RBrownie User's Guide (Version 0.0.4)

J. Conrad Stack with credit due to others(1)

August 27, 2010

1 Brief Introduction to Brownie

`RBrownie` is an R package which is constructed around the C++ command-line program Brownie (available here) designed by Brian O'Meara to estimate and compare rates and means of character evolution in different parts of a phylogeny (O'Meara, et al., 2006). More specifically, the original Brownie function was a censored rate test under Brownian motion, which allows the comparison of the rates of evolution (under brownian motion) across two monophyletic sister-clades or between a monophyletic and paraphyletic clade. This analysis allows you to answer questions about differences in the evolutionary rate of two distinct clades (e.g. large subspecies vs. small subspecies).

As Brownie has grown, more, related analyses have been added. Now you can perform non-censored ratetests and discrete character reconstructions. The former allows you to compare the rates (Brownian motion model) or means (Ornstein-Uhlenbeck model) of evolution in different character states that are mapped (or painted) onto the phylogeny. For example, say you had a tree with binary characters mapped onto it branches indicating whether or not ancestors along that branch were thought to have or not have a certain trait and you also had morphological data for all the taxa present in the tree (the tips). Using a model-testing approach, Brownie allows to you statistically assess whether or not that binary trait might have significantly influence the rate of evolution of the morphological characteristic measured (or set of morphological characteristics). Brownie also performs discrete ancestral state evolution (maximum likelihood approach), in order to get the branch-annotated trees which are used in non-censored rate tests.

Examples of the sort of analyses available in `RBrownie` are shown and described in detail below, so don't worry if this does not make sense yet. There is also a fairly technical explaination of the data types found in `RBrownie` which it uses to handle, manipulate, and visualize phylogenetic data and test results and instructions on how to create your own custom analyses.

1.1 How to run

Over it's lifetime Brownie has evolved from a MATLAB module into a command-line executable available to anyone. In it's current iteration, Brownie 2.0, is available in executable form for Macintosh operating systems and as source code which can be installed onto most other operating systems using gcc and the GSL.

1.2 Input / Output

The brownie command-line program is very similar in look, feel, and structure to command-line PAUP*. It reads NEXUS-formatted files including, like PAUP*, a special (optional) nexus block, 'BROWNIE', which holds a list of the commands to be executed by the program itself. This is noteworthy because `RBrownie` reads and writes these specialized nexus files and any files that `RBrownie` outputs should be directly executable by the Brownie command-line program.

1.3 More information

Brownie has a large number of function in addition to the rate tests described above that we encourage you to check out. This will allow you to exploit all the useful functions Brownie (and by extension `RBrownie`) have to offer. the manual for Brownie can be found here.

2 RBrownie

It was mentioned before that RBrownie is an R shell for the brownie program described above. This is indeed the case. Using the `Rcpp` package, RBrownie links R with the exact same Brownie code that is used in the Brownie command line program. It simply builds brownie as a static library and automates the piping and execution of strings that would normally be entered at the Brownie programs command line. This is the reason that the `brownie` S4 class has a slots for "commands" (more on this later). At it's heart, RBrownie simply takes all the information from a `brownie` object, writes it to a file, and then executes it using Brownie.

Of course to be able to process specially formatted nexus files and manipulate their contents from within R, RBrownie had to extend R's already rich phylogenetic toolkit with a number of data structures and methods. It literally extends a number of classes and methods from `phylobase`, an R package, it could be said, on which RBrownie is based. Loading the RBrownie package should show you a list of all the packages on which it is based.

```
> require(RBrownie)
```

2.1 Reading nexus files

Nexus files can be read into R using a number of packages. `ape` and `phylobase` both include the functionality to read in tree and sequence data, while phylobase also reads CHARACTERS blocks. For more information on the options presented by these too packages, check out their documentation files:

```
> `?`(read.nexus)
> `?`(read.nexus.data)
> `?`(readNexus)
```

However, Brownie uses nexus blocks beyond those supported by `ape` and `phylobase` and so `RBrownie` extends the methods from these two packages to accomadate. For example, Brownie uses an ASSUMPTIONS block to define subsets of the taxa which can be used in its various rate tests. It also uses a BROWNIE block to store a list of brownie commands to be run automatically if the file is "executed" within the Brownie environment (this is similar to how PAUP* executes files). So, RBrownie has it's own functions for reading and writing nexus files with these special blocks, `readBrownie` and `writeBrownie`.

