Package ‘diversitree’

March 25, 2012

Title diversitree: comparative phylogenetic analyses of diversification

Version 0.9-2

Date 2012-03-15

Author Richard G. FitzJohn <fitzjohn@zoology.ubc.ca>, with GeoSSE and ClaSSE by Emma E. Goldberg <eeg@uic.edu>, and BiSSE-ness by Karen Magnuson-Ford.

Maintainer Richard G. FitzJohn <fitzjohn@zoology.ubc.ca>

Depends R (>= 2.10), deSolve (>= 1.7), ape, subplex

Suggests numDeriv, tgp, multicore, minqa

SystemRequirements fftw3 (>= 3.1.2)

Description Contains a number of comparative phylogenetic methods,mostly focussing on analysing diversification and character evolution. Contains implementations of BiSSE (Binary State Speciation and Extinction; Maddison et al. 2007: Syst. Biol. 56: 701) and its unresolved tree extensions (FitzJohn et al. 2009: Syst. Biol. 58: 595), MuSSE (Multiple State Speciation and Extinction), QuaSSE (Quantitative State Speciation and Extinction; FitzJohn 2010: Syst. Biol. 59: 619), GeoSSE (Geographic State Speciation and Extinction; Goldberg et al. 2011: Syst. Biol. 60: 451), and BiSSE-ness (Magnuson-Ford and Otto 2012: Am. Nat.) Other included methods include Markov models of discrete and continuous trait evolution and constant rate speciation and extinction.

License GPL (>=2)

URL http://www.zoology.ubc.ca/prog/diversitree

Archs x86_64

R topics documented:

argnames 2
asr 3
asr-bisse 4
asr-mkn 6
check 8

combine . . . 9
constrain . . . 9
find.mle . . . 12
geosse.phy . . . 15
history.from.sim . . . 16
make.bd . . . 16
make.bd.split . . . 19
make.bd.t . . . 20
make.bisse . . . 22
make.bisse.split . . . 27
make.bisse.td . . . 29
make.bisseness . . . 32
make.bm . . . 36
make.clade.tree . . . 38
make.classe . . . 39
make.geosse . . . 41
make.geosse.split . . . 43
make.mkn . . . 44
make.mkn.deriv . . . 47
make.musse . . . 47
make.musse.multitrait . . . 50
make.musse.split . . . 54
make.musse.td . . . 56
make.prior . . . 58
make.quasse . . . 59
make.quasse.split . . . 61
mcmc . . . 62
plot.history . . . 65
profiles.plot . . . 67
quasse-common . . . 68
set.defaults . . . 69
sim.character . . . 70
simulate . . . 71
time . . . 73
trait.plot . . . 74

Index **77**

`argnames` *Argument Names for Vector-Argument Functions*

Description

Functions to get and set "argument names" for functions that take vectorised arguments. For example, the likelihood function returned by `make.bisse` takes a vector of six these functions can be used to get the canonical names for these six parameters, and also to set them to something more memorable. These names are used by the `constrain` function to specify submodels.

Usage

```
argnames(x, ...)
argnames(x) <- value
## S3 method for class 'constrained'
argnames(x, ...)
## S3 replacement method for class 'constrained'
argnames(x) <- value
```

Arguments

x	A function taking a vector of parameters as its first argument.
value	Vector of names to set the argument names to.
...	Ignored arguments to future methods.

Details

Methods exist for all models: bisse, geosse, bd, yule, mk2, and mkn. These are particulary useful for mkn as the number of parameters for the Q matrix can be very large.

Author(s)

Richard G. FitzJohn

Examples

```
## Same example likelihood function as for \link{make.bisse}:
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
set.seed(4)
phy <- tree.bisse(pars, max.t=30, x0=0)
f <- make.bisse(phy, phy$tip.state)

argnames(f) # Canonical argument names (set by default)
## Names that might be more informative for a tall/short state
argnames(f) <- c("l.tall", "l.short", "m.tall", "m.short",
                 "q.tall.short", "q.short.tall")
argnames(f)
```

---

asr — *Ancestral State Reconstruction*

Description

Perform ancestral state reconstruction. These functions are all generic and will dispatch on the class of the given likelihood functions. Currently methods exist for all generics for Mk2, and marginal ancestral state reconstructions are supported for BiSSE.

Usage

```
asr.marginal(lik, pars, nodes=NULL, ...)
asr.joint(lik, pars, n=1, ...)
asr.stoch(lik, pars, n=1, ...)

make.asr.marginal(lik, ...)
make.asr.joint(lik, ...)
make.asr.stoch(lik, ...)
```

Arguments

`lik`	A likelihood function.
`pars`	A vector of parameters, suitable for `lik`.
`nodes`	For `asr.marginal` only; an optional vector of nodes to return ancestral states for (using ape's index). By default, all nodes are returned.
`n`	The number of samples to draw from the joint distribution, or number of stochastic reconstructions to make.
`...`	Additional arguments passed through to future methods

Details

These three functions compute marginal, joint, and stochastic ancestral reconstructions. The `make` versions return functions that can efficiently be used many times over.

Value

The return values of the functions are likely to change in the near future. Watch out!

