Analysing diversification with diversitree

Rich FitzJohn

December 1, 2009

1 Introduction

This demo is in a very early stage of development. It does not aim to be a complete reference to the package, but should give some useful hints. The code all assumes that the diversitree package is loaded.

```
> library(diversitree)
```

2 Simulating phylogenies

diversitree includes code for simulating phylogenies, under both a character-dependent BiSSE-type model and a character-independent model (as in Nee et al., 1994). The tree simulation code is likely to change a reasonable amount soon. It will be useful to have a simulated tree to demonstrate the code; the following generates a birth-death tree following the BiSSE model with 203 species, using $\lambda_0 = 0.1$, $\lambda_1 = 0.2$, $\mu_0 = \mu_1 = 0.03$, and $q_{01} = q_{10} = 0.01$, starting in state 0 and running for 60 time units.

```
> pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
> set.seed(2)
> phy <- tree.bisse(pars, max.t = 60, x0 = 0)
```

3 Running BiSSE

The way diversitree runs BiSSE, you first construct a likelihood function with `make.bisse`, then use this in a maximum likelihood or MCMC approach to do your inference. The `make.bisse` takes as its first two arguments a tree and set of character states:

```
> lik <- make.bisse(phy, phy$tip.state)
```

The object "`lik`" is the likelihood function; given a vector of parameters (in the order λ_0, λ_1, μ_0, μ_1, q_{01}, q_{10}) it computes the likelihood of the tree and character data:

