Introduction to Phylogenetic Comparative Methods in R: Models of evolution

Natalie Cooper (ncooper@tcd.ie)

This problem set aims to show you how to use R to answer phylogenetic comparative questions. By the end of this problem set you should be able to:

1. Read your data and phylogeny into R
2. View and manipulate your data and phylogeny
3. Match taxa in your data with those in your phylogeny
4. Fit simple models of evolution to continuous data

Note that there is a second part to this problem set written by Graham Slater that will be available soon!

We will be using the evolution of primate life-history variables as an example. These data come from the PanTHERIA database (Jones et al., 2009) and 10kTrees (Arnold et al., 2010). Note that this is an old version of 10kTrees, so if you want to use it in your research please download the newest version.

Throughout, R code will be in shaded boxes:

```
install.packages("ape")
```

R output will be preceded by ## and important comments will be in boxes:

> The handout assumes that people have little or no experience with R so more experienced users may want to skip quickly through the first few sections. **To get the analyses from section 3 onwards to work you will need to have completed sections 1.1, 1.3, 1.4, 2.1 (reading in data only), 2.2 (reading in phylogeny only), 2.3 and 2.4.1**. Other sections are included to show you some of the other things R can do and to help you with problems you are likely to encounter when using your own data.

1 Preparations

> Note that many things in R can be done in multiple ways. You should choose the methods you feel most comfortable with, and do not panic if someone is doing the same analyses as you in a different way! This workshop will be full of different ways to do things.

1.1 Downloading the data and finding the path of your folder

First you need to download all the files for this problem set into a folder somewhere on your computer, let's call it "RAnalyses2014" and pop it on the Desktop. We will use this folder throughout the problem set. You'll need to know what the **path** of the folder is. For example on Natalie's Windows machine, the path is:

```
C:/Users/Natalie/Desktop/RAnalyses2014
```

The path is really easy to find in a Windows machine, just click on the address bar of the folder and the whole path will appear.

> In Windows, paths usually include \ but R can't read these. It's easy to fix in your R code, just change any \ in the path to / or \\.

On Natalie's Mac the path is:

```
~/Desktop/RAnalyses2014
```

It's a bit trickier to find the path on a Mac, so ask if you need help. Note that the tilde ~ is a shorthand for /Users/Natalie.

1.2 Using a text editor

Next, open a text editor. R has an inbuilt editor that works pretty well, but NotePad and TextEdit are fine too. However, in the future we **highly** recommend using something that will highlight code for you. Natalie's personal favorite is Sublime Text 2, because you can also use it for any other kind of text editing like LaTeX, html etc.

You should type (or copy and paste) your code into the text editor, edit it until you think it'll work, and then paste it into R's console window. Saving the text file lets you keep a record of the code you used, which can be a great timesaver if you want to use it again, especially as you know this code will work! It is also *essential* to keep **all** code used to produce the results of analyses and figures in your publications.

You can cut and paste code from this handout into your text editor, or straight into R. You don't need to retype everything!

If you want to add comments to the file (i.e., notes to remind yourself what the code is doing), put a hash/pound sign (#) in front of the comment.

```
# Comments are ignored by R but can remind you what the code is doing.
# You need a hash sign at the start of each line of a comment.
```

1.3 Installing packages in R

To run comparative analyses (or any specialized analysis) in R, you need to download one or more additional packages from the basic R installation. For this problem set you will need to install the following packages: `ape`, `geiger` and `OUwie`. To install the package `ape`:

```
install.packages("ape")
```

You will be asked to select a CRAN mirror for the session - just choose one that is geographically close to you. Now install `geiger` and `OUwie`.

1.4 Loading packages in R

You've installed the packages but they don't automatically get loaded into your R session. Instead you need to tell R to load them **every time** you start a new R session and want to use functions from these packages. To load the package `ape` into your current R session:

```
library(ape)
```

Don't forget to load `geiger` and `OUwie` too!

