Phylogenetics in R

Martin Brazeau (m.brazeau@imperial.ac.uk)

2016-11-14

1. Introduction and resources

This practical introduces you to basic phylogenetic computing in R. We will review importing phylogenetic trees as data files, displaying phylogenetic trees visually, and some basic evolutionary computations that can be conducted with phylogenetic trees. This practical will deliver some of the important background for Coursework 1. Below you will find some of the relevant resources required for this practical.

Parts (sections 2,3,4 and 5) of this practical are written by Natalie Cooper. The original can be found here

The data used throughout the practical can be downloaded from Blackboard or from here and here (right click and `save link as`).

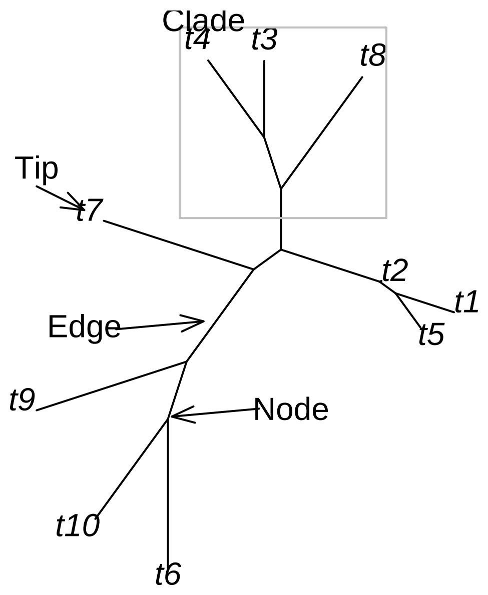
Further information can be found in Liam Revell's book chapter here.

2. A refresher of phylogenetic trees

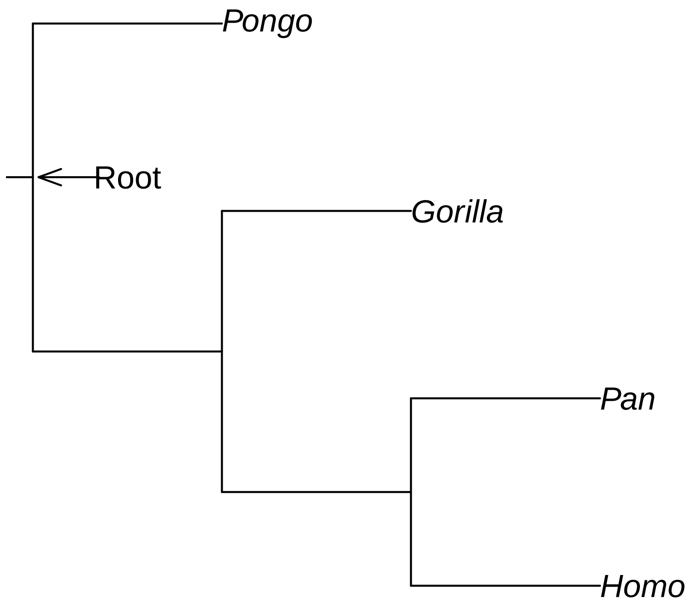
This section will review some basic aspects of phylogenetic trees and introduce how trees are handled at the level of software. Because you are now interacting with phylogenetic trees at a 'lower level' (i.e. at the bioinformatics level), it is also helpful to know some of the names for parts of phylogenetic trees used in computer science.

A. Tree parameters

A phylogenetic tree is an ordered, multifurcating graph with labeled **tips** (or **leaves**) (and sometimes labeled histories). It represents the relative degrees of relationships of species (i.e. tips or OTUs). The graph consists of a series of **branches** (or **edges**) with join successively towards **nodes** (or **vertices**, *sing.* **vertex**). Each node is subtended by a single branch, representing the lineage of ancestors leading to a node. The node is thus the common ancestor of two or more descendant branches. All the descendant branches of a given node (and all of the their respective descendants) are said to form a **clade** (or **monophyletic group**).

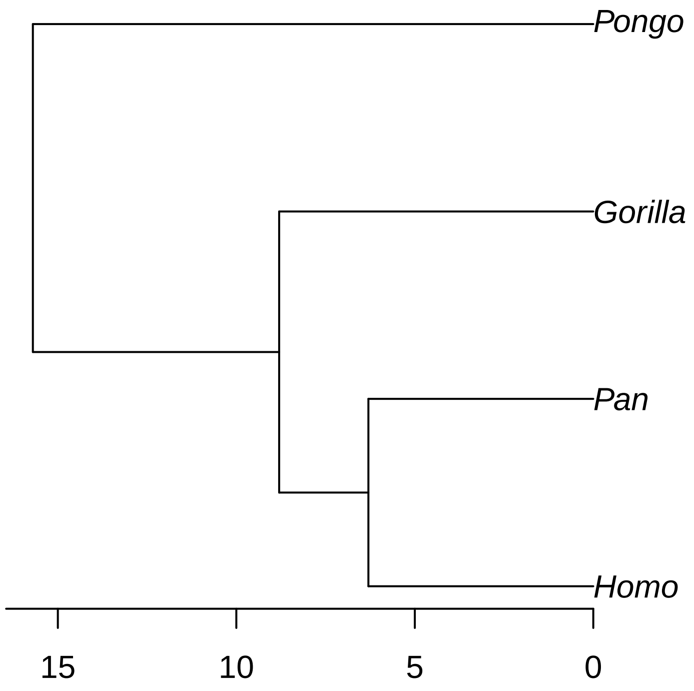


When we select a node to act as the base of a tree, the tree is said to be **rooted**. At the bottom of a tree, is the **root node** (or simply the **root**).



Phylogenetic trees of the kind shown above are fairly simple and lack information about time or character changes occurring along a branch. We can assign branch length in the form of either time or the amount of change/substitution along a branch. A tree with **branch lengths** depicted can be called a **phylogram**.

When (an implied) dimension of time is being considered, all the tips of the tree must be at the level representing the time in which they are observed. For trees where all the species are extant, the tips are flush at the top. This representation is called an **ultrametric** tree.



B. Informatic representations of tree

To perform any useful calculations on a tree, we need both a computer-readable tree format and (in part) to understand how trees are constructed in computer memory.

Text based formats

Storage of trees for transfer between different software is essential. This is most commonly achieved with a text-based format stored in a file. The most common file format for representing phylogenetic trees is **Newick format**. This consists of clades represented within parentheses. Commas separate each clade. Either tip names or symbols representing the tips are nested within the lowest orders of parentheses. Each tip or branch can be associated with a branch length scalar that follows a colon.

