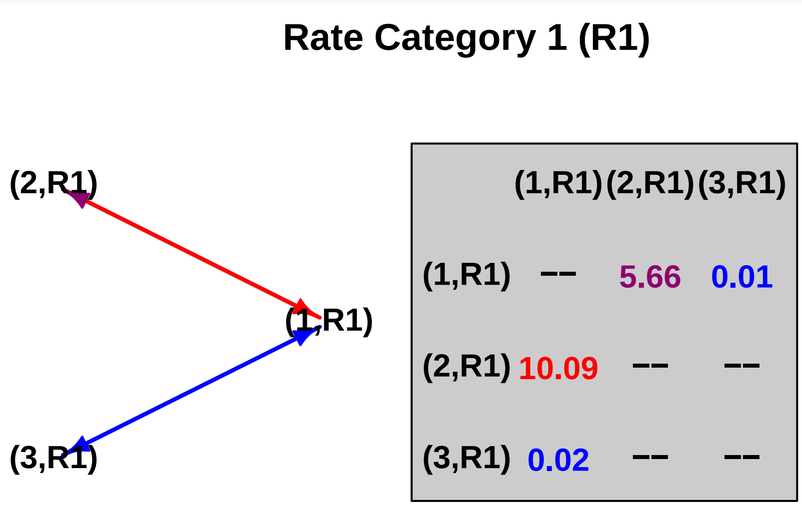
```
##                          3                         NA
## "noLimbs & 0 & corporeal" "noLimbs & 1 & corporeal"
##
## $rate.mat
##     (1) (2) (3)
## (1)   0   3   4
## (2)   1   0   0
## (3)   2   0   0
```

Even though there were 3 binary characters (meaning 8 possible states), only 3 combinations were actually found. This is because all of the organisms were corporeal and thus that state didn't factor into the matrix structure. The next thing to notice is that one of the potential states (No Limbs, Yes Fingers) is not present in the dataset and thus not included in the model. Finally, dual transitions have been removed. The transition from 3 (No Limbs, No Fingers) to 2 (Yes Limbs, Yes Fingers) is not allowed.

```
Ont_res.corHMM <- corHMM(phy = phy, data = Ont_Dat, rate.cat = 1, rate.mat = Ont_LegendAndMat$rate.mat,
```

```
##
## Input data has more than a single column of trait information, converting...
## 4 unique trait combinations found.
##                          1                          2
##   "Limbs & 0 & corporeal"   "Limbs & 1 & corporeal"
##                          3                         NA
## "noLimbs & 0 & corporeal" "noLimbs & 1 & corporeal"
##
## The potential number of trait combinations is 4, but only 3 were found.
##
## State distribution in data:
## States:  1   2   3
## Counts:  25  14  21
## Beginning thorough optimization search -- performing 0 random restarts
```

```
plotMKmodel(Ont_res.corHMM)
```



2.3: Estimating models when node states are fixed

Jeremy's code goes here.