1.5 Citing packages in R

Note that all these extra packages take lots of work to write so make sure to cite them properly if you use them. You can get the citation of a package by typing:

```
citation(package = "ape")
```

You should also cite R. You can get the correct citation for the version you are using as follows:

```
citation()
```

Make sure to include version numbers in your citations for reproducibility purposes.

2 Reading your data and phylogeny into R

2.1 Reading data into R

Next we need to load the data we are going to use for the analysis. R can read files in lots of formats, including comma-delimited and tab-delimited files. Excel (and many other applications) can output files in this format (it's an option in the "Save As" dialog box under the "File" menu). To save time we have given you a tab-delimited text file called "Primatedata.txt" which we are going to use. Load these data as follows.

> You will need to replace MYPATH with the name of the path to the folder containing the data and tree on your computer.

```
primatedata <- read.table("MYPATH/Primatedata.txt", sep = "\t",
                          header = TRUE)
```

Note that `sep = "\t"` indicates that you have a tab-delimited file, `sep = ","` would indicate a comma-delimited csv file. You can also use `read.delim` for tab delimited files or `read.csv` for comma delimited files. `header = TRUE`, indicates that the first line of the data contains column headings.

This is a good point to note that unless you **tell** R you want to do something, it won't do it automatically. So here if you successfully entered the data, R won't give you any indication that it worked. Instead you need to specifically ask R to look at the data.

We can look at the data by typing:

```
str(primatedata)
```

```
## 'data.frame':    77 obs. of  8 variables:
##  $ Order          : Factor w/ 1 level "Primates": 1 1 1 ...
##  $ Family         : Factor w/ 15 levels "Aotidae","Atelidae",...
##  $ Binomial       : Factor w/ 77 levels "Alouatta palliata",...
##  $ AdultBodyMass_g: num  6692 7582 8697 958 558 ...
##  $ GestationLen_d : num  138 226 228 164 154 ...
##  $ HomeRange_km2  : num  2.28 0.73 1.36 0.02 0.32 0.02 ...
##  $ MaxLongevity_m : num  336 328 454 304 215 ...
##  $ SocialGroupSize: num  14.5 42 20 2.95 6.85 ...
```

This shows the structure of the data frame (this can be a really useful command when you have a big data file). It also tells you what kind of variables R thinks you have (characters, integers, numeric, factors etc.). Some R functions need the data to be certain kinds of variables so it's useful to check this.

As you can see, the data contains the following variables: Order, Family, Binomial, AdultBodyMass_g, GestationLen_d, HomeRange_km2, MaxLongevity_m, and SocialGroupSize.

```
head(primatedata)
```

```
##      Order      Family           Binomial AdultBodyMass_g GestationLen_d
## 1 Primates    Atelidae   Ateles belzebuth          6692.4          138.2
## 2 Primates    Atelidae   Ateles geoffroyi          7582.4          226.4
## 3 Primates    Atelidae    Ateles paniscus          8697.2          228.2
## 4 Primates Pitheciidae  Callicebus moloch           958.1          164.0
## 5 Primates     Cebidae  Callimico goeldii           558.0          154.0
## 6 Primates     Cebidae Callithrix jacchus           290.2          144.0
##   HomeRange_km2 MaxLongevity_m SocialGroupSize
## 1          2.28          336.0           14.50
## 2          0.73          327.6           42.00
## 3          1.36          453.6           20.00
## 4          0.02          303.6            2.95
## 5          0.32          214.8            6.85
## 6          0.02          201.6            8.55
```

This gives you the first few rows of data along with the column headings.

```
names(primatedata)
```

```
## [1] "Order"           "Family"          "Binomial"        "AdultBodyMass_g"
## [5] "GestationLen_d"  "HomeRange_km2"   "MaxLongevity_m"  "SocialGroupSize"
```

This gives you the names of the columns.

```
primatedata
```

This will print out all of the data!

