An introduction to the pmc Package

Carl Boettiger[a,*]

[a] *Center for Population Biology, University of California, Davis, United States*

Abstract

The Phylogenetic Monte Carlo package provides a suite of utilities for more robust model choice and uncertainty analysis in comparative phylogenetic models. This vignette accompanies the manuscript by Boettiger et al, and describes how to use the package by illustrating how each of the analyses in the manuscript were performed.

Keywords: R, vignette

1. Introduction

The Phylogentic Monte Carlo (pmc) package provides methods for estimating the power phylogenetic methods and providing a robust comparison of common phylogenetic models. This package accompanies Boettiger *et. al.* 2011, *Is your phylogeny informative? Measuring the power of comparative methods.* The following vignette is provided as the supplement to the paper.

This document uses Sweave to automatically regenerate the figures presented in the manuscript from the data provided. The different demos create all the figures from the four different datasets analyzed in the paper (*Geospiza*, simulated large tree, *Anoles*, collection of simulated trees of varying sizes and shapes). Some demos are run on a reduced number of replicates relative to the paper, but all demos may take some time to run on a personal computer.

2. Package availability

The development version of the package is available on Github: https://github.com/cboettig/pmc, under Downloads. The current source-code, issues tracking features, and version history are also available there. A stable version will soon be submitted to the CRAN network.

3. Finches Examples

The first set of examples uses the finches data and geiger functions˜[2] to look at uncertainty in parameter estimates using the pmc method. We start off by loading the required libraries

```
library(pmc)
library(geiger)  # for the data
library(ggplot2)  #graphics
data(geospiza)
attach(geospiza)
```

These next commands are optional, allowing the package to use the specified number of processors, if available.

*Corresponding author.
Email address: cboettig@ucdavis.edu (Carl Boettiger)

```
require(snowfall)  # load the parallelization package
sfInit(parallel = T, cpu = 4)  # specify how many cpus we have

## R Version:  R version 2.14.1 (2011-12-22)
##

sfLibrary(pmc)

## Library pmc loaded.

sfLibrary(geiger)  # export any libraries we've loaded

## Library geiger loaded.
```

Now actually running pmc takes just one line:

```
bm_v_lambda <- pmc(geospiza.tree, geospiza.data["wingL"],
    "BM", "lambda", nboot = 100)
```

Currently the output will only run for a single trait at a time, for efficient memory usage. Here we specify the wing length trait.

We can analyze the parameter distributions as presented in the manuscript. For instance, we can look at a histogram of values of lambda obtained from the different simulations. Because the pmc approach runs four different fits:

- "AA" fitting model A on data obtained by simulations from model A,
- "BA" fitting model B on the data simulated from model A
- "AB" fitting model A on simulations from B
- "BB" fitting B on simulations from B

there are actually 4 different parameter distributions we can use. The comparisons "AA" and "BB" are the typical way one would bootstrap the model fits. All of these combinations are returned in the data set `par_dists` which is one of the items in the list returned by pmc (which we have named `bm_v_lambda` above). Subsetting is a good way to get the parameter of interest, lambda, for the comparison of interest, BB. (Note that for comparisons AA and AB, which fit model Brownian motion, there is of course no parameter lambda).

```
lambdas <- subset(bm_v_lambda[["par_dists"]],
    comparison == "BB" & parameter == "lambda")
```

The returned list from pmc also stores the two models it fit to the original data, inder the names A and B. We can use this to extract the value of lambda estimated on model B from the raw data:

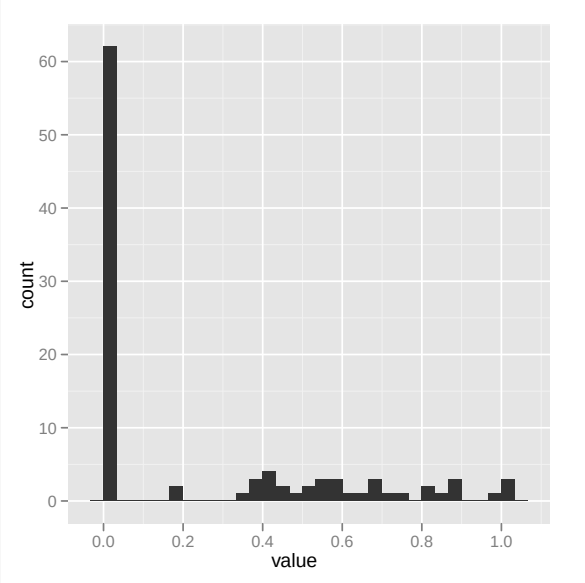
```
est <- bm_v_lambda[["B"]][["wingL"]][["lambda"]]
```

Note that the ability to estimate lambda is very poor, with most simulations returning an estimate of almost zero despite the true value used in the simulations being . Estimating the sigma parameter is somewhat more reliable, even on this small tree:

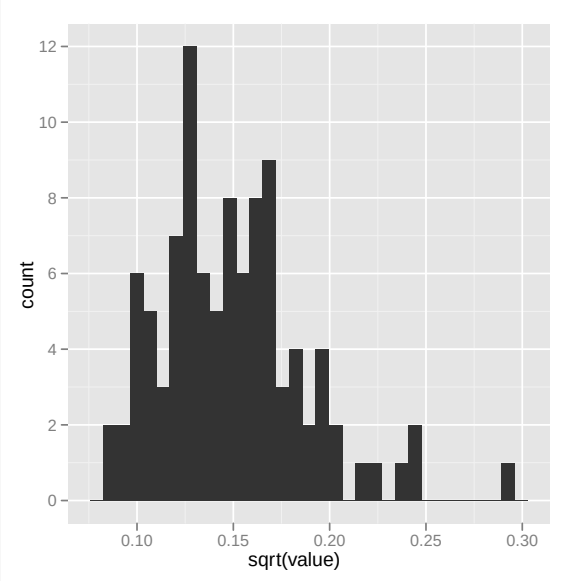
We can also query the confidence intervals directly from the estimates returned by the pmc function. Using the lambda value we already extracted, we can get confidence intervals,

```
cast(lambdas, comparison ~ parameter, function(x) quantile(x,
    c(0.05, 0.95)), value = c("lower", "upper"))

##   comparison lambda_X5. lambda_X95.
## 1         BB      1e-07      0.8914
```

```
p1 <- ggplot(lambdas) + geom_histogram(aes(value)) +
    geom_vline(xintercept = est)
print(p1)
```




```
betas <- subset(bm_v_lambda[["par_dists"]], comparison ==
    "BB" & parameter == "beta")
p2 <- ggplot(betas) + geom_histogram(aes(sqrt(value)))  # beta == sigma^2
print(p2)
```



Or get the intervals for all parameters at once. Note that the "comparison" indicates the model being fit with the first letter, and the model used for the simulations under the second letter.

```
cast(bm_v_lambda$par_dist, comparison ~ parameter,
    function(x) quantile(x, c(0.05, 0.95)), value = c("lower",
        "upper"))

##   comparison beta_X5. beta_X95. k_X5. k_X95. lambda_X5. lambda_X95.
## 1         AA 0.050629   0.81811     2      2         NA          NA
## 2         AB 0.030543   0.29036     2      2         NA          NA
## 3         BA 0.013356   0.09850     3      3      1e-07      0.9733
## 4         BB 0.009438   0.04731     3      3      1e-07      0.8914
##   lnl_X5. lnl_X95.  lr_X5. lr_X95. root_X5. root_X95.
## 1 -7.6841    10.40 -7.6841   10.40    4.107     4.395
## 2 -0.9532    13.68 -0.9532   13.68    4.166     4.355
## 3  1.7368    13.74  1.7368   13.74    4.073     4.423
## 4  6.6146    15.85  6.6146   15.85    4.160     4.367
```

This way we can see the interval for lambda estimated when simulating under the brownian motion model (equiv to lambda=1) under the row BA, as well as the confidence interval on the lambda estimate when simulating with the correct model, lambda=0.6. These cross-comparisons can also be useful. We could always use variations of the subset command above to include only certain parameters or comparisons. We can also summarize all this information at once in a traditional box-and-whiskers plot, Fig˜3. We can repeat the analysis with a larger tree, showing the uncertainty in the estimate of lambda decreases, if only slowly, Fig˜3.