For example:

```
"(((Homo, Pan), Gorilla), Pongo);"
```

Or with branch length:

```
"(((Homo:6.3, Pan:6.3):2.5, Gorilla:8.8):6.9, Pongo:15.7);"
```

Trees are also increasing use of XML formats such as PhyloXML and NeXML.

Edge table

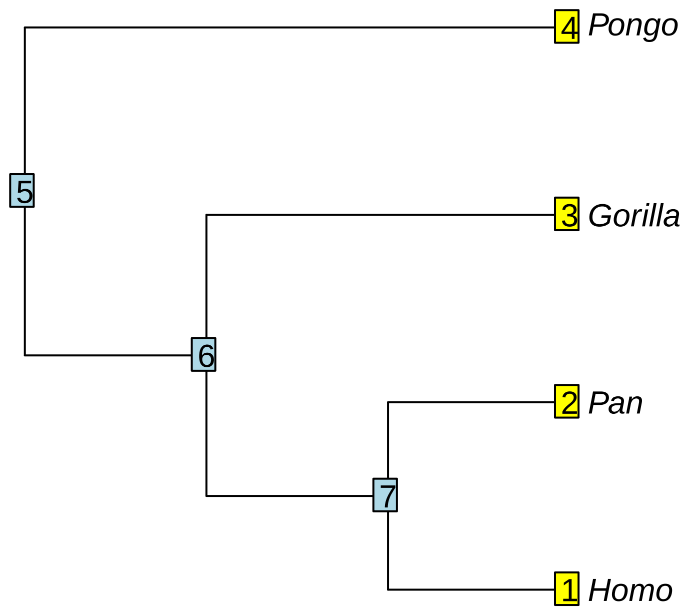
It is also possible to represent a phylogenetic tree as a matrix of edges and vertices called an edge table. This is an even less intuitive representation, but it is implemented in R and worth reviewing here.

There are a number of conventions that can be used to create an edge table. The general concept consists of numbering the tips *1 - n*, and all internal nodes labeled *n+1 ... n+n-1*. The numbers for the internal nodes can be assigned arbitrarily or according to an algorithm.

In `R` packages like `ape`, edge tables are constructed as follows:

node	connects to
5	6
6	7
7	1
7	2
6	3
5	4

You read the table as follows: node `5` (root) connects to node `6`. The node `6` connects to node `7`. Node `7` connects to node `1` that happen to be the first tip (`Homo`) and to node `2` (`Pan`) etc... Note that in a binary tree (i.e. a tree where each node has only two descendants) each node always connects to two elements (nodes or tips).



Records & pointers

At a lower level, phylogenetic trees can be represented in computer memory as more complex data objects. We don't need to go into detail here, but if you consider nodes and tips as data objects (i.e. a dataframe), a tree could be stored as an array of dataframes which store information about which store information about which members of that same array are descendants and which are ancestors.

2. Installing and loading extra packages in `R`

To plot phylogenies (or use any specialized analysis) in R, you need to download one or more additional packages from the basic R installation. For this practical you will need to install the following packages:

- `ape`
- `phytools`

To install the package `ape`:

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

Pick the closest mirror to you if asked. Now install `phytools`.

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)
```

You can think of `install.packages` like installing an app from the App Store on your smart phone - you only do this once - and `library` as being like pushing the app button on your phone - you do this every time you want to use the app.

Don't forget to load `phytools` too!

```
library(phytools)
```

3. Loading your phylogeny and data into R

Reading in a phylogeny from a file

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. `elopomorph.tre` is not in NEXUS format so we use `read.tree`.

```
fishtree <- read.tree("elopomorph.tre")
```

> Be sure you are always in the right directory. Remember you can navigate in `R` using `setwd()`, `getwd()` and `list.files()` (to see what's in the current directory).

Reading in a phylogeny that is already built into R

The bird and anole phylogenies are already built into R so we don't need to read them in using `read.tree`. Instead we just use:

```
data(bird.orders)
data(anoletree)
```

Reading and viewing your data in R

Later we will use some Greater Antillean *Anolis* lizard data to add data to a phylogeny. Before we can add data to our tree, we need to load the data we are going to use. 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 I have given you a comma-delimited text file called `anole.data.csv` which we are going to use. Load these data as follows. I am assuming you have set your working directory, if not don't forget the path.

```
anoledata <- read.csv("anole.data.csv", header = TRUE)
```

You can 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(anoledata)
```

```
## 'data.frame':    100 obs. of  23 variables:
##  $ species              : Factor w/ 100 levels "ahli","alayoni",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ AVG.SVL              : num  4.04 3.82 3.53 4.04 4.38 ...
##  $ AVG.hl               : num  2.88 2.7 2.38 2.9 3.36 ...
##  $ AVG.hw               : num  2.36 1.99 1.56 2.37 2.69 ...
##  $ AVG.hh                 : num  2.13 1.75 1.39 2.05 2.32 ...
##  $ AVG.ljl                : num  2.85 2.71 2.32 2.9 3.38 ...
##  $ AVG.outlever           : num  2.75 2.62 2.26 2.83 3.29 ...
##  $ AVG.jugal.to.symphysis: num  2.54 2.37 2.08 2.6 3.07 ...
##  $ AVG.femur              : num  2.74 2.07 2.17 2.48 2.8 ...
##  $ AVG.tibia              : num  2.69 2.02 2.09 2.34 2.69 ...
##  $ AVG.met                : num  2.25 1.54 1.55 1.87 2.18 ...
##  $ AVG.ltoe.IV            : num  2.55 1.88 1.73 2.26 2.53 ...
##  $ AVG.toe.IV.lam.width  : num  0.1795 0.0488 -0.5361 0.4904 0.8441 ...
##  $ AVG.humerus            : num  2.46 1.95 1.63 2.3 2.62 ...
##  $ AVG.radius             : num  2.27 1.69 1.4 2.09 2.34 ...
##  $ AVG.lfing.IV           : num  1.94 1.4 1.04 1.7 1.98 ...
##  $ AVG.fing.IV.lam.width : num  0.0754 -0.0739 -0.755 0.3155 0.6584 ...
##  $ AVG.pelv.ht            : num  1.99 1.51 1.19 1.87 2.1 ...
##  $ AVG.pelv.wd            : num  1.71 1.419 0.946 1.752 2.014 ...
##  $ Foot.Lam.num           : num  3.28 3.43 3.2 3.58 3.72 ...
##  $ Hand.Lam.num           : num  2.87 3.08 2.73 3.16 3.24 ...
##  $ Avg.lnSVL2             : num  3.94 3.74 3.48 3.93 4.36 ...
##  $ Avg.ln.t1              : num  4.41 4 4.37 4.44 5.04 ...
```

Always look at your data before beginning any analysis to check it read in correctly.