2.2 Reading and displaying your phylogeny in R

To load a tree you need either the function `read.tree` or `read.nexus`. `read.tree` can deal with a number of different types of data (including DNA) whereas `read.nexus` reads NEXUS files. We will use a NEXUS file of the consensus tree from 10kTrees.

You will need to replace MYPATH with the name of the path to the folder containing the data and tree on your computer.

```
primatetree <- read.nexus("MYPATH/consensusTree_10kTrees_Version2.nex")
```

Let's examine the tree by typing:

```
primatetree
```

```
## Phylogenetic tree with 226 tips and 221 internal nodes.
##
## Tip labels:
##  Allenopithecus_nigroviridis, Cercopithecus_ascanius,
Cercopithecus_cephus, Cercopithecus_cephus_cephus, ...
##
## Rooted; includes branch lengths.
```

```
str(primatetree)
```

```
## List of 4
##  $ edge       : int [1:446, 1:2] 227 228 229 230 231 232 ...
##  $ edge.length: num [1:446] 4.95 17.69 19.65 8.12 4.82 ...
##  $ Nnode      : int 221
##  $ tip.label  : chr [1:226] "Allenopithecus_nigroviridis" ...
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

`primatetree` is a fully resolved tree with branch lengths. There are 226 species and 221 internal nodes. We can plot the tree by using the `plot` function of `ape`:

```
plot(primatetree)
```

Note that the tree has too many species for us to read the species names. You can make the tip labels smaller, but that doesn't help much here:

```
plot(primatetree, cex = 0.5)
```

Alternatively you can zoom into different sections of the tree that you're interested in:

```
zoom(primatetree, list(grep("Cercopithecus", primatetree$tip.label)),
      subtree = FALSE)
```

This just gives you the tree for *Cercopithecus* species but you can also see how the species fit into the rest of the tree using:

```
zoom(primatetree, list(grep("Cercopithecus", primatetree$tip.label)),
      subtree = TRUE)
```

Note that `zoom` automatically sets the plotting window to display two plots at once. To reset this to one plot only use:

```
par(mfrow = c(1, 1))
```

You can also remove species from the tree very easily using the `ape` function `drop.tip`:

```
primatetree2 <- drop.tip(primatetree,  "Aotus_azarae_infulatus")

str(primatetree2)
```

```
## List of 4
##  $ edge       : int [1:444, 1:2] 226 227 228 229 ...
##  $ edge.length: num [1:444] 4.95 17.69 19.65 8.12 ...
##  $ Nnode      : int 220
##  $ tip.label  : chr [1:225] "Allenopithecus_nigroviridis" ...
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

To remove more than one species see below. To get further options for the plotting of phylogenies:

```
?(plot.phylo)
```

Note that although you can use `plot` to plot the phylogeny, you need to specify `plot.phylo` to find out the options for plotting trees. You can change the style of the tree (type), the color of the branches and tips (edge.color, tip.color). Here's an fun* example! *Definitions of fun may vary.

```
par(mfrow = c(1, 1))

plot(primatetree, type = "fan", edge.color = "deeppink", tip.color =
      rainbow(8), cex = 0.5)
```

You can also add a timescale to your tree:

```
plot(primatetree)

axisPhylo()
```

For many more options for tree plotting check out Liam Revell's `phytools` package.

2.3 Making your phylogeny binary and rooting your phylogeny in R

Most R functions require your tree to be dichotomous, i.e. to have no polytomies. To check whether your tree is dichotomous use `is.binary.tree`. If this is FALSE, use `multi2di` to make the tree dichotomous. This function works by randomly resolving polytomies with zero-length branches.

```
is.binary.tree(primatetree)# we want this to be TRUE
```

```
## [1] FALSE
```

```
primatetree <- multi2di(primatetree)
```

Most functions also require the tree to be rooted, i.e., to have one taxon designated as the outgroup. Our tree is rooted but if you wanted to change the root, or root an unrooted tree use `root`. Note that here we've just chosen a random species (*Saimiri sciureus*) to be the root.

```
primatetree.reroot <- root(primatetree, "Saimiri_sciureus")

plot(primatetree.reroot)
```