Author(s)

Richard G. FitzJohn

See Also

asr.mkn and asr.bisse for methods specific to particular classes, with examples of use.

`asr-bisse` *Ancestral State Reconstruction Under BiSSE*

Description

Perform ancestral state reconstruction under BiSSE and other constant rate Markov models. Marginal reconstructions are supported (c.f. `asr`). Documentation is still in an early stage, and mostly in terms of examples.

Usage

```
## S3 method for class 'bisse'
make.asr.marginal(lik, ...)
## S3 method for class 'musse'
make.asr.marginal(lik, ...)
```

Arguments

`lik`	A likelihood function, returned by `make.mk2` or `make.mkn`.
`...`	Additional arguments passed through to future methods. Currently unused.

Author(s)

Richard G. FitzJohn

Examples

```
## Start with a simple tree evolved under a BiSSE with all rates
## asymmetric:
pars <- c(.1, .2, .03, .06, .01, .02)
set.seed(3)
phy <- trees(pars, "bisse", max.taxa=50, max.t=Inf, x0=0)[[1]]

## Here is the true history
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy, main="True history")

## BiSSE ancestral state reconstructions under the ML model
lik <- make.bisse(phy, phy$tip.state)
fit <- find.mle(lik, pars, method="subplex")
st <- asr.marginal(lik, coef(fit))
nodelabels(thermo=t(st), piecol=1:2, cex=.5)

## Mk2 ancestral state reconstructions, ignoring the shifts in
## diversification rates:
lik.m <- make.mk2(phy, phy$tip.state)
fit.m <- find.mle(lik.m, pars[5:6], method="subplex")
st.m <- asr.marginal(lik.m, coef(fit.m))
## The Mk2 results have more uncertainty at the root, but both are
## similar.
nodelabels(thermo=t(st.m), piecol=1:2, cex=.5, adj=-.5)

## Not run:
## (This section will take 10 or so minutes to run.)
## Try integrating over parameter uncertainty and comparing the BiSSE
## with Mk2 output:
prior <- make.prior.exponential(2)
samples <- mcmc(lik, coef(fit), 1000, w=1, prior=prior,
               print.every=100)
st.b <- apply(samples[2:7], 1, function(x) asr.marginal(lik, x)[2,])
st.b.avg <- rowMeans(st.b)

samples.m <- mcmc(lik.m, coef(fit.m), 1000, w=1, prior=prior,
                 print.every=100)
st.m <- apply(samples.m[2:3], 1, function(x) asr.marginal(lik.m, x)[2,])
st.m.avg <- rowMeans(st.m)

## These end up being more striking in their similarity than their
## differences, except for the root node, where BiSSE remains more sure
## that is in state 0 (there is about 0.05 red there).
plot(h, phy, main="Marginal ASR, BiSSE (left), Mk2 (right)",
     show.node.state=FALSE)
nodelabels(thermo=1-st.b.avg, piecol=1:2, cex=.5)
```

```
nodelabels(thermo=1-st.m.avg, piecol=1:2, cex=.5, adj=-.5)

## End(Not run)

## Equivalency of Mk2 and BiSSE where diversification is state
## independent.  For any values of lambda/mu (here .1 and .03) where
## these do not vary across character states, these two methods will
## give essentially identical marginal ancestral state reconstructions.
st.id <- asr.marginal(lik, c(.1, .1, .03, .03, coef(fit.m)))
st.id.m <- asr.marginal(lik.m, coef(fit.m))

## Reconstructions are identical to a relative tolerance of 1e-7
## (0.0000001), which is similar to the expected tolerance of the BiSSE
## calculations.
all.equal(st.id, st.id.m, tolerance=1e-7)

## Equivalency of BiSSE and MuSSE reconstructions for two states:
lik.b <- make.bisse(phy, phy$tip.state)
lik.m <- make.musse(phy, phy$tip.state + 1, 2)

st.b <- asr.marginal(lik.b, coef(fit))
st.m <- asr.marginal(lik.m, coef(fit))

all.equal(st.b, st.m)
```

---

asr-mkn *Ancestral State Reconstruction Under Mk2/Mkn*

Description

Perform ancestral state reconstruction under Mk2 and other constant rate Markov models. Marginal, joint, and stochastic reconstructions are supported. Documentation is still in an early stage, and mostly in terms of examples.

Usage

```
## S3 method for class 'mkn'
make.asr.marginal(lik, ...)
## S3 method for class 'mkn'
make.asr.joint(lik, ...)
## S3 method for class 'mkn'
make.asr.stoch(lik, slim=FALSE, ...)
```

Arguments

`lik`	A likelihood function, returned by `make.mk2` or `make.mkn`.
`slim`	Should the history object be slimmed down?
`...`	Additional arguments; currently ignored.

Details

Output will differ slightly when `mk2` and `mkn` models are used as `lik`, as `mk2` uses states 0/1, while 2-state `mkn` uses 1/2.

This is all quite slow. Faster versions are coming eventually.

These functions all return functions that generate different types of ancestral reconstruction.