`str` 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.

```r
head(anoledata)
```

```
##    species  AVG.SVL   AVG.hl   AVG.hw   AVG.hh  AVG.ljl AVG.outlever
## 1     ahli 4.039125 2.882657 2.356126 2.131599 2.854745     2.754934
## 2  alayoni 3.815705 2.702116 1.990610 1.751588 2.708966     2.617852
## 3  alfaroi 3.526655 2.378156 1.556037 1.390037 2.323857     2.263844
## 4 aliniger 4.036557 2.898836 2.366592 2.046660 2.903946     2.828555
## 5 allisoni 4.375390 3.358957 2.690339 2.317309 3.381306     3.288402
## 6  allogus 4.040138 2.861027 2.351275 2.142107 2.848331     2.750404
##   AVG.jugal.to.symphysis AVG.femur AVG.tibia  AVG.met AVG.ltoe.IV
## 1               2.538052  2.741378  2.686032 2.254620    2.553344
## 2               2.371995  2.065121  2.016402 1.535253    1.875258
## 3               2.077565  2.172476  2.094946 1.547563    1.732540
## 4               2.596821  2.475445  2.344015 1.867176    2.256541
## 5               3.071149  2.795145  2.687734 2.178155    2.533697
## 6               2.539320  2.739175  2.684476 2.217314    2.494900
##   AVG.toe.IV.lam.width AVG.humerus AVG.radius AVG.lfing.IV
## 1           0.17953991    2.460728   2.268511     1.943288
## 2           0.04879016    1.947873   1.687093     1.403029
## 3          -0.53614343    1.628260   1.401183     1.040277
## 4           0.49041881    2.298878   2.091864     1.702199
## 5           0.84407840    2.618551   2.341565     1.983412
## 6           0.19625907    2.459994   2.267176     1.919010
##   AVG.fing.IV.lam.width AVG.pelv.ht AVG.pelv.wd Foot.Lam.num Hand.Lam.num
## 1            0.07541664    1.987646   1.7095849     3.279600     2.866197
## 2           -0.07391568    1.507128   1.4194873     3.432372     3.075269
## 3           -0.75502258    1.189367   0.9458495     3.198016     2.733866
## 4            0.31554040    1.867949   1.7521520     3.582875     3.156774
## 5           0.65838319    2.097302   2.0142361     3.721185     3.239211
## 6           0.05329130    2.008426   1.6875679     3.335990     2.808270
##   Avg.lnSVL2 Avg.ln.t1
## 1   3.939015  4.406652
## 2   3.743863  4.002788
## 3   3.478776  4.369448
## 4   3.933181  4.441203
## 5   4.355582  5.039851
## 6   4.028863  4.510920
```

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

```
names(anoledata)
```

```
##  [1] "species"                "AVG.SVL"
##  [3] "AVG.hl"                 "AVG.hw"
##  [5] "AVG.hh"                 "AVG.ljl"
##  [7] "AVG.outlever"           "AVG.jugal.to.symphysis"
##  [9] "AVG.femur"              "AVG.tibia"
## [11] "AVG.met"                "AVG.ltoe.IV"
## [13] "AVG.toe.IV.lam.width"   "AVG.humerus"
## [15] "AVG.radius"             "AVG.lfing.IV"
## [17] "AVG.fing.IV.lam.width"  "AVG.pelv.ht"
## [19] "AVG.pelv.wd"            "Foot.Lam.num"
## [21] "Hand.Lam.num"           "Avg.lnSVL2"
## [23] "Avg.ln.t1"
```

This gives you the names of the columns.

```
anoledata
```

This will print out all of the data!

4. Basic tree viewing in R

Now let's visualise some phylogenies! We'll use the Elopomorpha (eels and similar fishes) tree to start as it is simple.

```
fishtree <- read.tree("elopomorph.tre")
```