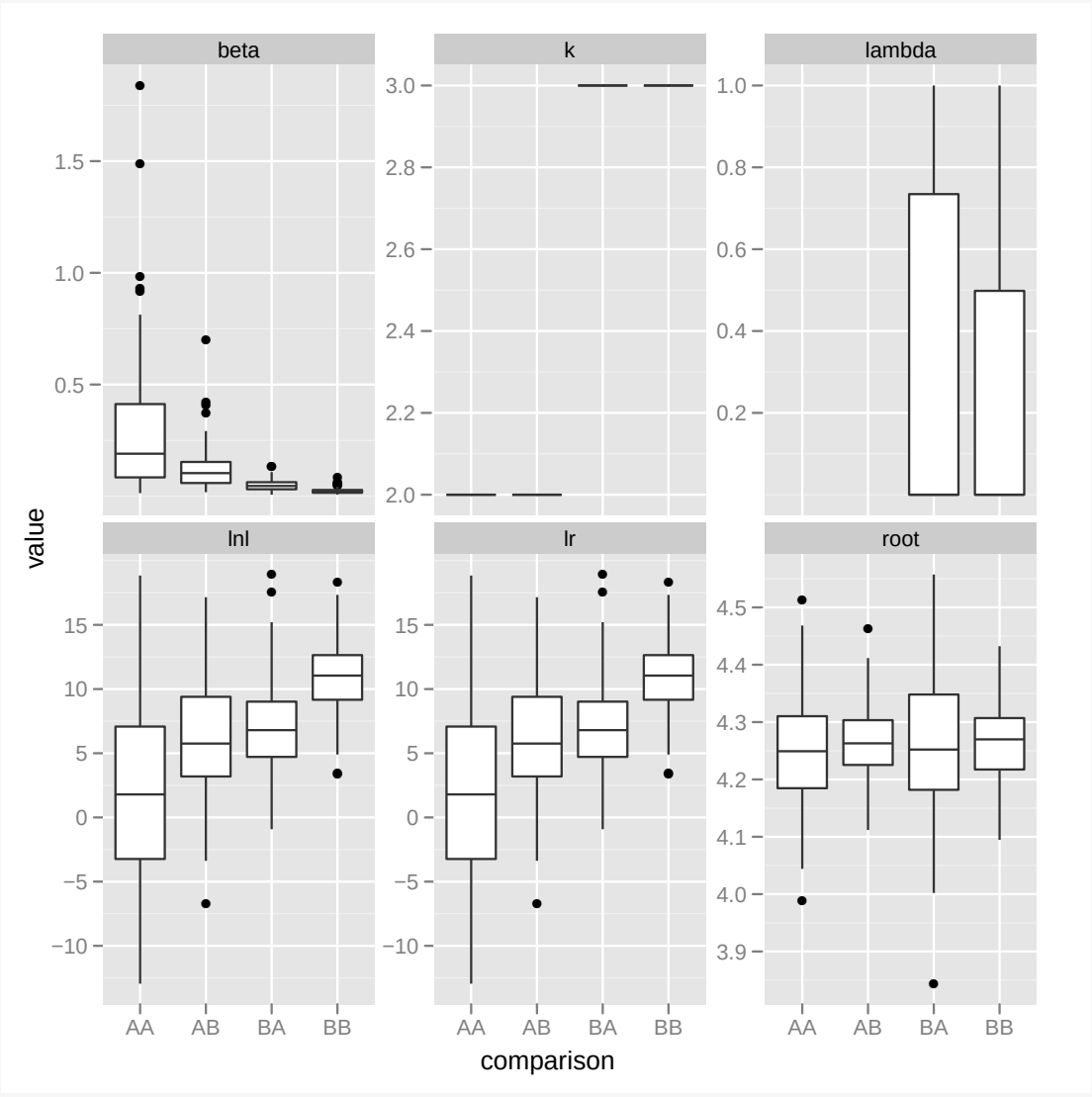
```
require(TreeSim)
simtree <- sim.bd.taxa(n = 281, numbsim = 1, lambda = 1,
    mu = 0, frac = 1, complete = FALSE, stochsampling = FALSE)[[1]][[1]]
simdat <- rTraitCont(lambdaTree(simtree, 0.6),
    sigma = 2)
bm_v_lambda <- pmc(simtree, simdat, "BM", "lambda",
    nboot = 20)
```