Author(s)

Richard G. FitzJohn

Examples

```
## Start with a simple tree evolved under a constant rates birth-death
## model with asymetric character evolution
pars <- c(.1, .1, .03, .03, .03, .06)
set.seed(1)
phy <- trees(pars, "bisse", max.taxa=50, max.t=Inf, x0=0)[[1]]

## Here is the true history.  The root node appears to be state 1 (red)
## at the root, despite specifying a root of state 0 (x0=0, in statement
## above).  This is because the tree started with a single lineage, but
## had changed state by the time the first speciation event happened.
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy, main="True history")

## All of the methods need a likelihood function; build a mk2 function:
lik <- make.mk2(phy, phy$tip.state)

## Using the true parameters, compute the marginal ancestral state
## reconstructions:
st.m <- asr.marginal(lik, pars[5:6])

## There is still not a good stand-alone plotting command for nodes.
## For now, use ape's nodelabels().
plot(h, phy, main="Marginal ASR", show.node.state=FALSE)
nodelabels(thermo=t(st.m), piecol=1:2, cex=.5)

## Again, with the true parameters, a sample from the joint
## distribution:
st.j <- asr.joint(lik, pars[5:6])

## Plotting this sample against the true values.
plot(h, phy, main="Joint ASR", show.node.state=FALSE)
nodelabels(pch=19, col=st.j + 1)

## This is just one sample, and is not very accurate in this case!  Make
## 1,000 such samples and average them:
st.j2 <- asr.joint(lik, pars[5:6], 1000)
st.j2.mean <- colMeans(st.j2)

plot(h, phy, main="Joint ASR (averaged)", show.node.state=FALSE)
nodelabels(thermo=1-st.j2.mean, piecol=1:2, cex=.5)

## Check the estimates against one another:
plot(st.m[2,], st.j2.mean, xlab="Marginal", ylab="Joint", las=1)
```

```
abline(0, 1)

## Finally, the stochastic character mapping.  This uses samples from
## the joint distribution at its core.
st.s <- asr.stoch(lik, pars[5:6])
plot(st.s, phy)

## Again, multiple samples can be done at once.  There is a function for
## summarising histories, but it is still in the works.

## Repeating the above with a two-state mkn model:
lik2 <- make.mkn(phy, phy$tip.state + 1, 2, FALSE)

## Everything works:
st2.m <- asr.marginal(lik2, pars[5:6])
st2.j <- asr.joint(lik2, pars[5:6], 100)
st2.s <- asr.stoch(lik2, pars[5:6])

## Marginal likelihoods agree:
all.equal(st.m, st2.m)
## Joint reconstructions are stochastic, so just check with a
## regression:
summary(lm(colMeans(st2.j) - 1 ~ colMeans(st.j2) - 1))

## Integrate parameter uncertainty, and see how far down the tree there
## is any real information on parameter states for this tree (this takes
## about 6s)
## Not run:
set.seed(1)
prior <- make.prior.exponential(.5)
samples <- mcmc(lik, pars[5:6], 1000, w=1, prior=prior, print.every=100)
st.m.avg <- rowMeans(apply(samples[2:3], 1, asr.joint, lik=lik))

plot(h, phy, main="MCMC Averaged ASR", show.node.state=FALSE)
nodelabels(thermo=1 - st.m.avg, piecol=1:2, cex=.5)

## End(Not run)
```

---

check *Check Capabilities of the Diversitree Install*

Description

These check to see if FFTW or CVODES support was included in diversitree. They rarely need to be called directly.

Usage

```
check.fftC(error=TRUE)
check.cvodes(error=TRUE)
```

Arguments

error	Logical: causes an error if the FFTW or CVODES are not available if TRUE

Author(s)

Richard G. FitzJohn

`combine` *Combine Several Likelihood Functions Multiplicatively*

Description

Combine several likelihood functions, so that the new functions gives the product of all likelihoods (the sum of the log likelihoods). This assumes that all likelihoods are independent from one another!

This function is little tested. Use at your own risk!

Usage

```
combine(liks)
```

Arguments

`liks`	A list of likelihood functions. All must be of the same type, with the same argnames, and not constrained.

Author(s)

Richard G. FitzJohn

`constrain` *Constrain Parameters of a Model*

Description

Constrain a model to make submodels with fewer parameters. If `f` is a function that takes a vector `x` as its first argument, this function returns a new function that takes a shorter vector `x` with some elements constrained in some way; parameters can be fixed to particular values, constrained to be the same as other parameters, or arbitrary expressions of free parameters.

Usage

```
constrain(f, ..., formulae=NULL, names=argnames(f), extra=NULL)
constrain.i(f, p, i.free)
```

Arguments

`f`	A function to constrain.
`...`	Formulae indicating how the function should be constrained (see Details and Examples).
`formulae`	Optional list of constraints, possibly in addition to those in `...`
`names`	Optional Character vector of names, the same length as the number of parameters in `x`. Use this only if `argnames` does not return a vector for your function. Generally this should not be used.
`extra`	Optional vector of additional names that might appear on the RHS of constraints but do not represent names in the function's `argnames`. This can be used to set up dummy variables (example coming later).
`p`	A parameter vector (for `constrain.i`) indicating values for all parameters.
`i.free`	Indices of the parameters that are **not** constrained. Other indices will get the value in `p`. The element of `p[i.free]` will never be used and can be zero, `NA`, or any other value.