Let's examine the tree by typing:

```
fishtree
```

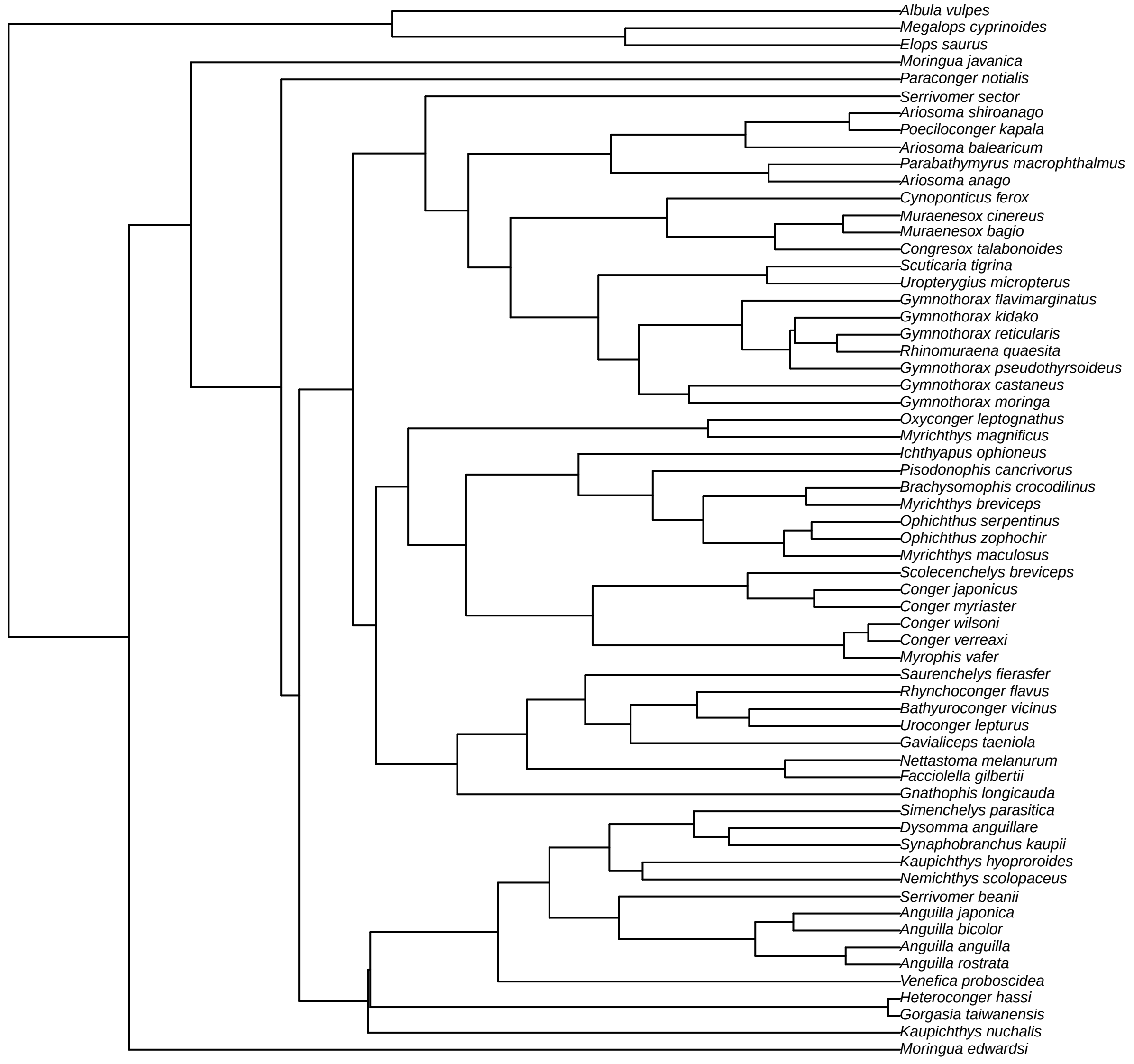
```
##
## Phylogenetic tree with 62 tips and 61 internal nodes.
##
## Tip labels:
##  Moringua_edwardsi, Kaupichthys_nuchalis, Gorgasia_taiwanensis, Heteroconger_hassi, Venefica_probosc
##
## Rooted; includes branch lengths.
```

```r
str(fishtree)
```

```
## List of 4
##  $ edge       : int [1:122, 1:2] 63 64 64 65 66 67 68 68 69 70 ...
##  $ Nnode      : int 61
##  $ tip.label  : chr [1:62] "Moringua_edwardsi" "Kaupichthys_nuchalis" "Gorgasia_taiwanensis" "Hetero
##  $ edge.length: num [1:122] 0.0105 0.0672 0.00537 0.00789 0.00157 ...
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

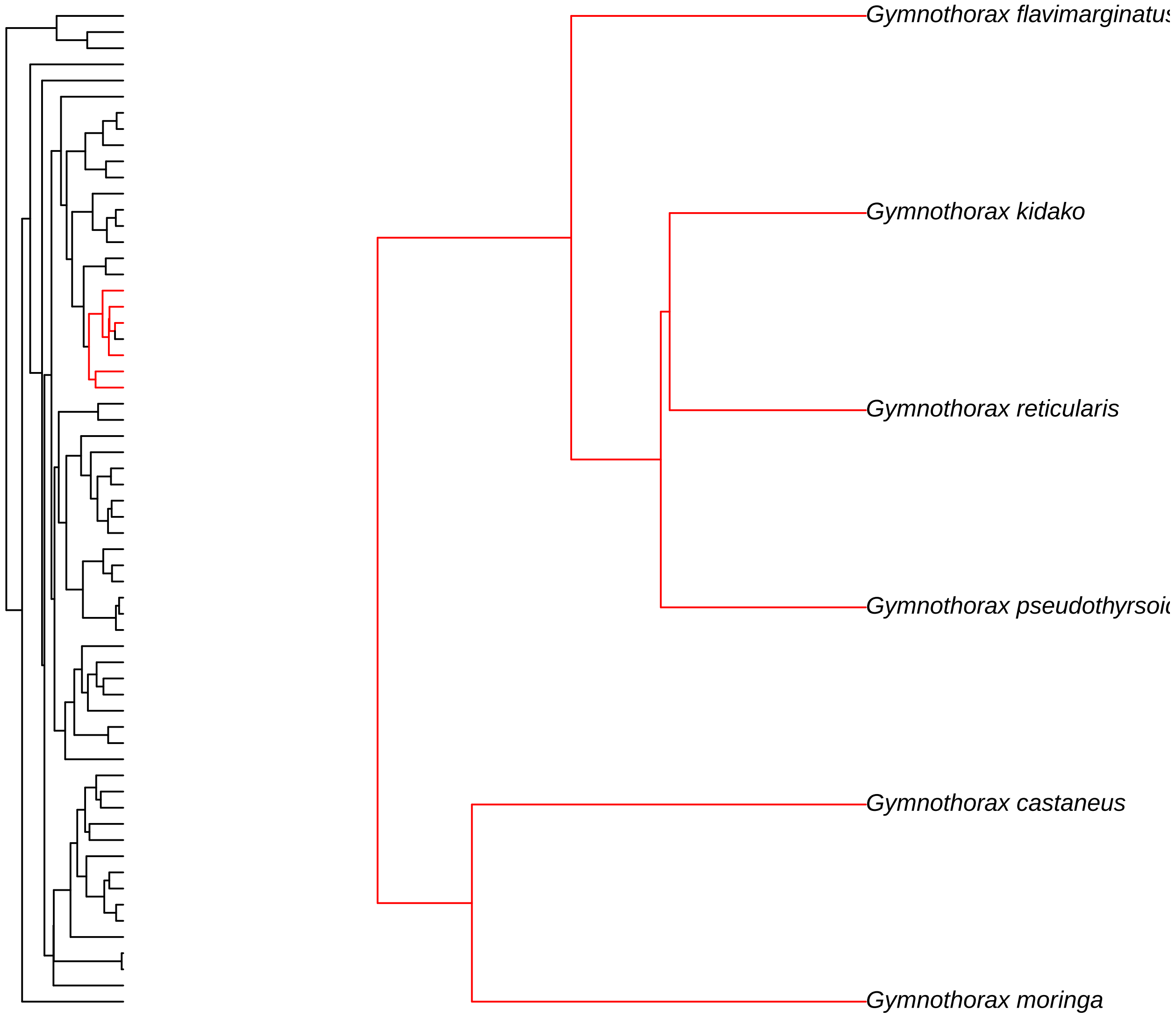
`fishtree` is a fully resolved tree with branch lengths. There are 62 species and 61 internal nodes. We can plot the tree by using the `plot.phylo` function of `ape`. Note that we can just use the function `plot` to do this as R knows if we ask it to plot a phylogeny to use `plot.phylo` instead!