2.4 Matching species names in your data and phylogeny in R

2.4.1 Species names with spaces

Species names in the tree cannot contain spaces so they are generally written as Genus_species (the gap between the genus name and species name replaced by _). If the species names in the data are written as Genus species with a space, then you will have to replace the spaces with _ so that they match up with the species names in the tree. You can do this as follows:

```
primatedata$Binomial <- gsub(" ", "_", primatedata$Binomial)
```

`gsub` means **g**eneral **sub**stitution. It replaces any instance of the first item (here it's a space) with the second item (_) but only in the variable you tell it to (`primatedata$Binomial`).

2.4.2 Species names = row names

A number of R functions require that species names are the row names of the data. This is really easy to fix:

```
row.names(primatedata) <- primatedata$Binomial
```

2.4.3 Mismatches between species in your data and phylogeny

Often you will have data for species which are not in your phylogeny and/or species in your phylogeny which are not in your data. Some functions in R can deal with this, others will produce an error telling you the tree and data do not match (e.g., most ape functions). It's useful to know how to deal with this so we have provided code below.

> Note that many R functions match up the species names in the tree and data for you before you run any analyses. However, these functions are only as good as their inputs. If you have even slightly misspelled a species name in the tree or the data it will automatically be dropped from the analyses. It is therefore **very important** to check this before running an analysis.

2.4.4 Species in the phylogeny but not in the data

These are easy to identify using `setdiff`:

```
setdiff(primatetree$tip.label, primatedata$Binomial)
```

```
##   [1] "Allenopithecus_nigroviridis"
##   [2] "Cercopithecus_cephus_cephus"
##   [3] "Cercopithecus_cephus_ngottoensis"
##   [4] "Cercopithecus_diana"
etc.
```

`setdiff` tells you what is found in the first list, but not in the second. Here, it has listed the species that are found in the tree but **not** in the data. We can then use `setdiff` with `drop.tip` to prune the tree to just the species we have data for.

```
primatetree2 <- drop.tip(primatetree, setdiff(primatetree$tip.label,
                         primatedata$Binomial))
```

Note that you need to list the species which you do **not** want to select and then drop them from the tree instead of selecting the species you want.

2.4.5 Species in the data but not in the phylogeny

Again these are easy to identify using `setdiff`, just swap the inputs around:

```
setdiff(primatedata$Binomial, primatetree$tip.label)
```

```
## character(0)
```

In this case we don't have any species in the tree missing from the data. However, if you do, to remove species from the data which are not in the tree you can use `match` and `subset`:

```
matches <- match(primatedata$Binomial, primatetree2$tip.label, nomatch = 0)

primatedata2 <- subset(primatedata, matches != 0)
```

Remember `!=` means "does not equal". So this line of code only selects species which do appear in the tree, i.e. their value from `matches` is not 0.

Always check this has worked as expected by checking the data and the phylogeny. In the first instance you can just use `str` to make sure you have the expected number of species in each:

```
str(primatedata2)

str(primatetree2)
```

3 Fitting simple models of evolution to continuous data

For fitting models of evolution to continuous data we will use the `fitContinuous` function in the R package `geiger`.

`fitContinuous` is a likelihood based method, so the output will give the maximum likelihood (ML) estimates of the parameters. Bayesian methods are becoming preferred for these kinds of analyses and `fitContinuousMCMC` will perform these analyses. However, due to time constraints (and a need to accompany it with a primer on Bayesian statistics!) we will not cover this function.