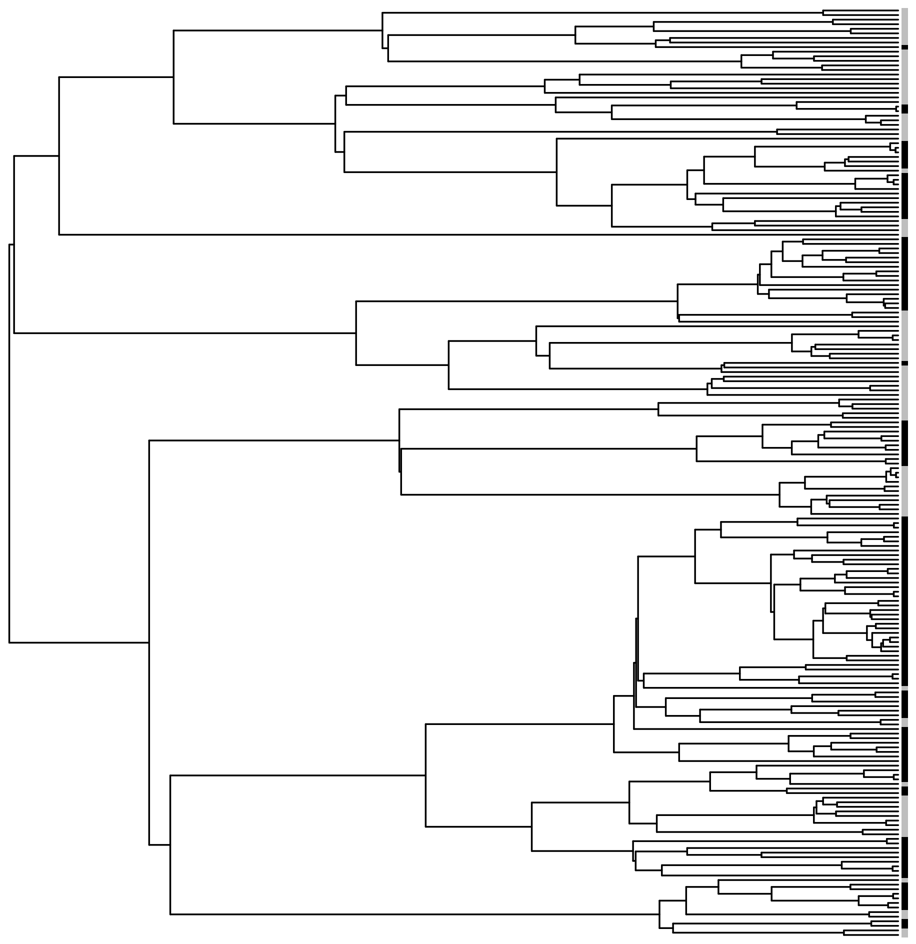


Figure 1: A random BiSSE tree. Gray indicates state `0` (low speciation) and black indicates state 1 (high speciation).

```
> lik(pars)

[1] -660.0173
```

To do an maximum likelihood analysis, we need to find the set of parameters that maximises the likelihood. The `find.mle` function provides an interface to R's built-in `optim` that simplifies this process. To perform an optimisation, we need a starting point. The `starting.point.bisse` function does a very basic heuristic search for a sensible starting point, based on the character-independent birth-death fit.

```
> p <- starting.point.bisse(phy)
> p

   lambda0    lambda1        mu0        mu1        q01        q10 
0.16721949 0.16721949 0.08017315 0.08017315 0.01740927 0.01740927 
```

Run the optimisation (this may take up to a minute, depending on the speed of your computer).

```
> fit <- find.mle(lik, p)
> fit

$par
    lambda0     lambda1         mu0         mu1         q01         q10 
0.099429683 0.196200094 0.024026313 0.032468600 0.010031906 0.009869937 

$lnLik
[1] -659.9226

$counts
function gradient 
      41       41 

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"

attr(,"class")
[1] "mle.bisse"
```

The object returned by `find.mle` will change in the future, but the key bits are the element `par` with the ML parameters and `lnLik` with the best log-likelihood value. It would be preferable to start this search from multiple starting points, and I will add something to automate that to some degree soon.

3.1 Constrained models

To constrain a model, use the function `constrain`. This can be used to set parameters to a constant, to be the same as other parameters, or even to arbitrary expressions of free parameters. This is done through a formula interface that is still undergoing some change. As an example, consider the model where $\lambda_1 = \lambda_0$:

```
> lik.equal.l <- constrain(lik, lambda1 ~ lambda0)
```

The second argument to `constrain` describes the constraint. Additional arguments could impose different constraints; for example, setting μ_0 to zero.

```
> lik.equal.l.2 <- constrain(lik, lambda1 ~ lambda0, mu0 ~ 0)
```

This can be optimised as before, starting this time at the full model's MLE (`find.mle` here automatically does the filtering of the paramters, but you can alternatively specify a correct-length parameter vector).

```
> fit.equal.l <- find.mle(lik.equal.l, fit$par, method = "subplex")
> fit.equal.l[1:2]

$par
     lambda0          mu0          mu1          q01          q10
1.625970e-01 1.075971e-01 5.852687e-06 6.083873e-03 1.310082e-02

$lnLik
[1] -663.4045
```

(subplex does better on this search, so I used it here.) This model has tweaked the extinction rates to get a suitable diversification rate, and settled on a λ that is intermediate between the true rates. The likelihood has dropped 3.5 units; a likelihood ratio test gives this a p value of 0.008:

```
> anova(fit, equal.l = fit.equal.l)

        Df   lnLik    AIC  ChiSq Pr(>|Chi|)
full     6 -659.92 1331.8
equal.l  5 -663.40 1336.8 6.9637   0.008318
```

In a second example, consider preventing $0 \rightarrow 1$ transitions:

```
> lik.no.01 <- constrain(lik, q01 ~ 0)
> fit.no.01 <- find.mle(lik.no.01, fit$par)
> fit.no.01[1:2]
```

```
$par
   lambda0    lambda1        mu0        mu1        q10
0.10568906 0.23544862 0.05835147 0.14417802 0.02559472

$lnLik
[1] -673.0052
```