```r
plot(fishtree, cex = 0.5)
```



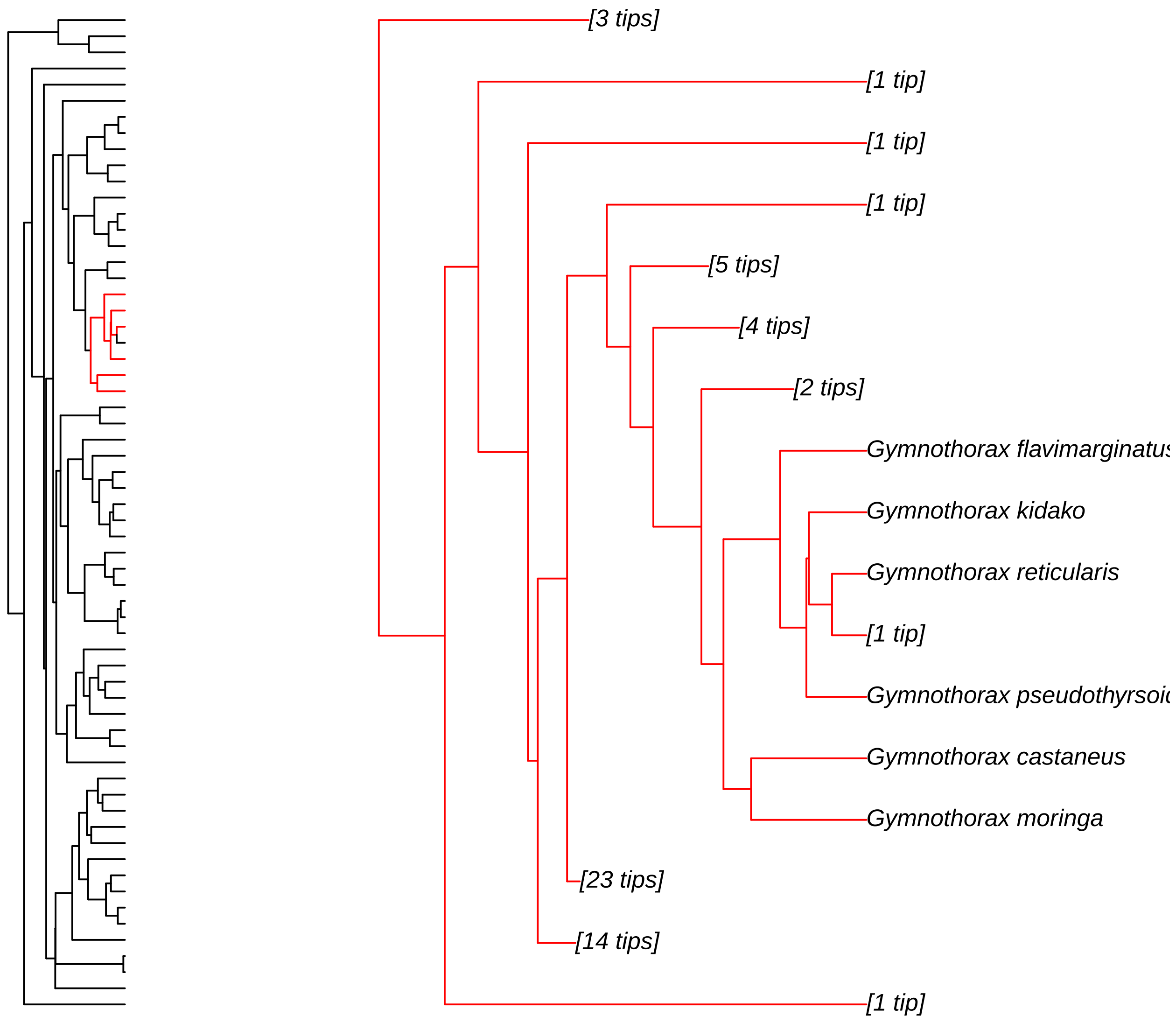
`cex = 0.5` reduces the size of the tip labels so we can read them. We can also zoom into different sections of the tree that you're interested in:

```r
zoom(fishtree, grep("Gymnothorax", fishtree$tip.label), subtree = FALSE, cex = 0.8)
```



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

```r
zoom(fishtree, grep("Gymnothorax", fishtree$tip.label), subtree = TRUE, cex = 0.8)
```



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

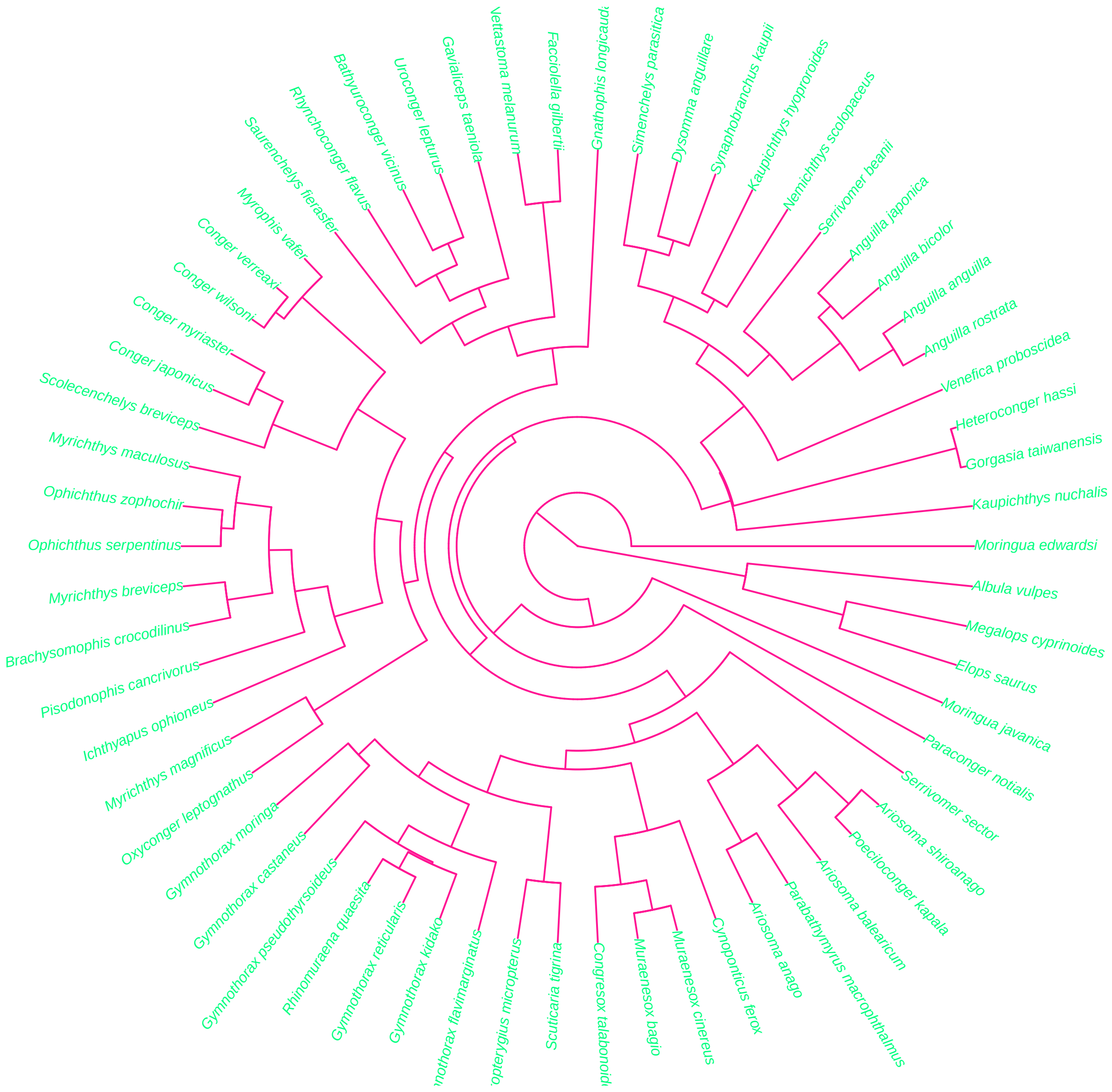
```
par(mfrow = c(1, 1))
```

To get further options for the plotting of phylogenies:

```
?plot.phylo
```

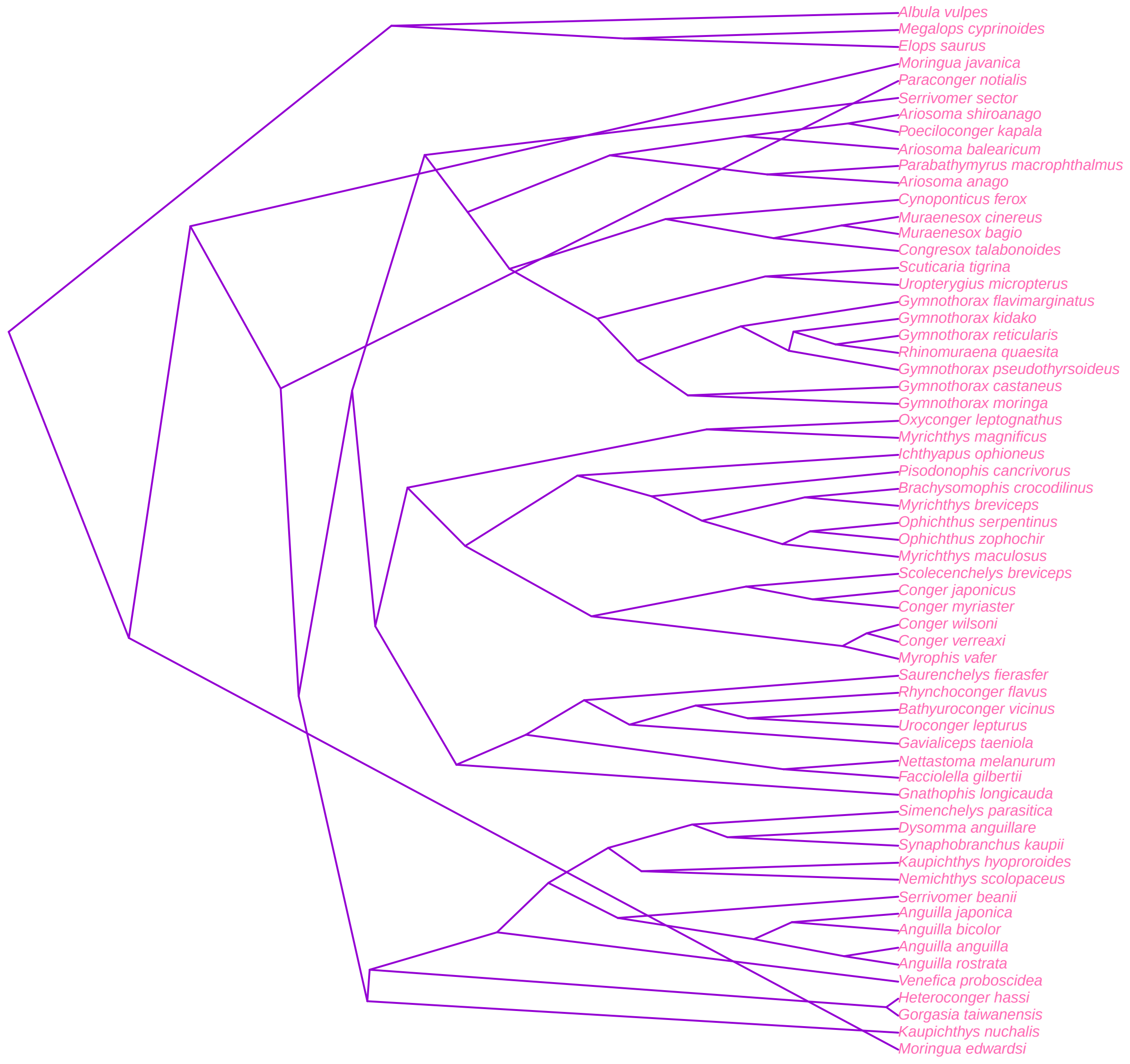
Note that although you can use `plot` to plot the phylogeny, you need to specify `plot.phylo` to find out the options for plotting trees. You can change the style of the tree (`type`), the color of the branches and tips (`edge.color`, `tip.color`), and the size of the tip labels (`cex`). Here's an fun/hideous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

```
plot(fishtree, type = "fan", edge.color = "deeppink", tip.color = "springgreen",
     cex = 0.5)
```



Or try

```
plot(fishtree, type = "c", edge.color = "darkviolet", tip.color = "hotpink",
     cex = 0.5)
```



5. Adding trait data to trees in R

A. Ancestral state reconstructions on discrete data

We will use the bird data Remember we already loaded the phylogeny and data as follows:

```
data(bird.orders)
```

First we will invent some data for each bird order that we can reconstruct along the tree. This creates three variables, 0, 1 and 2.