4. Anoles example

Next we consider the examples re-analyzing the Anoles data from˜[1], using methods from the ouch package.

```
data(anoles)
ou3v4 <- pmc(tree, log(anoles["size"]), modelA = "hansen",
    modelB = "hansen", optionsA = list(regimes = anoles["OU.LP"],
        sqrt.alpha = 1, sigma = 1), optionsB = list(regimes = anoles["OU.4"],
        sqrt.alpha = 1, sigma = 1), nboot = 100)

ou3v15 <- pmc(tree, log(anoles["size"]), "hansen",
    "hansen", list(regimes = anoles["OU.LP"], sqrt.alpha = 1,
        sigma = 1), list(regimes = anoles["OU.15"], sqrt.alpha = 1,
        sigma = 1), nboot = 100)

ou1v3 <- pmc(tree, log(anoles["size"]), "hansen",
    "hansen", list(regimes = anoles["OU.1"], sqrt.alpha = 1,
        sigma = 1), list(regimes = anoles["OU.LP"], sqrt.alpha = 1,
        sigma = 1), nboot = 100)
```

```
p3 <- plot_pars(bm_v_lambda$par_dists)
print(p3)
```




```
lambdas <- subset(bm_v_lambda[["par_dists"]],
    comparison == "BB" & parameter == "lambda")
p4 <- ggplot(lambdas) + geom_histogram(aes(value)) +
    geom_vline(xintercept = bm_v_lambda[["B"]][["wingL"]][["lambda"]])
print(p4)
```

```
ou0v1 <- pmc(tree, log(anoles["size"]), "brown",
    "hansen", list(), list(regimes = anoles["OU.1"], sqrt.alpha = 1,
        sigma = 1), nboot = 100)
```

Fits from the ouch methods include a phylogeny in the ouch format. As many researchers would rather manipulate and plot the resulting phylogeny in the "phylo" format from the ape package, pmc provides a simple utility to convert the tree while keeping track of the regimes in ouch. [1]

5. Early Burst models

We can also mix and match methods, comparing a fit from geiger with a fit from ouch. We do so in this example, which shows that it may be very difficult to identify if a given dataset comes from an early burst or a OU model, Fig˜1.

```
require(pmc)
require(TreeSim) # to simulate a sample phyologeny
simtree <- sim.bd.taxa(n=60, numbsim=1, lambda=1, mu=0, frac=1,
                       complete=FALSE, stochsampling=FALSE)[[1]][[1]]
dat <- rTraitCont(exponentialchangeTree(simtree, a=-0.5), sigma=5)
regimes <- format_data(simtree,dat)$noregimes
  eb_v_ou <- pmc(simtree, dat, modelA = "EB", modelB = "hansen",
    optionsB = list(sqrt.alpha = 1, sigma = 1, regimes = regimes),
    nboot = 50)
```

6. Utilities

To implement the pmc package over a variety of methods, I have written an abstraction layer for phylogenetic comparative methods. A few of those methods may be useful to others, so I have made them accessible from the package and introduce them here. Using the simulated tree and data from the early burst example above, we can swap between formats of both the tree and the data at whim, something that might be useful for anyone using both ouch and the other methods.

```
ouch <- format_data(simtree, dat)
ape <- format_data(ouch$tree, ouch$dat)
```