This model is strongly rejected ($p < 0.001$)

```
> anova(fit, no.01 = fit.no.01)

      Df   lnLik    AIC  ChiSq Pr(>|Chi|)
full   6 -659.92 1331.8
no.01  5 -673.01 1356.0 26.165  3.134e-07
```

3.2 Markov Chain Monte Carlo (MCMC)

The likelihood function can also be used to perform Markov Chain Monte Carlo (MCMC). I will use an exponential prior with a rate of 5.7 (having a mean twice the character independent diversification rate). Running this will take some time (allow up to 1 s per step on this tree). The output is a large data frame with the contents of the chain.

```
> r <- 1/(2 * (p[1] - p[3]))
> ans <- mcmc(lik, fit$par, nsteps = 100, w = rep(0.1, 6), lower = rep(0,
+     6), upper = rep(Inf, 6), prior = r, fail.value = -Inf)
```

3.3 Hints on picking starting points for analyses

It is possible that the heuristic starting point chosen by `starting.point.bisse` will not lead the optimiser to find the true optimum. This is particularly likely on small trees where there is little good information about the parameters.

```
> str <- paste("((a:0.45474,((b:0.31527,(((c:0.120609,d:0.120609):0.13884,",
+     "((e:0.160074,(f:0.099202,g:0.099202):0.060872):0.016381,",
+     "h:0.176455):0.082993):0.035362,i:0.29481):0.02046):0.094062,",
+     "j:0.409331):0.045409):0.54526,(((k:0.202246,l:0.202246):",
+     "0.149872,m:0.352119):0.05863,n:0.410749):0.589251);", collapse = "")
> tree <- read.tree(text = str)
> states <- rep(c(1, 0, 1), c(10, 1, 3))
> names(states) <- tree$tip.label
> lik <- make.bisse(tree, states)
> p <- starting.point.bisse(tree)
> fit.1 <- find.mle(lik, p)
> fit.1[1:2]
```

```
$par
  lambda0   lambda1       mu0       mu1       q01       q10
0.0000000 2.6144377 0.0000000 0.0000000 4.8089370 0.4849328

$lnLik
[1] -5.058652
```

The ML point here has a log likelihood of -5.06, but there is a better point out there:

```
> lik(c(12.64, 0, 0, 8.45, 15.76, 0))

[1] -3.845798

> lik(c(12.64, 0, 0, 8.45, 15.76, 0)) - fit.1$lnLik

[1] 1.212854
```

To find this point, we need to use "better" starting conditions. However, there is no surefire way of doing this. It is never knowable that the best point has been found. One option that is popular in other similar problems is to try the ML search from several different points. Picking points so that $\lambda_i - \mu_i$ stays equal to the estimated diversification rate (which is `p[1] - p[3]`, above), but that different combinations of high and low rates are tried:

```
> r <- p[1] - p[3]
> mu <- c(0, r * 4)
> q <- c(r/5, r)
> pars <- expand.grid(mu0 = mu, mu1 = mu, q01 = q, q10 = q)
> pars <- cbind(lambda0 = pars$mu0 + r, lambda1 = pars$mu1 + r,
+     pars)
```

This produces 16 different candidate starting points. We can run the search from each of these and take the best case:

```
> fits <- apply(pars, 1, function(p) find.mle(lik, p))
> fits.ll <- sapply(fits, function(x) x$lnLik)
> fit.2 <- fits[[which.max(fits.ll)]]
> fit.2[1:2]

$par
  lambda0   lambda1       mu0       mu1       q01       q10
12.640520  0.000000  0.000000  8.446932 15.755291  0.000000

$lnLik
[1] -3.845795
```

However, this is not guaranteed to work in general. Other options for chosing starting parameters might be better; for example Latin hypercube sampling (see package `tgp`) will probably produce points that are better dispersed in the high-dimensional parameter space.