```
mydata <- factor(c(rep(0, 5), rep(1, 13), rep(2, 5)))
mydata
```

```
##  [1] 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
## Levels: 0 1 2
```

We can then use the `ape` function `ace` to reconstruct ancestral characters along the nodes of the tree. `type = d` means the character to be reconstructed is discrete.

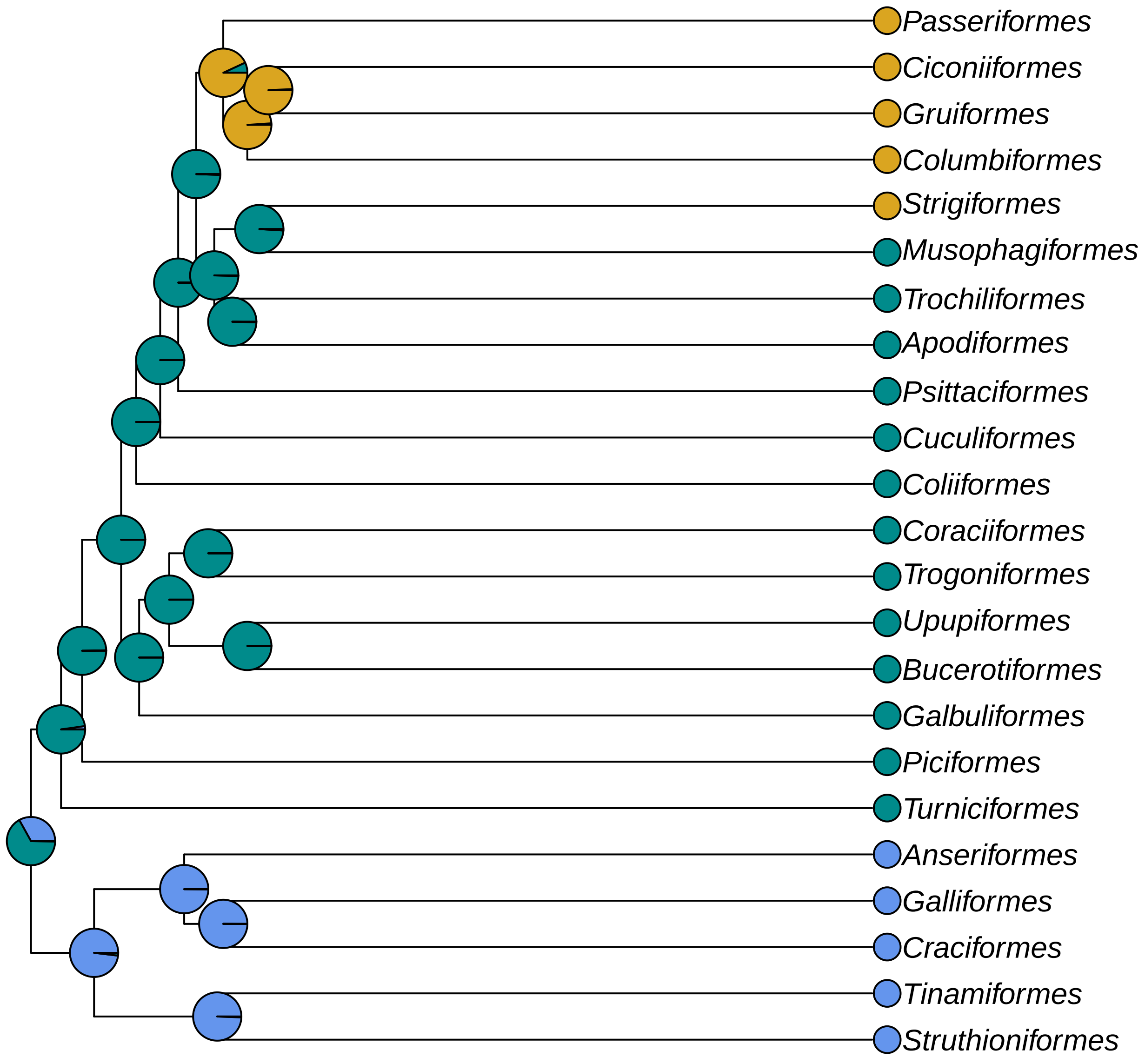
```
ancestors <- ace(mydata, bird.orders, type = "d")
```

Now we can plot this on a phylogeny. First decide which colours we'd like. To look at a list of colours in R type in `colors()`.

```
colours <- c("cornflowerblue", "cyan4", "goldenrod")
```

Now plot the tree and add labels to the tips and the nodes using the results in `ancestors`. We use `label.offset = 1` to move the labels to the right a bit so the pie charts will fit...

```
plot(bird.orders, label.offset = 1)
tiplabels(pch = 21, bg = colours[as.numeric(mydata)], cex = 2, adj = 1)
nodelabels(pie = ancestors$lik.anc, piecol = colours)
```



`pch = 21` sets the tip labels to be unfilled circles, `bg` defines the colours of the circles using the list of colours we provided, and ordering them based on what the species value was for mydata (i.e. 0, 1 or 2). `cex = 2` doubles the point size, and `adj = 1` moves the tip labels sideways so they don't obscure the ends of the branches. `pie` makes pie charts coloured using the ancestral state reconstructions in `ancestors`, and `piecol` tells it to use the colours we have defined.

B. Ancestral state reconstructions on continuous data

We are going to use the *Anolis* data to create a phylogeny with different colours for different observed and reconstructed body sizes (snout-to-vent length, SVL). Remember we already loaded the phylogeny and data as follows:

```
data(anoletree)
anoledata <- read.csv("anole.data.csv", header = TRUE, row.names = 1)
```

Note the names in `anoledata` are the species names without the Genus. In the phylogeny the species names are `Anolis_species`. So to get the two to match we need to add `Anolis_` to each name.

```
rownames(anoledata) <- paste("Anolis", rownames(anoledata), sep = "_")
```

`paste` just sticks together `Anolis` with the names in `anoles` already with an underscore (_) separating them (`sep = "_"`)

We then need to make sure the order of the species in the data matches that of the phylogeny.