As an example, let's look at the evolution of log(body size) in Primates. We'll fit three evolutionary models – the Brownian motion (BM) model, the Ornstein-Uhlenbeck (OU) model and the Early Burst (EB) model. `fitContinuous` can also fit several other models. For more details look at the help file by typing:

```
?fitContinuous
```

3.1 Model descriptions

3.1.1 The Brownian motion (BM model)

The Brownian motion model ((Cavalli-Sforza and Edwards, 1967; Felsenstein, 1973) is assumed to be the underlying mode of evolution in the majority of phylogenetic comparative methods (though this assumption is rarely tested; Freckleton and Harvey 2006). In the model, a trait X evolves at random at a rate σ:

$$dX(t) = \sigma dW(t) \tag{1}$$

where $W(t)$ is a white noise function and is a random variate drawn from a normal distribution with mean 0 and variance σ^2. This model assumes that there is no overall drift in the direction of evolution (hence the expectation of $W(t)$ is zero) and that the rate of evolution is constant. Because the direction of change in trait values at each step is random, Brownian motion is often described as a "random walk" (note that you can also fit models where $W(t)$ is not zero and there is drift in the direction of evolution. This is known as the drift model, or Brownian motion with a trend. We will not cover this here).

The model assumes the correlation structure among trait values is proportional to the extent of shared ancestry for pairs of species. This means that close relatives will be more similar in their trait values than more distant relatives. It also means that variance in the trait will increase (linearly) in proportion to time.

The model has two parameters, the Brownian rate parameter, σ^2 and the state of the root at time zero, $X(0)$. `fitContinuous` estimates σ^2 and $X(0)$. Note that σ^2 can be used as a measure of the rate of trait evolution (Cooper and Purvis, 2010; Cooper et al., 2011).

3.1.2 The Ornstein-Uhlenbeck (OU) model

The Ornstein-Uhlenbeck (OU) model (Hansen, 1997; Butler and King, 2004) is a random walk where trait values are pulled back towards some "optimal" value with an attraction strength proportional to the parameter α. The model has the following form:

$$dX(t) = -\alpha(X(t) - \mu) + \sigma dW(t) \tag{2}$$

Note that this model has two parameters in addition to those of the Brownian model, α and μ. The parameter μ is a long-term mean, and it is assumed that species evolve around this value. α is the strength of evolutionary force that returns traits back towards the long-term mean if they evolve away from it. α is sometimes referred to as the "rubber band" parameter because of the way it forces traits back towards μ.

The OU model was introduced to population genetics by Lande (1976) to model stabilizing selection in which the mean was recast as a fitness optimum on an adaptive landscape. The process operating in comparative data is analogous, although clearly is not stabilizing selection (despite being sometimes referred to as such).

The model has four parameters, the Brownian rate parameter, σ^2, the state of the root at time zero, $X(0)$, the attraction strength or "rubber band" parameter, α, and the long-term mean, μ.

`fitContinuous` estimates σ^2, $X(0)$, and α. It does not estimate μ but in this implementation of the model, μ is equivalent to $X(0)$. Note that if α is close to zero then evolution is approximately Brownian.

3.1.3 The Early Burst (EB) model

The Early Burst (EB) model (Harmon et al. 2010, also called the ACDC model; Blomberg et al. 2003) is a Brownian motion/random walk model where the rate of evolution decreases exponentially through time under the model:

$$r(t) = \sigma^2 e(at) \tag{3}$$

Where $r(t)$ is the rate of evolution at time t, σ^2 is the initial value of the Brownian rate parameter, i.e. the initial rate of evolution, a is the rate change parameter, and t is time. The value of a is generally less than or equal to o (note that you can force a to be greater than zero by changing the bounds - see section 3.4 - however, this will only work if you have fossil species in your data; Slater et al. 2012). When a is negative, rates of evolution decrease through time.

The model fits traits where diversification occurs most rapidly early in a lineage and slows as the lineage approaches the present, so that subclades tend to retain their differences through time. This is consistent with a clade radiating adaptively into a fixed set of niches and has been used as evidence of niche-filling modes of evolution (Harmon et al., 2010; Cooper and Purvis, 2010).

The model has three parameters, the Brownian rate parameter, σ^2, the state of the root at time zero, $X(0)$, and the rate of change parameter, a. `fitContinuous` estimates σ^2, $X(0)$ and a. Note that if a is close to zero then evolution is approximately Brownian.