4 Skeleton trees

To sample from the simulated phylogeny above to make a skeleton tree, we can do this:

```
> set.seed(1)
> n.taxa <- length(phy$tip.label)
> keep <- sort(sample(n.taxa, 50))
> phy.s <- drop.tip(phy, setdiff(seq_len(n.taxa), keep))
```

A need likelihood function is needed for this tree. We also need to specify what the sampling fraction is (close to 20% for both).

```
> n.real <- table(phy$tip.state)
> n.samp <- table(phy.s$tip.state[phy.s$tip.label])
> sampling.f <- n.samp/n.real
> sampling.f

        0         1
0.2048193 0.2750000
```

Pass this in to `make.bisse` and construct a new likelihood function that accounts for the sampling:

```
> lik.s <- make.bisse(phy.s, phy.s$tip.state, sampling.f = sampling.f)
```

This can then be optimised, as before:

```
> fit.s <- find.mle(lik.s, p)
> fit.s[1:2]

$par
    lambda0     lambda1         mu0         mu1         q01         q10
0.085199300 0.301619056 0.000000000 0.128489700 0.011591898 0.006958608

$lnLik
[1] -186.4951
```

The fits compare reasonably well from the full tree and sampled tree

```
> round(rbind(full = fit$par, sampled = fit.s$par), 3)

        lambda0 lambda1   mu0   mu1   q01   q10
full      0.099   0.196 0.024 0.032 0.010 0.010
sampled   0.085   0.302 0.000 0.128 0.012 0.007
```

5 Terminally unresolved trees

For this section, I will show how the shorebird example in the manuscript can be performed. The file `Thomas-tree.nex` contains the shorebird supertree assembled by Thomas et al. (2004) in nexus format and the file `Lislevand-states.csv` contains a measure of the level of sexual dimorphism, computed from the data in Lislevand et al. (2007). The tree contains many polytomies, so I am converting it into a terminally unresolved tree, with the `polytomies.to.clades` function.

```
> tree <- read.nexus("data/Thomas-tree.nex")
> tree2 <- polytomies.to.clades(tree)
```

Terminally unresolved trees can be assembled manually from a phylogeny that contains only the exemplar tips using the `make.clade.tree` function.

```
> states <- read.csv("data/Lislevand-states.csv", as.is = TRUE)
> states <- structure(states$dimorph, names = states$species)
```

Every species needs tip data

```
> states <- states[tree$tip.label]
> names(states) <- tree$tip.label
```

This is all the information `make.bisse` needs. Construct a likelihood function by categorising states with 15% dimorphism being the cut-off:

```
> lik.sb <- make.bisse(tree2, abs(states) > 0.15)
```

This starting point comes from ape's function `bd.ext`, but this cannot be automated, as it does not return an object.

```
> p <- c(0.147, 0.147, 0, 0, 0.02, 0.02)
> names(p) <- c("lambda0", "lambda1", "mu0", "mu1", "q01", "q10")
```

Find the ML point (this will take $1-2$ minutes, and will produce warnings that can be ignored).

```
> fit.sb <- find.mle(lik.sb, p)
> fit.sb[1:2]

$par
   lambda0    lambda1        mu0        mu1        q01        q10
0.05304103 0.20340438 0.00000000 0.00000000 0.04235569 0.28197862

$lnLik
[1] -633.7833
```

It looks from this that the speciation rate and the character transition rate in state 1 are higher than the corresponding rates in state 0 ($\lambda_1 > \lambda_0$, $q_{10} > q_{01}$). The significance of this can be tested with a likelihood ratio test:

```
> lik.sb.l <- constrain(lik.sb, lambda1 ~ lambda0)
> lik.sb.q <- constrain(lik.sb, q10 ~ q01)
> fit.sb.l <- find.mle(lik.sb.l, fit.sb$par)
> fit.sb.q <- find.mle(lik.sb.q, fit.sb$par)
```