Details

The relationships are specified in the form `target ~ rel`, where `target` is the name of a vector to be constrained, and `rel` is some relationship. For example `lambda0 ~ lambda1` would have the effect of making the parameters `lambda0` and `lambda1` take the same value.

The `rel` term can be a constant (e.g., `target ~ 0`), another parameter (as above) or some expression of the parameters (e.g., `lambda0 ~ 2 * lambda1` or `lambda0 ~ lambda1 - mu1`).

Terms that appear on the right hand side of an expression may not be constrained in another expression, and no term may be constrained twice.

Value

This function returns a constrained function that can be passed through to `find.mle` and `mcmc`. It will behave like any other function. However, it has a modified `class` attribute so that some methods will dispatch differently (`argnames`, for example). All arguments in addition to `x` will be passed through to the original function `f`.

For help in designing constrained models, the returned function has an additional argument `pars.only`, when this is `TRUE` the function will return a named vector of arguments rather than evaluate the function (see Examples).

Warning

Only a few checks are done to ensure that the resulting function makes any sense; it is possible that I have missed some cases. There is currently no way of modifying constrained functions to remove the constraints. These weaknesses will be addressed in a future version.

Author(s)

Richard G. FitzJohn

Examples

```
## Same example likelihood function as for \link{find.mle} - BiSSE on a
## tree with 203 species, generated with an asymmetry in the speciation
## rates.
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 <- make.bisse(phy, phy$tip.state)

argnames(lik) # Canonical argument names

## Specify equal speciation rates
lik.2 <- constrain(lik, lambda0 ~ lambda1)
argnames(lik.2) # Note lambda0 now missing

## On constrained functions, use the "pars.only" argument to see what
## the full argument list would be:
lik.2(c(.1, pars[3:6]), pars.only=TRUE)

## Check this works:
lik(c(.1, .1, pars[3:6])) == lik.2(c(.1, pars[3:6]))

## For optimisation of these functions, see \link{find.mle}, which
## includes an example.

## More complicated; constrain lambda0 to half of lambda1, constrain mu0
## to be the same mu1, and set q01 equal to zero.
lik.3 <- constrain(lik, lambda0 ~ lambda1 / 2, mu0 ~ mu1, q01 ~ 0)
argnames(lik.3) # lambda1, mu1, q10
lik(c(.1, .2, .03, .03, 0, .01)) == lik.3(c(.2, .03, .01))

## Alternatively, coefficients can be specified using a list of
## constraints:
cons <- list(lambda1 ~ lambda0, mu1 ~ mu0, q10 ~ q01)
constrain(lik, formulae=cons)

## Using the "extra" argument allows recasting things to dummy
## parameters.  Here both lambda0 and lambda1 are mapped to the
## parameter "lambda":
lik.4 <- constrain(lik, lambda0 ~ lambda, lambda1 ~ lambda, extra="lambda")
argnames(lik.4)

## constrain.i can be useful for setting a number of values at once.
## Suppose we wanted to look at the shape of the likelihood surface with
## respect to one parameter around the ML point.  For this tree, the ML
## point is approximately:
p.ml <- c(0.09934, 0.19606, 0.02382, 0.03208, 0.01005, 0.00982)

## Leaving just lambda1 (which is parameter number 2) free:
lik.l1 <- constrain.i(lik, p.ml, 2)

## The function now reports that five of the parameters are constrained,
## with one free (lambda1)
lik.l1

## Likewise:
```

```
argnames(lik.l1)

## Looking in the neighbourhood of the ML point, the likelihood surface
## is approximately quadratic:
pp <- seq(p.ml[2] - .02, p.ml[2] + .02, length.out=15)
yy <- sapply(pp, lik.l1)
plot(yy ~ pp, type="b", xlab="lambda 1", ylab="Log likelihood")
abline(v=p.ml[2], col="red", lty=2)

## pars.only works as above, returning the full parameter vector
lik.l1(p.ml[2], pars.only=TRUE)
identical(p.ml, lik.l1(p.ml[2], pars.only=TRUE))
```

`find.mle` *Maximimum Likelihood Inference*

Description

Find the maximum likelihood point of a model by nonlinear optimisation. `find.mle` is generic, and allows different default behaviour for different likelihood functions.

Usage

```
find.mle(func, x.init, method, ...)
## S3 method for class 'fit.mle'
coef(object, full=FALSE, extra=FALSE, ...)
## S3 method for class 'fit.mle'
logLik(object, ...)
## S3 method for class 'fit.mle'
anova(object, ..., sequential=FALSE)
```

Arguments

`func`	A likelihood function. This is assumed to return the log likelihood (see Details). The function must take a vector of parameters as the first argument.
`x.init`	Initial starting point for the optimisation.
`method`	Method to use for optimisation. May be one of "optim", "subplex", "nlminb", "nlm" (partial unambigious string is allowed).
`...`	For `find.mle`, additional arguments passed through to the methods, optimisation routines, or to the likelihood function `func` - see Details. For `anova`, this is one or more models to compare against the model `object` (either submodels or supermodels or the test is meaningless).
`object`	A fitted model, returned by `find.mle`.
`full`	When returning the coefficients for a constrained model, should be coefficients for the underlying constrained model be returned?
`extra`	When returning the coefficients for a constrained model, should dummy "extra" parameters be returned as well?
`sequential`	Should `anova` treat the models as a series of increasing complexity? Currently this is a little overzealous in checking and will refuse to work if the likelihood values are not strictly increasing.