To illustrate how they work we'll use the parrot dataset which is included with RBrownie (NOTE: you need to have write access to the directory you are in to run this example).

```
> data(parrot)
> writeBrownie(parrot, file = "parrotdata.nex")
> newparrot = readBrownie("parrotdata.nex")
```

The two objects `parrot` and `newparrot` should be indentical. It would also be instructive to open the parrotdata.nex file to see how RBrownie writes ASSUMPTIONS blocks, but this isn't necessary. You may have noticed that `parrot` and `newparrot` are both of class `list`.

```
> class(parrot)
> class(parrot[[1]])
```

This is because when multiple trees are found in a single nexus file, RBrownie represents them as a list of objects which all are of class `brownie` - these objects are discussed in the next section.

2.2 Classes in RBrownie

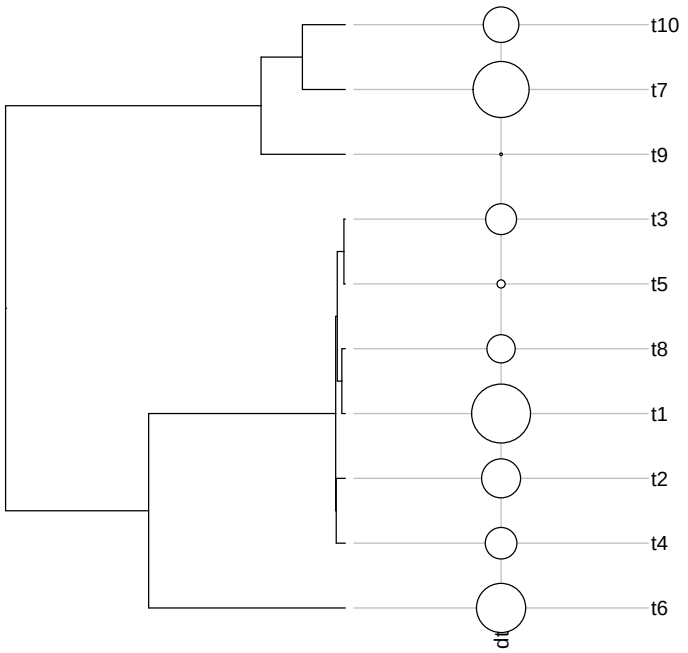
At it's core RBrownie is built around two major classes, `phylo4d_ext` and `brownie`, which both extend `phylo4d` from phylobase. In fact, `phylo4d_ext` extends `phylo4d` and `brownie` extends `phylo4d_ext`. In non-computer-ese this mean that `phylo4d_ext` adds a few new data containers (or "slots" as they are known in the S4 world) and `brownie` adds a few new data containers to `phylo4d_ext`. To illustrate this, look at the R code below, it shows the new slots that are added to `phylo4d_ext`.

```
> phylo4d_slots = names(getSlots("phylo4d"))
> phylo4d_ext_slots = names(getSlots("phylo4d_ext"))
> brownie_slots = names(getSlots("brownie"))
> phylo4d_slots
> setdiff(phylo4d_ext_slots, phylo4d_slots)
> setdiff(brownie_slots, phylo4d_ext_slots)
```

2.2.1 phylo4d_ext class

This class (as implied by the name) is an extension of the `phylo4d` class from phylobase. This base class holds a phylogenetic tree and a `data.frame` containing data for each node of the tree. The code below shows how to construct a phylogenetic tree and add junk data to the tips of tree which in practice might represent morphological data for each of the taxa.