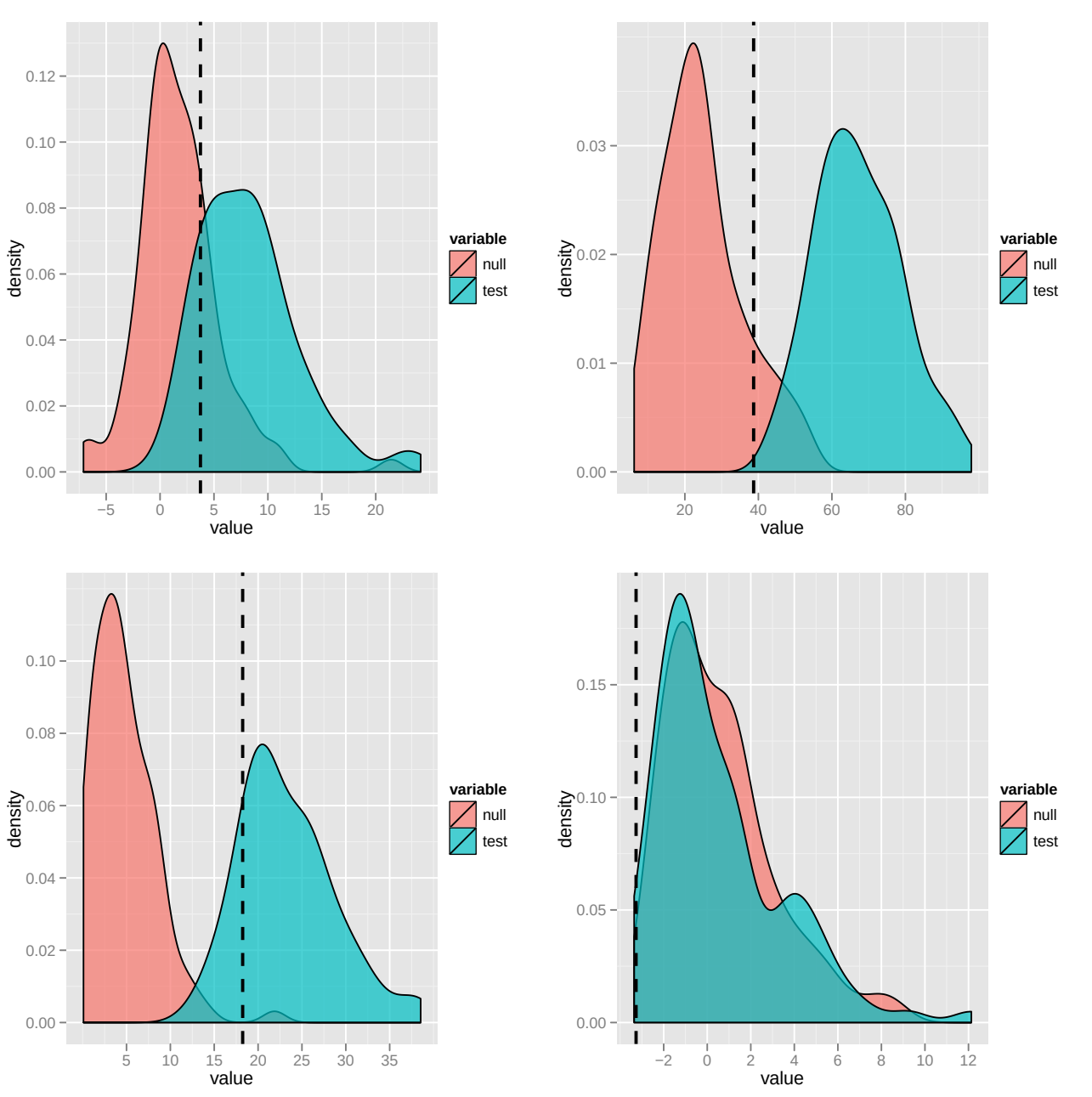
This format swapping is handled automatically in pmc's generic function for fitting phylogenetic models. This function acts as a wrapper around existing methods, allowing the user to call the ouch methods while passing in data in either ouch or geiger format.

```
A <- pmc_fit(ape$tree, ape$data, model = "BM")

## Fitting  BM model:

B <- pmc_fit(simtree, dat, model = "hansen", options = list(sqrt.alpha = 1,
    sigma = 1, regimes = regimes))
```