Note that although many people report a values when reporting the results of fitting an Early Burst model, it is often more intuitive to report the rate half-life, $t_{\frac{1}{2}}$ (Slater and Pennell, 2014). This is calculated as:

$$t_{\frac{1}{2}} = \frac{log(2)}{|a|} \tag{4}$$

It can be interpreted as the time it takes for the rate of evolution of the trait to halve (see example below).

3.2 Fitting the models using fitContinuous

Before we can fit the models we need to get the tree and data into a format that `geiger` can deal with. All `geiger` functions require that species names are the row names of the data. This is really easy to fix:

```
row.names(primatedata) <- primatedata$Binomial
```

Next, for some weird reason the function below don't work if you input a dataset with variables that are characters i.e. words or letters. Our taxonomic variables `Order`, `Family` and `Binomial` are characters so we need to exclude them from the data. We will do this by making a new dataset called `primatedata2`.

```
primatedatasubset <- (primatedata[, 4:8])
```

Here the [] tells R we want to subset the dataset. R data frames are always described by [X,Y] where X is rows and Y is columns. So [1, 1] will select the entry in the first column and the first row of the data frame. [, 4:8] selects all rows but only columns 4 to 8. These are the columns containing our numeric variables.

The fitContinuous function also requires that the species in the phylogeny match those in the dataset. To do this we use the geiger function treedata:

```
match.species <- treedata(phy = primatetree, dat = primatedatasubset, sort = TRUE)
```

This produces the following warning message telling us the species that were not found in both the data and the tree and have been dropped.

```
## Warning: The following tips were not found in 'data' and were dropped from 'phy':
##  Allenopithecus_nigroviridis
##  Allocebus_trichotis
##  Alouatta_caraya
##  Alouatta_sara
etc.
```

> Note that if you have even slightly misspelled a species name in the tree or the data it will automatically be dropped from the analyses. It is therefore **very important** to check this list before running an analysis.

Next we use the output from the treedata function to get a dataset and phylogeny with perfectly matching names:

```
mytree <- match.species$phy

mydata <- match.species$data
```

We are now ready to fit our models of evolution. First we will fit a Brownian motion (BM) model to log(body mass):

```
BM <- fitContinuous(phy = mytree, dat = log(mydata[,"AdultBodyMass_g"]),
                    model = c("BM"))
```

To look at the output type:

```
BM
```

The output should look like this:

```
## GEIGER-fitted comparative model of continuous data
##  fitted 'BM' model parameters:
##  sigsq = 0.028655
##  z0 = 6.773956
##
##  model summary:
##  log-likelihood = -78.096042
##  AIC = 160.192084
##  AICc = 160.354246
##  free parameters = 2
##
## Convergence diagnostics:
##  optimization iterations = 100
##  failed iterations = 0
##  frequency of best fit = 1.00
##
##  object summary:
##  'lik' -- likelihood function
##  'bnd' -- bounds for likelihood search
##  'res' -- optimization iteration summary
##  'opt' -- maximum likelihood parameter estimates
```

The maximum likelihood estimates (lnL) of the model parameters are found near the top of the output. In a Brownian motion (BM) model we estimate the Brownian rate parameter, σ^2 or `sigsq` in the output above, which is 0.028655 and the value of the trait at the root of the tree, X_0 or z0 in the output above, which is 6.773956.

Other useful things in the output are the maximum-likelihood estimate (lnL) of the model (`log-likelihood`), the Akaike Information Criterion (`AIC`), sample-size corrected AIC (`AICc`) and the number of model parameters (`free parameters`)also known as k in the literature. We will return to the AIC values below.