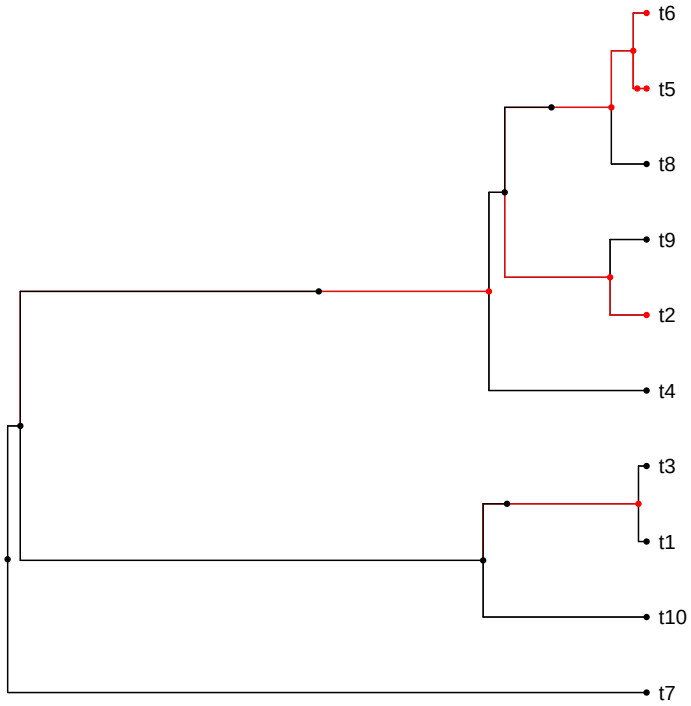
```
> require(phylobase)
> # generate a random coalescent tree with 10 tips
> ape_tree = rcoal(10)
> # convert tree from 'phylo' (ape) to 'phylo4' (phylobase) format
> phy_tree = as(ape_tree,"phylo4")
> # generate junk data for the tips
> sample_tipdata = runif(10)
> # combine the phylogenetic tree and the sample data
> phyd_tree = phylo4d(phy_tree,tip.data=sample_tipdata)
> plot(phyd_tree)
```



phylo4d_ext adds to this class 'subnodes' and tree weights. Subnodes are internal nodes mapped directly onto an existing branches and are conceptually similar to singleton nodes (but offer a bit more flexibility from a programming standpoint). The phylo4d_ext class was created mainly to handle SIMMAP-formatted nexus trees, but instances can be created from scratch as well.

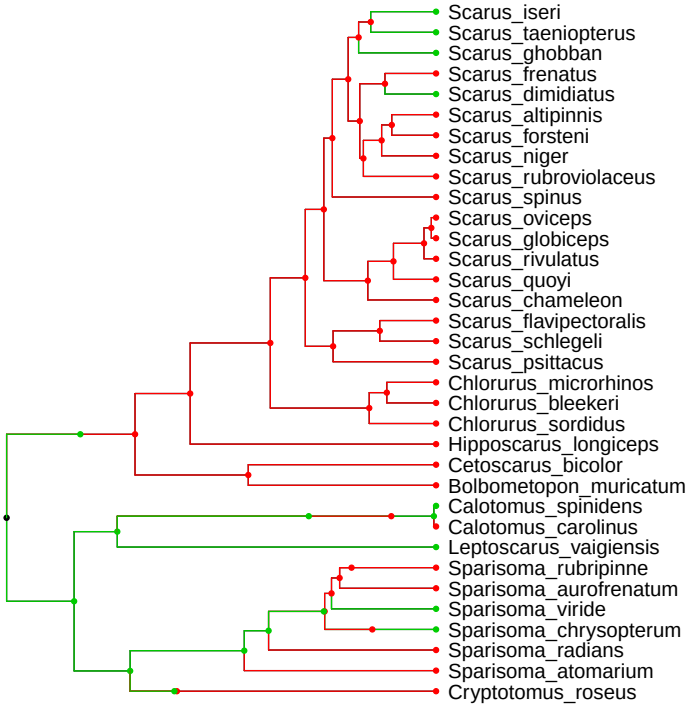
```
> require(RBrownie)
> ape_tree = rcoal(10)
> phy_tree = as(ape_tree,"phylo4")
> # create tip data
> # create binary data indicators for all nodes (including internal)
> sample_binarydata = data.frame(sample(c(0,1),19,replace=TRUE))
> names(sample_binarydata) <- "hasTrait"
> # subnodes
> sample_subnodedata = data.frame(sample(c(0,1),4,replace=TRUE))
> names(sample_subnodedata) <- "hasTrait"  # should be the same name as the parent data.fram
> sample_edges = sample(seq(nrow(edges(phy_tree))),4) # add subnodes to 4 random branches
> sample_positions = runif(4) # could positions for the subnodes along their branches (as a
> # note that "extra" arguments (all.data) are passed on to the phylo4d constructor
> phyext_tree = phyext(phy_tree,
+                          snode.data=sample_subnodedata,
+                          snode.branch=edges(phy_tree)[sample_edges,],
+                          snode.pos=sample_positions,
```

```
+                          all.data=sample_binarydata)
> # nodes, subnodes, and branches without the "trait" (state 0) are colored black
> # nodes, subnodes, and branches with the "trait" (state 1) are colored red
> plot(phyext_tree,states=c(0,1),states.col=c("black","red"))
```