This function also returns an object of a class that matches the fitting method used. This means that we can use commands such as:

[1] Some caution should always be applied in using "phylo" format trees. Unlike the ouch format, a given tree does not have a unique specification, and consequently some functions that use "phylo" trees make assumptions about the ordering of edges that may not be true, even for a technically valid, plottable "phylo" specification of the phylogeny.

```
a <- plot(ou3v4, A = "OU.3", B = "OU.4")
b <- plot(ou3v15, A = "OU.3", B = "OU.15")
c <- plot(ou1v3, A = "OU.1", B = "OU.3")
d <- plot(ou0v1, A = "BM", B = "OU.1")

grid.newpage()
pushViewport(viewport(layout = grid.layout(2,
    2)))
vplayout <- function(x, y) viewport(layout.pos.row = x,
    layout.pos.col = y)
print(a, vp = vplayout(1, 1))
print(b, vp = vplayout(1, 2))
print(c, vp = vplayout(2, 1))
print(d, vp = vplayout(2, 2))
```



```
plot(eb_v_ou)
```

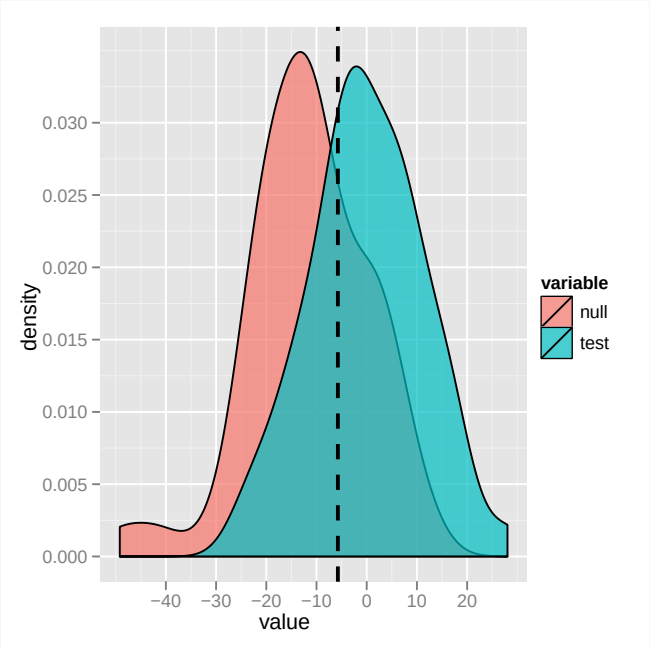


Figure 1: Early burst models can be very almost impossible to distinguish from OU models when each has been parameterized from the same data.

```
sim <- simulate(A)

## Error: no applicable method for 'simulate' applied to an object of class "fitContinuous"

newA <- update(A, simulate(A))

## Error: no applicable method for 'simulate' applied to an object of class "fitContinuous"
```

which automatically recognize the kind of model fit and use that method to perform the simulations and updates. This object-oriented design already exists in ouch, so this simply extends the practice to other methods. The fitted object stores the traits and phylogeny used to make the fit, as well as the model and fitting options, which can also help the user avoid reproducibility errors of mismatching data, fit options and results.