```
anoledata <- anoledata[anoletree$tip.label, ]
```

Next we make a matrix containing only the SVL values for each *Anolis* species:

```
SVL <- as.matrix(anoledata)[,"AVG.SVL"]
```

This code selects only the variable `AVG.SVL` from `anoledata` (square brackets subset in R in the form [rows, columns]), and then uses `as.matrix` to make this into a matrix.

Take a look at `SVL`

```
head(SVL)
```

```
##        Anolis_ahli     Anolis_allogus Anolis_rubribarbus
##           4.039125           4.040138           4.078469
##        Anolis_imias      Anolis_sagrei     Anolis_bremeri
##           4.099687           4.067162           4.113371
```

We then use the function `contMap`. `contMap` creates a tree with a mapped continuous character, i.e. where the value of the character is known at the tips, and estimated along the tree. The estimating of the character along the tree uses a Maximum Likelihood estimation procedure. Here we will tell `contMap` not to automatically plot the tree (using `plot = FALSE`) so we can make some modifications.

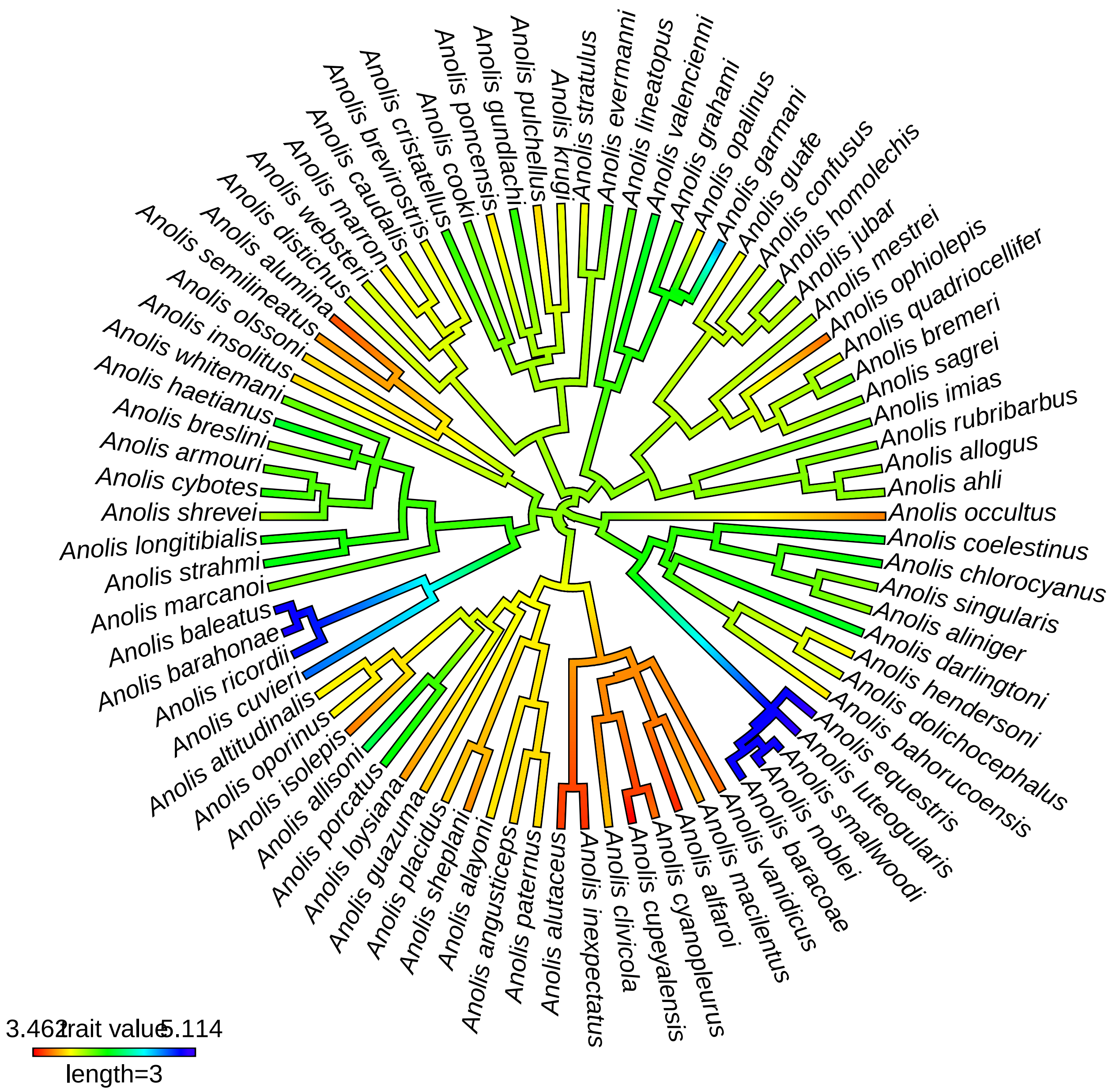
```
SVLplot <- contMap(anoletree, SVL, plot = FALSE)
```

For some reason this code breaks…

Finally let's plot the tree as a fan (`legend = 10` just spreads the legend out so it is readable).

```r
plot(SVLplot, type = "fan", legend = 10)
```

```
## legend scale cannot be longer than total tree length; resetting
```


Anolis occultus
Anolis coelestinus
Anolis chlorocyanus
Anolis singularis
Anolis aliniger
Anolis darlingtoni
Anolis hendersoni
Anolis dolichocephalus
Anolis bahorucoensis
Anolis equestris
Anolis luteogularis
Anolis smallwoodi
Anolis noblei
Anolis baracoae
Anolis vanidicus
Anolis macilentus
Anolis alfaroi
Anolis cyanopleurus
Anolis cupeyalensis
Anolis clivicola
Anolis inexpectatus
Anolis alutaceus
Anolis paternus
Anolis angusticeps
Anolis alayoni
Anolis sheplani
Anolis placidus
Anolis guazuma
Anolis loysiana
Anolis porcatus
Anolis allisoni
Anolis isolepis
Anolis oporinus
Anolis altitudinalis
Anolis cuvieri
Anolis ricordii
Anolis barahonae
Anolis baleatus
Anolis marcanoi
Anolis strahmi
Anolis longitibialis
Anolis shrevei
Anolis cybotes
Anolis armouri
Anolis breslini
Anolis haetianus
Anolis whitemani
Anolis insolitus
Anolis olssoni
Anolis semilineatus
Anolis alumina
Anolis distichus
Anolis websteri
Anolis marron
Anolis caudalis
Anolis brevirostris
Anolis cristatellus
Anolis cooki
Anolis poncensis
Anolis gundlachi
Anolis pulchellus
Anolis krugi
Anolis stratulus
Anolis evermanni
Anolis lineatopus
Anolis valencienni
Anolis grahami
Anolis opalinus
Anolis garmani
Anolis guafe
Anolis confusus
Anolis homolechis
Anolis jubar
Anolis mestrei
Anolis ophiolepis
Anolis quadriocellifer
Anolis bremeri
Anolis sagrei
Anolis imias
Anolis rubribarbus
Anolis allogus
Anolis ahli
trait value
length=3