Details

find.mle starts a search for the maximum likelihood (ML) parameters from a starting point x.init. x.init should be the correct length for func, so that func(x.init) returns a valid likelihood. However, if func is a constrained function (via constrain) and x.init is the correct length for the unconstrained function then an attempt will be made to guess a valid starting point. This will often do poorly and a warning will be given.

Different methods will be dispatched for different types of likelihood functions. Currently all models in diversitree are supported (bisse, geosse, mk2, mkn, bd, and yule). With the exception of the Yule pure-birth process, these methods just specify different default arguments for the underlying optimisation routines (the Yule model has an analytical solution, and no optimisation step is required). Generally, it will not be necessary to specify the method argument to find.mle as a sensible method is chosen during dispatch.

The ... argument may contain additional arguments for the function func. This includes things like condition.surv for conditioning on survival in BiSSE, birth-death, and Yule models. Specify this as

```
find.mle(lik, x.init, condition.surv=TRUE)
```

(see the Examples).

Different method arguments take different arguments passed through ... to control their behaviour:

method="optim": Uses R's optim function for the optimisation. This allows access to a variety of general purpose optimisation algorithms. The method *within* optim can be chosen via the argument optim.method, which is set to "L-BFGS-B" by default (box constrained quasi-Newton optimisation). This should be suitable for most uses. See the method argument of optim for other possibilities. If "L-BFGS-B" is used, then upper and lower bounds may be specified by the arguments lower and upper. The argument control can be used to specify other control parameters for the algorithms - see optim for details. Most of the optim algorithms require finite values be returned at every evaluated point. This is often not possible (extreme values of parameters or particular combinations may have zero likelihood and therefore -Inf log-likelihood). To get around this, the argument fail.value can be used to specify a fallback value. By default this is set to func(x.init) - 1000, which should work reasonably well for most cases.

method="subplex": Uses the "subplex" algorithm (a variant of the downhill simplex/Nelder-Mead algorithm that uses Nelder-Mead on a sequence of subspaces). This algorithm generally requires more evaluations than optim-based optimisation, but does not require approximation of derivatives and seems to find the global optimum more reliably (though often less precisely). Additional arguments are control to control aspects of the search (see subplex for details). The argument fail.value can be used as in method="optim", but by default -Inf will be used on failure to evaluate, which is generally appropriate.

method="nlminb": Uses the function nlminb for optimisation, so that optimising a Mk2/Mkn likelihood function behaves as similarly as possible to ape's ace function. As for method="optim", lower and upper bounds on parameters may be specified via lower and upper. fail.value can be used to control behaviour on evaluation failure, but like method="subplex", -Inf is used which should work in most cases. Additional control parameters may be passed via control - see link{nlminb} for details. This function is not generally recommended for use.

method="nlm": Uses the function nlm for optimisation, so that optimising a birth-death likelihood function behaves as similarly as possible to ape's birthdeath function. Takes the same additional arguments as method="nlminb" (except that fail.value behaves as for method="optim"). Like method="nlminb", this is not recommended for general use.

code and logLik methods exist for fit.mle objects so that parameters and log-likelihoods may be extracted. This also allows use with AIC.

Simple model comparison by way of likelihood ratio tests can be performed with anova. See Examples for usage.

Value

A list of class fit.mle, with at least the components

- par The estimated parameters.
- lnLik The log likelihood at the ML point.
- counts The number of function evaluations performed during the search.
- code Convergence code. See the documentation for the underlying optimisation method for meaning, but "0" is usually good.
- func The likelihood function used in the fit.
- method The optimisation method used.

Model comparison

The anova function carries out likelihood ratio tests. There are a few possible configurations.

First, the first fit provided could be the focal fit, and all other fits are either special cases of it (every additional model is nested within the focal model) or generalisations of it (the focal model is nested within every additional model).

Second, the models could be sequential series of fits (if sequential=TRUE), such that models (A, B, C, D) are to be compared A vs. B, B vs. C, C vs. D. The models can either be strictly increasing in parameters (A nested in B, B nested in C, ...) or strictly decreasing in parameters (D nested in C, C nested in B, ...).

In both cases, nestedness is checked. First, the "class" of the fitted object must match. Second, the argnames of the likelihood function of a sub model must all appear in the argnames of the parent model. There are some cases where this second condition may not be satisfied and yet the comparison is valid (e.g., comparing a time-varying model against a non time varying model, and some make.quasse fits). We attempt to detect this but it may fail on some valid comparisons and silently allow some invalid comparisons.