To fit an Ornstein-Uhlenbeck model to log(body mass) we only need to change the model in the formula we used above:

```
OU <- fitContinuous(phy = mytree, dat = log(mydata[,"AdultBodyMass_g"]),
                    model = c("OU"))
```

To look at the output type:

```
OU
```

The output should look like this:

```
## GEIGER-fitted comparative model of continuous data
##  fitted 'OU' model parameters:
##  alpha = 0.000000
##  sigsq = 0.028655
##  z0 = 6.773956
##
##  model summary:
##  log-likelihood = -78.096042
##  AIC = 162.192084
##  AICc = 162.520851
##  free parameters = 3
##
## Convergence diagnostics:
##  optimization iterations = 100
##  failed iterations = 0
##  frequency of best fit = 0.54
##
##  object summary:
##  'lik' -- likelihood function
##  'bnd' -- bounds for likelihood search
##  'res' -- optimization iteration summary
##  'opt' -- maximum likelihood parameter estimates
```

As above, the maximum likelihood estimates (lnL) of the model parameters are found near the top of the output. In an Ornstein-Uhlenbeck (OU) model we estimate the Brownian rate parameter, σ^2 or `sigsq` in the output above, the value of the trait at the root of the tree, X_0 or `z0` in the output above, and the selection strength parameter, α or `alpha` in the output above. As $alpha = 0$ here, the model is identical to the Brownian motion model.

Finally, to fit an Early Burst (EB) model to log(body mass):

```
EB <- fitContinuous(phy = mytree, dat = log(mydata[,"AdultBodyMass_g"]),
                    model = c("EB"))
```

To look at the output type:

```
EB
```

The output should look like this:

```
## GEIGER-fitted comparative model of continuous data
##  fitted 'EB' model parameters:
##  a = -0.037692
##  sigsq = 0.276300
##  z0 = 6.695068
##
##  model summary:
##  log-likelihood = -75.428628
##  AIC = 156.857257
##  AICc = 157.186024
##  free parameters = 3
##
## Convergence diagnostics:
##  optimization iterations = 100
##  failed iterations = 0
##  frequency of best fit = 0.20
##
##  object summary:
##  'lik' -- likelihood function
##  'bnd' -- bounds for likelihood search
##  'res' -- optimization iteration summary
##  'opt' -- maximum likelihood parameter estimates
```

As above, the maximum likelihood estimates (lnL) of the model parameters are found near the top of the output. In an Early Burst (EB) model we estimate the Brownian rate parameter, σ^2 or `sigsq` in the output above, the value of the trait at the root of the tree, X_0 or $z0$ in the output above, and the rate of change parameter, a. Here $a = -0.0377$ indicating that the rate of log(body mass) evolution in Primates has decreased through time.

We can also extract the rate half-life, $t_{\frac{1}{2}}$, for this model as follows:

```
log(2)/abs(EB$opt$a)
```

```
## [1] 18.38991
```

For these data, the rate half-life is ≈ 18 million years, which means over the course of primate evolution, body size rate has undergone ≈ 4 rate halvings. This is also well within the range of "hard to detect" bursts reported in Figure 1 of Slater and Pennell (2014).

3.3 Comparing models

Often we want to know which of the models fits our variable best. We can use `fitContinuous` to fit the models we are interested in and can then compare them using AIC. We can extract the AICs from the models we fitted above as follows:

```
BM$opt$aic
```

```
##  160.1921
```

```
OU$opt$aic
```

```
##  162.1921
```

```
EB$opt$aic
```

```
##  156.857257
```

The "best" model is the one with the smallest AIC, in this case the Early Burst model. There is much debate about how big of a difference in AIC values can be classed as substantial improvement to a model fit (it ranges from 2-10 AIC units). Generally we use 4 units, so EB probably doesn't fit substantially better than BM ($160.1921 - 156.857257 = 3.334843$).

Alternatively we can use ΔAIC or AIC weights to compare our models using the following code and the `geiger` function `aicw`:

```
aic.scores <- setNames(c(BM$opt$aic, OU$opt$aic, EB$opt$aic),
                       c("BM","OU","EB"))

aicw(aic.scores)
```

```
##          fit    delta          w
##  BM 160.1921 3.334827 0.15000767
##  OU 162.1921 5.334827 0.05518474
##  EB 156.8573 0.000000 0.79480759
```

`aicw` outputs the AIC (`fit`), ΔAIC (`delta`) and AIC weights (`w`) for each of the models we fitted. The best model is the model with $\Delta AIC = 0$ or with AICw closest to 1. Using ΔAIC we can conclude that the Early Burst model is the best fit to the data.