These constrained models are significantly worse fits than the full model (λ: $p = 0.0005$, q: $p = 0.01$).

```
> anova(fit.sb, equal.l = fit.sb.l, equal.q = fit.sb.q)

        Df   lnLik    AIC   ChiSq Pr(>|Chi|)
full     6 -633.78 1279.6
equal.l  5 -639.88 1289.8 12.1913  0.0004801
equal.q  5 -636.88 1283.8  6.2008  0.0127691
```

This analysis can also be done with MCMC (not shown, and will take some time)

```
> ans <- mcmc(lik.sb, fit.sb$par, nsteps = 10000, w = rep(0.1,
+     6), lower = rep(0, 6), upper = rep(Inf, 6), prior = 1/p[1] *
+     2, fail.value = -Inf)
```

6 Constant-rate birth-death models

The ape package already has some support for constant-rate birth-death models, but diversitree duplicates for this for completeness. The major differences are (1) the function is not constrained to positive diversification rates (μ can exceed λ), (2) [eventual] support for both random taxon sampling and unresolved terminal clades (but see ape's `bd.ext`), and (3) run both MCMC and MLE fits to birth death trees.

The analysis proceeds in much the same way as for BiSSE models; first construct a likelihood function with `make.bd`, possibly constrain it, and run ML or MCMC analyses. To illustrate, here is a 100 species tree

```
> set.seed(2)
> phy <- trees(c(0.1, 0.03), "bd", max.taxa = 100)[[1]]
```

To do a ML model fit:

```
> lik <- make.bd(phy)
> fit <- find.mle(lik)
> fit[1:2]
```

```
$par
    lambda         mu
0.10105046 0.06088239

$lnLik
[1] -3.162741
```

Does a Yule model (no extinction) fit better?

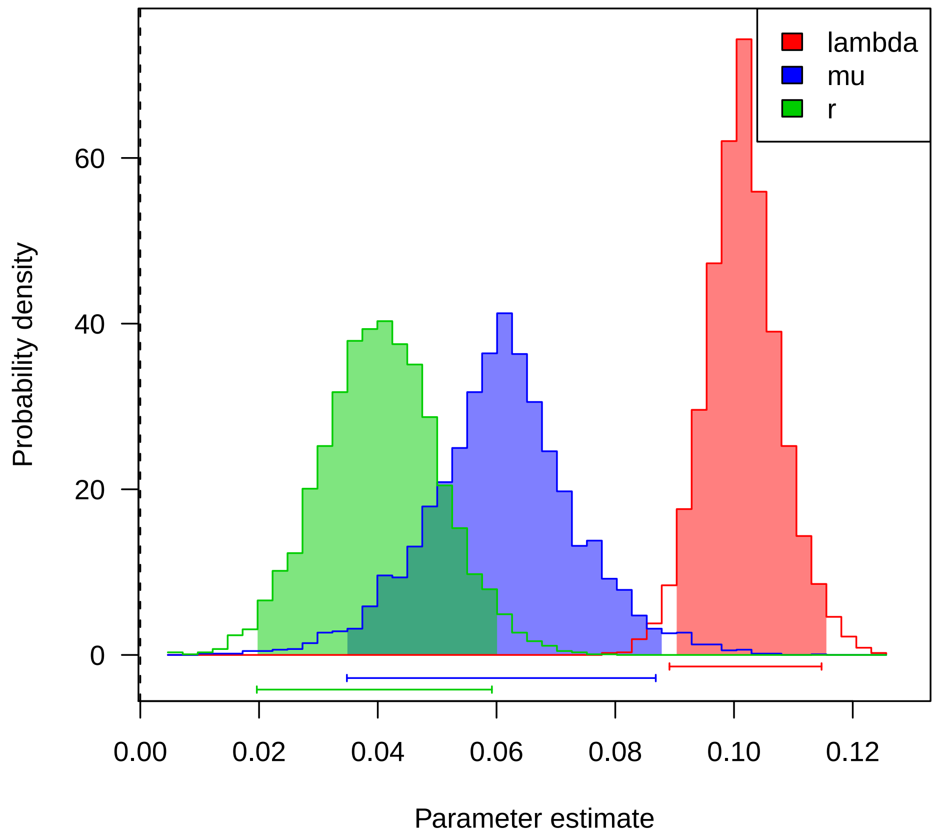
```
> lik.yule <- constrain(lik, mu ~ 0)
> fit.yule <- find.mle(lik.yule, fit$par, method = "L-BFGS-B")
```