Author(s)

Richard G. FitzJohn

Examples

```
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)

## Here is the 203 species tree with the true character history coded.
## Red is state '1', which has twice the speciation rate of black (state
## '0').
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy, cex=.5, show.node.state=FALSE)

## Make a BiSSE likelihood function
lik <- make.bisse(phy, phy$tip.state)
```

```
lik(pars)

## This takes ~30s to run, so is not enabled by default
## Not run:
## Fit the full six-parameter model
fit <- find.mle(lik, pars)
fit[1:2]

coef(fit)   # Named vector of six parameters
logLik(fit) # -659.93
AIC(fit)    # 1331.86

## find.mle works with constrained models (see \link{constrain}).  Here
## the two speciation rates are constrained to be the same as each
## other.
lik.l <- constrain(lik, lambda0 ~ lambda1)
fit.l <- find.mle(lik.l, pars[-2])
logLik(fit.l) # 663.41

## Compare the models with \link{anova} - this shows that the more
## complicated model with two separate speciation rates fits
## significantly better than the simpler model with equal rates
## (p=0.008).
anova(fit, equal.lambda=fit.l)

## You can return the parameters for the full six parameter model from
## the fitted five parameter model - this makes a good starting point
## for a ML search.
coef(fit.l, full=TRUE)

## End(Not run)
```

`geosse.phy` *Example GeoSSE Tree*

Description

This is a test GeoSSE tree, created by Emma Goldberg

Usage

```
geosse.phy
```

Format

An 'ape' tree, with additional element `tip.state` that includes the tip states.

Source

Generated from a C program by Emma Goldberg

history.from.sim — *Extract Character Histories From Simulations*

Description

This function extracts a history object from a simulated phylogeny produced by `tree.bisse`.

Usage

```
history.from.sim.discrete(phy, states)
```

Arguments

`phy`	A phylogeny produced by `tree.bisse`.
`states`	Possible states. For `tree.bisse` this should be `0:1`.

Author(s)

Richard G. FitzJohn

make.bd — *Constant Rate Birth-Death Models*

Description

Prepare to run a constant rate birth-death model on a phylogenetic tree. This fits the Nee et al. 1994 equation, duplicating the `birthdeath` function in ape. Differences with that function include (1) the function is not constrained to positive diversification rates (mu can exceed lambda), (2) [eventual] support for both random taxon sampling and unresolved terminal clades (but see `bd.ext`), and (3) run both MCMC and MLE fits to birth death trees.

Usage

```
make.bd(tree, sampling.f=NULL, unresolved=NULL, times=NULL, control=list())
make.yule(tree, sampling.f=NULL, unresolved=NULL, times=NULL, control=list())
starting.point.bd(tree, yule=FALSE)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`times`	Vector of branching times, as returned by `branching.times`. You don't need to use this unless you know that you need to use this. Don't use it at the same time as `tree`.
`sampling.f`	Probability of an extant species being included in the phylogeny (sampling fraction). By default, all extant species are assumed to be included.

`unresolved` Unresolved clade information. This is a named vector, with the number of species as the value and names corresponding to tip labels. Tips that represent a single species should not be included in this vector. For example `sp1=10`, `sp2=2`, would mean that `sp1` represents 10 species, while `sp2` represents two. These labels must exist in `tree$tip.label` and all other tips are assumed to represent one species.

`yule` Should the starting point function return a Yule model (zero extinction rate)?

`control` List of control parameters. The element `method` can be either `nee` or `ode` to compute the likelihood using the equation from Nee et al. (1994) or in a BiSSE-style ODE approach respectively. `nee` should be faster, and `ode` is provided for completeness (and forms the basis of other methods). When `ode` is selected, other elements of `control` affect the behaviour of the ODE solver: see details in `make.bisse`.

Details

`make.bd` returns a function of class `bd`. This function has argument list (and default values)

```
f(pars, prior=NULL, condition.surv=TRUE)
```

The arguments are interpreted as

- `pars` A vector of two parameters, in the order `lambda`, `mu`.
- `prior`: a valid prior. See `make.prior` for more information.
- `condition.surv` (logical): should the likelihood calculation condition on survival of two lineages and the speciation event subtending them? This is done by default, following Nee et al. 1994.

The function "ode" method is included for completeness, but should not be taken too seriously. It uses an alternative ODE-based approach, more similar to most diversitree models, to compute the likelihood. It exists so that other models that extend the birth-death models may be tested.

Author(s)

Richard G. FitzJohn

References

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.

See Also

`constrain` for making submodels, `find.mle` for ML parameter estimation, `mcmc` for MCMC integration, and `make.bisse` for state-dependent birth-death models.

Examples

```
## Simulate a tree under a constant rates birth-death model and look at
## the maximum likelihood speciation/extinction parameters:
set.seed(1)
phy <- trees(c(.1, .03), "bd", max.taxa=25)[[1]]
lik <- make.bd(phy)
```

```
## By default, optimisation gives a lambda close to 0.1 and extremely
## small mu:
fit <- find.mle(lik, c(.1, .03))
coef(fit)

## The above optimisation uses the algorithm \link{nlm} for
## compatibility with ape's \link{birthdeath}.  This can be slightly
## improved by using \link{optim} for the optimisation, which allows
## bounds to be specified:
fit.o <- find.mle(lik, c(.1, .03), method="optim", lower=0)
coef(fit.o)

logLik(fit.o) - logLik(fit) # slight improvement

## Special case methods are worked out for the Yule model, for which
## analytic solutions are available.  Compare a direct fit of the Yule
## model with one where mu is constrained to be zero:
lik.yule <- make.yule(phy)
lik.mu0 <- constrain(lik, mu ~ 0)