6.1. Power Curves

A seperate function, `treepower`, creates the power curves in Figure 6 of Boettiger et al, Evolution. This code illustrates how such an analysis is run with the pmc package.

```
require(pmc)
require(TreeSim)
require(ouch)
nboot <- 200
cpu <- 4
alpha <- c(0.01, 0.1, 1, 1.5, 2, 5, 10, 20)
n <- c(5, 10, 20, 50, 100)
lambda <- c(0.25, 0.5, 0.75, 1)
# size simulations
size <- lapply(1:length(n), function(i) {
```

```
    simtree <- sim.bd.taxa(n = n[i], numbsim = 1, lambda = 1,
        mu = 0, frac = 1, complete = FALSE, stochsampling = FALSE)[[1]][[1]]
    treepower(ape2ouch(simtree), nboot = nboot, cpu = cpu,
        alpha = alpha)
})

## Library pmc loaded.

## Library pmc loaded.

## Library pmc loaded.

## Library pmc loaded.

## Library pmc loaded.

## Shape simulations
N <- 50  ## number of taxa fixed to 50
shape <- lapply(1:length(lambda), function(i) {
    simtree <- sim.bd.taxa(n = N, numbsim = 1, lambda = 1,
        mu = 0, frac = 1, complete = FALSE, stochsampling = FALSE)[[1]][[1]]
    simtree <- lambdaTree(simtree, lambda[i])
    treepower(ape2ouch(simtree), nboot = nboot, cpu = cpu,
        alpha = alpha)
})

## Library pmc loaded.

## Library pmc loaded.

## Library pmc loaded.

## Library pmc loaded.
```


[1] Butler, M.˜A., King, A.˜A., Dec. 2004. Phylogenetic Comparative Analysis: A Modeling Approach for Adaptive Evolution. The American Naturalist 164˜(6), 683–695.
URL http://www.jstor.org/stable/10.1086/426002

[2] Harmon, L.˜J., Weir, J.˜T., Brock, C.˜D., Glor, R.˜E., Challenger, W., 2008. Geiger: investigating evolutionary radiations. Bioinformatics 24˜(1), 129–131.

```
plot_size <- function() {
    k <- length(n)
    plot(1, 1, type = "n", xlim = c(min(alpha), max(alpha)),
        ylim = c(0, 1), main = "Power by tree size", log = "x",
        xlab = "alpha", ylab = "power")
    for (i in 1:k) {
        ## skip the last 2, which haven't converged
        points(alpha, size[[i]]$power, pch = 16, col = i)
        lines(alpha, size[[i]]$power, col = i)
    }
    legend("topleft", paste(n, "taxa"), col = 1:k, pch = 16)

}
plot_size()
```

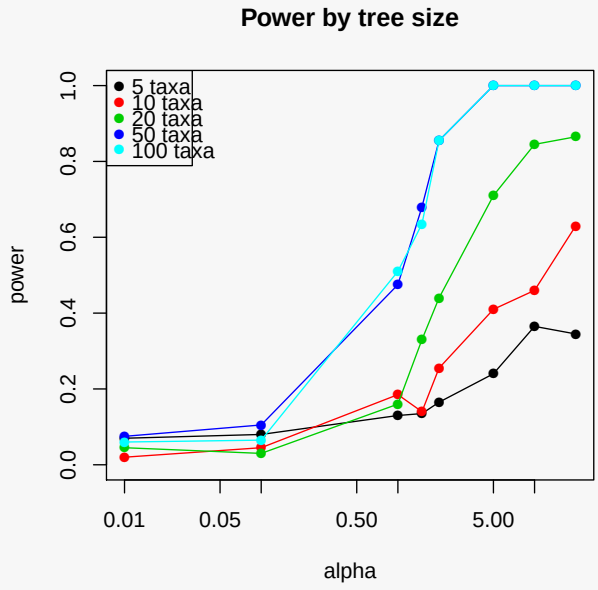


Figure 2:

```
plot_shape <- function() {
    plot(1, 1, type = "n", xlim = c(min(alpha), max(alpha)),
        ylim = c(0, 1), main = "Power by tree topology",
        log = "x", xlab = "alpha", ylab = "power")
    k <- length(lambda)
    for (i in 1:k) {
        points(alpha, shape[[i]]$power, pch = 16, col = i)
        lines(alpha, shape[[i]]$power, col = i)
    }
    legend("topleft", paste(lambda, "lambda"), col = 1:k,
        pch = 16)
}
plot_shape()
```