The model with the nonzero extinction estmate is preferred:

```
> anova(fit, fit.yule)

        Df   lnLik    AIC  ChiSq Pr(>|Chi|)
full     2 -3.1627 10.325
model 1  1 -6.3670 14.734 6.4085    0.01136
```

MCMC methods also work:

```
> samples <- mcmc(lik, fit$par, nsteps = 5000, lower = c(-Inf,
+     -Inf), upper = c(Inf, Inf), w = c(0.1, 0.1), fail.value = -Inf,
+     print.every = 500)

500: {0.0978, 0.0558} -> -3.18117
1000: {0.1014, 0.0764} -> -4.19869
1500: {0.0982, 0.0606} -> -3.22054
2000: {0.0979, 0.0500} -> -3.38768
2500: {0.1046, 0.0519} -> -3.93809
3000: {0.0934, 0.0616} -> -3.72046
3500: {0.1107, 0.0848} -> -3.92409
4000: {0.0983, 0.0566} -> -3.17590
4500: {0.1067, 0.0654} -> -3.22772
5000: {0.1060, 0.0695} -> -3.21930

> samples$r <- with(samples, lambda - mu)
> samples$a <- with(samples, mu/lambda)
> col <- c("red", "blue", "green3")
> profiles.plot(samples[c("lambda", "mu", "r")], col.line = col,
+     las = 1, ylab = "Probability density", xlab = "Parameter estimate")
> legend("topright", c("lambda", "mu", "r"), fill = col)
> abline(v = 0, lty = 2)
```



6.1 Equivalency of BiSSE and constant-rate models

The estimates from the constant rate birth-death model should match up with those from BiSSE (though the BiSSE calculation is substantially slower)

```
> pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
> set.seed(2)
> phy <- tree.bisse(pars, max.t = 60, x0 = 0)
> lik.cr1 <- make.bd(phy)
> fit.cr1 <- find.mle(lik.cr1)
> lik <- make.bisse(phy, phy$tip.state)
> lik.cr2 <- constrain(lik, lambda1 ~ lambda0, mu1 ~ mu0, q10 ~
+     q01)
> fit.cr2 <- find.mle(lik.cr2, starting.point.bisse(phy))
> fit.cr3 <- suppressWarnings(birthdeath(phy))
> r <- fit.cr3$para[2]
> e <- fit.cr3$para[1]
> fit.cr3$par <- c(lambda = r/(1 - e), mu = r * e/(1 - e))
> rbind(constant = fit.cr1$par, bisse = fit.cr2$par[1:2], ape = fit.cr3$par)

            lambda         mu
constant 0.1672197 0.08017355
bisse    0.1672621 0.08023830
ape      0.1672195 0.08017315
```

Note that discrepencies can occur because of approximations in computing the likelihoods in the BiSSE method.

References

Lislevand T., Figuerola J., and Székely T. 2007. Avian body sizes in relation to fecundity, mating system, display behavior, and resource sharing. Ecology 88:1605.

Nee S., May R.M., and Harvey P.H. 1994. The reconstructed evolutionary process. Philos. Trans. R. Soc. Lond. B Biol. Sci. 344:305–311.

Thomas G.H., Wills M.A., and Székely T. 2004. A supertree approach to shorebird phylogeny. BMC Evolutionary Biology 4:28.