## The same to a reasonable tolerance:
fit.yule <- find.mle(lik.yule, .1)
fit.mu0 <- find.mle(lik.mu0, .1)
all.equal(fit.yule[1:2], fit.mu0[1:2], tolerance=1e-6)

## There is no significant improvement in the fit by including the mu
## parameter (unsurprising as the ML value was zero)
anova(fit.o, yule=fit.yule)

## Optimisation can be done without conditioning on survival:
fit.nosurv <- find.mle(lik, c(.1, .03), method="optim", lower=0,
                       condition.surv=FALSE)
coef(fit.nosurv) # higher lambda than before

## Look at the marginal likelihoods, computed through MCMC (see
## \link{mcmc} for details, and increase nsteps for smoother
## plots [takes longer]).
samples <- mcmc(lik, fit$par, nsteps=2000,
                lower=c(-Inf, -Inf), upper=c(Inf, Inf), w=c(.1, .1),
                fail.value=-Inf, print.every=200)
samples$r <- with(samples, lambda - mu)

## Plot the profiles (see \link{profiles.plot}).
## The vertical lines are the simulated parameters, which match fairly
## well with the estimated ones.
col <- c("red", "blue", "green3")
profiles.plot(samples[c("lambda", "mu", "r")], col.line=col, las=1,
              legend="topright")
abline(v=0, lty=2)
abline(v=c(.1, .03, .07), col=col)

## Sample the phylogeny to include 20 of the species, and run the
## likelihood search assuming random sampling:
set.seed(1)
phy2 <- drop.tip(phy, sample(25, 5))
lik2 <- make.bd(phy2, sampling.f=20/25)
```

```
fit2 <- find.mle(lik2, c(.1, .03))

## The ODE based version gives comparable results.  However, it is
## about 12x slower.
lik.ode <- make.bd(phy, control=list(method="ode"))
all.equal(lik.ode(coef(fit)), lik(coef(fit)), tolerance=2e-7)

## The CVODES-based ODE version is about 4.5 slower
if ( check.cvodes(FALSE) ) {
  lik.ode <- make.bd(phy, control=list(method="ode", backend="CVODES"))
  all.equal(lik.ode(coef(fit)), lik(coef(fit)), tolerance=3e-7)
}
```

`make.bd.split` *Constant Rate Birth-Death Models: Split Models*

Description

Create a likelihood function for a birth-death model where the tree is partitioned into regions with different parameters.

Usage

```
make.bd.split(tree, nodes, split.t, sampling.f=NULL, unresolved=NULL)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`nodes`	Vector of nodes that will be split (see Details).
`split.t`	Vector of split times, same length as `nodes` (see Details).
`sampling.f`	Probability of an extant species being included in the phylogeny (sampling fraction). By default, all extant species are assumed to be included.
`unresolved`	Unresolved clade information. This is a named vector, with the number of species as the value and names corresponding to tip labels. Tips that represent a single species should not be included in this vector. For example `sp1=10`, `sp2=2`, would mean that `sp1` represents 10 species, while `sp2` represents two. These labels must exist in `tree$tip.label` and all other tips are assumed to represent one species.

Details

Branching times can be controlled with the `split.t` argument. If this is `Inf`, split at the base of the branch (as in MEDUSA). If `0`, split at the top (closest to the present, as in the new option for MEDUSA). If `0 < split.t < Inf` then we split at that time on the tree (zero is the present, with time growing backwards).

This function is related to MEDUSA (Alfaro et al. 2009), but does not include any of the code for efficiently moving between different splits (split creation here is fairly slow). The primary use for this model is for generating starting points for state dependent split models (e.g., `make.bisse.split`) and testing *a priori* splits.

Author(s)

Richard G. FitzJohn

Examples

```
set.seed(1)
pars <- c(.1, .03)
phy <- trees(pars, "bd", max.taxa=30)[[1]]

## Here is the phylogeny:
plot(phy, show.node.label=TRUE, label.offset=.1, font=1, cex=.75,
     no.margin=TRUE)

## Construct the plain likelihood function as a benchmark:
lik <- make.bd(phy)
lik(pars) # -21.74554

## Split this phylogeny at three points: nd11, nd13 and nd26
nodes <- c("nd11", "nd13", "nd26")

## This is the index in ape's node indexing:
nodes.i <- match(nodes, phy$node.label) + length(phy$tip.label)

nodelabels(node=nodes.i, pch=19, cex=2, col="#FF000099")

## To make a split likelihood function, pass the node locations and times in:
lik.s <- make.bd.split(phy, nodes)

## The parameters must be a list of the same length as the number of
## partitions.  Partition '1' is the root partition, and partition i is
## the partition rooted at the node[i-1]
pars4 <- rep(pars, 4)
names(pars4) <- argnames(lik.s)

## Run the likelihod calculation:
lik.s(pars4) # -21.74554

## These are basically identical (to acceptable tolerance)
lik.s(pars4) - lik(pars)

## You can use the labelled nodes rather than indices:
lik.s2 <- make.bd.split(phy, nodes)
identical(lik.s(pars4), lik.s2(pars4))

## All the usual ML/MCMC functions work as before:
fit <- find.mle(lik.s, pars4)
```

`make.bd.t` *Time-varing Birth-Death Models*

Description

Create a likelihood function for the birth-death model, where birth and/or death rates are arbitrary functions of time.