In the plot above, the subnodes are clearly seen as dots on an existing branch. This is usually used to indicate state changes along a branch, but because of the randomly generated data above you might see an entirely red branch with a red subnode or something similarly odd. A better example might be seen in the parrot data set.

```
> require(RBrownie)
> data(parrot)
> plot(parrot[[1]])
```



Now the binary state data are represented by the default colors and a state change is clearly visible on the branch from the taxon Sparisoma_chrysopterum. In this case, state 0 and state 1 represent a life history trait of parrot fish, namely whether they scrape (0) or excavate (1) coral. You can clearly see that, in this tree at least, that life history trait changes throughout the evolutionary history of the parrot fish at both internal nodes and along a branch at a subnode. Brownie uses this state information, as we will see later, to address questions such as "Does coral excavatingscraping limit morphological evolution in parrotfishes" (Bodega Phylogenetics Wiki , 2010)

Anyhow, to view subnode information these accessor functions are available.

```
> snid(phyext_tree)
> sndata(phyext_tree)
> snposition(phyext_tree)
> snbranch(phyext_tree)
> showSubNodes(phyext_tree)
```

And to add a new subnode manually, try this

```
> # place the node along the first branch
> ancestor.node = edges(phyext_tree)[1,1]
> descendant.node = edges(phyext_tree)[1,2]
> node.position = 0.5 # half way along the branch
```

```
> node.data = data.frame(1) # give it state = 1
> names(node.data) <- "hasTrait"
> phyext_tree = addSubNode(phyext_tree,
+                                  ancestor.node,
+                                  descendant.node,
+                                  node.position,
+                                  node.data)
```

You might notice that the subnode position is represented currently as a two-column matrix. This is because, in a future version, the subnodes will float between two values representing our confidence on where the state change (and thus the subnode) might occur. `weight` is another aspect of the `phylo4d_ext` class that will be supported better in future versions.

Finally, subnodes will most be read in from SIMMAP-formatted nexus files and not added manually. The SIMMAP format was created represent state mappings onto branches (subnodes) in nexus tree files and the standard has gone through a number of iterations. Currently, only SIMMAP version 1.0 and 1.1 are supported by `RBrownie` - we are working to add support for others. `RBrownie` has functions for reading and / writing in this format with functions similar to `ape`'s `read.nexus` and `write.nexus` functions. (NOTE: you need to have write access to the directory you are in to run this example).

```
> data(parrot)
> write.nexus.simmap(parrot, file = "parrottmp.nex")
> newparrot = read.nexus.simmap("parrottmp.nex")
```

2.2.2 brownie class

References

Price, S., Wainwright, P. (2010). Testing for different rates of continuous trait evolution using likelihood. (http://bodegaphylo.wikispot.org/Morphological_Diversification_and_Rates_of_Evolution)

O'Meara, B. C., Ane, C., Sanderson, M. J. and Wainwright, P. C. (2006). Testing for different rates of continuous trait evolution using likelihood. In *Evolution*, 60 ,922–933.