3.4 Problems with convergence and bounds

Above we have mentioned the default bounds on each parameter. Sometimes these need to be changed because the model will not converge. This happens when the likelihood surface has long flat ridges that cause the likelihood search to get "stuck" (this is particularly common under the OU model). You can change bounds with the `bounds` argument in `fitContinuous`. Several bounds can be given at a time e.g. `bounds=list(sigsq=c(0,0.1),alpha=c(0,1))` would constrain both the σ^2 and α parameters.

For example, if an OU model keeps getting stuck you could try changing the lower bound on α:

```
OU <- fitContinuous(phy = mytree, dat = log(mydata[,"AdultBodyMass_g"]),
                    model = c("OU"), bounds = list(alpha = c(0.01, 1)))
```

This example gives the following warning message because α is zero, so when we make the lower bound larger the method ends up giving us this value instead because it's as close to zero as we are allowing the model to go.

```
##  Warning message:
##  In fitContinuous(phy = mytree, dat = log(mydata[, "AdultBodyMass_g"]),  :
##    Parameter estimates appear at bounds:
##    alpha
```

References

Arnold, C., L. J. Matthews, and C. L. Nunn. 2010. The 10ktrees website: a new online resource for primate phylogeny. Evolutionary Anthropology: Issues, News, and Reviews 19:114–118.

Blomberg, S. P., T. Garland, and A. R. Ives. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. Evolution 57:717–745.

Butler, M. A. and A. A. King. 2004. Phylogenetic comparative analysis: a modeling approach for adaptive evolution. The American Naturalist 164:683–695.

Cavalli-Sforza, L. L. and A. W. Edwards. 1967. Phylogenetic analysis. models and estimation procedures. American Journal of Human Genetics 19:233.

Cooper, N., R. P. Freckleton, and W. Jetz. 2011. Phylogenetic conservatism of environmental niches in mammals. Proceedings of the Royal Society B: Biological Sciences 278:2384–2391.

Cooper, N. and A. Purvis. 2010. Body size evolution in mammals: complexity in tempo and mode. The American Naturalist 175:727–738.

Felsenstein, J. 1973. Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters. Systematic Biology 22:240–249.

Freckleton, R. P. and P. H. Harvey. 2006. Detecting non-brownian trait evolution in adaptive radiations. PLoS Biology 4:e373.

Hansen, T. F. 1997. Stabilizing selection and the comparative analysis of adaptation. Evolution Pages 1341–1351.

Harmon, L. J., J. B. Losos, T. Jonathan Davies, R. G. Gillespie, J. L. Gittleman, W. Bryan Jennings, K. H. Kozak, M. A. McPeek, F. Moreno-Roark, T. J. Near, et al. 2010. Early bursts of body size and shape evolution are rare in comparative data. Evolution 64:2385–2396.

Jones, K. E., J. Bielby, M. Cardillo, S. A. Fritz, J. O'Dell, C. D. L. Orme, K. Safi, W. Sechrest, E. H. Boakes, C. Carbone, et al. 2009. Pantheria: a species-level database of life history, ecology, and geography of extant and recently extinct mammals: Ecological archives e090-184. Ecology 90:2648–2648.

Lande, R. 1976. Natural selection and random genetic drift in phenotypic evolution. Evolution 30:314–334.

Slater, G. J., L. J. Harmon, and M. E. Alfaro. 2012. Integrating fossils with molecular phylogenies improves inference of trait evolution. Evolution 66:3931–3944.

Slater, G. J. and M. W. Pennell. 2014. Robust regression and posterior predictive simulation increase power to detect early bursts of trait evolution. Systematic Biology 63:293–308.