Usage

```
make.bd.t(tree, functions, sampling.f=NULL, unresolved=NULL,
          control=list())
```

Arguments

tree	A phylogenetic tree, in ape "phylo" format.
functions	A named list of functions of time. See details.
sampling.f	Probability of an extant species being included in the phylogeny (sampling fraction). By default, all extant species are assumed to be included.
unresolved	Not yet included: present in the argument list for future compatibility with make.bd.
control	List of control parameters for the ODE solver. See details in make.bisse.

Details

There is significant overhead in using this function, compared with the plain birth-death (make.bd) models. Usually, all of the calculations are done directly, or in C for the ODE based models. However, because of the generality here – parameters may be **any** function of time – we must call back to R many times during the integration process. The overhead is something like a factor of 10 compared with the ODE-based constant rate birth death model, and a factor of 40 compared with the direct-calculation constant rate birth death model.

Author(s)

Richard G. FitzJohn

Examples

```
## First, show equivalence to the plain Birth-death model.  This is not
## a very interesting use of the functions, but it serves as a useful
## check.

## Here is a simulated 25 species tree for testing.
set.seed(1)
pars <- c(.1, .03)
phy <- trees(pars, "bd", max.taxa=25)[[1]]

## Next, make three different likelihood functions: a "normal" one that
## uses the direct birth-death calculation, an "ode" based one (that
## uses numerical integration to compute the likelihood, and is
## therefore not exact), and one that is time-varying, but that the
## time-dependent functions are constant.t().
lik.direct <- make.bd(phy)
lik.ode <- make.bd(phy, control=list(method="ode"))
lik.t <- make.bd.t(phy, list(constant.t, constant.t))

lik.direct(pars) # -22.50267

## ODE-based likelihood calculations are correct to about 1e-6.
lik.direct(pars) - lik.ode(pars)

## The ODE calculation agrees exactly with the time-varying (but
## constant) calculation.
```

```
lik.ode(pars) - lik.t(pars)

## Next, make a real case, where speciation is a linear function of
## time.
lik.t2 <- make.bd.t(phy, list(linear.t, constant.t))

## Confirm that this agrees with the previous calculations when the
## slope is zero
pars2 <- c(pars[1], 0, pars[2])
lik.t2(pars2) - lik.t(pars)

## However, notice that this is much slower than the previous versions:
system.time(lik.direct(pars)) # ~ 0.001
system.time(lik.ode(pars))    # ~ 0.004
system.time(lik.t(pars))      # ~ 0.020
system.time(lik.t2(pars2))    # ~ 0.040

fit <- find.mle(lik.direct, pars)
fit.t2 <- find.mle(lik.t2, pars2)

## No significant improvement in model fit:
anova(fit, time.varying=fit.t2)
```

`make.bisse` *Binary State Speciation and Extinction Model*

Description

Prepare to run BiSSE (Binary State Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.bisse(tree, states, unresolved=NULL, sampling.f=NULL, nt.extra=10,
           strict=TRUE, control=list())
starting.point.bisse(tree, q.div=5, yule=FALSE)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be 0 or 1, or `NA` if the state is unknown. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.
`unresolved`	Unresolved clade information: see section below for structure.
`sampling.f`	Vector of length 2 with the estimated proportion of extant species in state 0 and 1 that are included in the phylogeny. A value of `c(0.5, 0.75)` means that half of species in state 0 and three quarters of species in state 1 are included in the phylogeny. By default all species are assumed to be known.
`nt.extra`	The number of species modelled in unresolved clades (this is in addition to the largest observed clade).

control	List of control parameters for the ODE solver. See details below.
strict	The states vector is always checked to make sure that the values are 0 and 1 only. If strict is TRUE (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing.
q.div	Ratio of diversification rate to character change rate. Eventually this will be changed to allow for Mk2 to be used for estimating q parameters.
yule	Logical: should starting parameters be Yule estimates rather than birth-death estimates?

Details

make.bisse returns a function of class bisse. This function has argument list (and default values)

```
f(pars, condition.surv=TRUE, root=ROOT.OBS, root.p=NULL,
  intermediates=FALSE)
```

The arguments are interpreted as

- pars A vector of six parameters, in the order lambda0, lambda1, mu0, mu1, q01, q10.
- condition.surv (logical): should the likelihood calculation condition on survival of two lineages and the speciation event subtending them? This is done by default, following Nee et al. 1994.
- root: Behaviour at the root (see Maddison et al. 2007, FitzJohn et al. 2009). The possible options are
  - ROOT.FLAT: A flat prior, weighting D_0 and D_1 equally.
  - ROOT.EQUI: Use the equilibrium distribution of the model, as described in Maddison et al. (2007).
  - ROOT.OBS: Weight D_0 and D_1 by their relative probability of observing the data, following FitzJohn et al. 2009:

$$D = D_0 \frac{D_0}{D_0 + D_1} + D_1 \frac{D_1}{D_0 + D_1}$$

  - ROOT.GIVEN: Root will be in state 0 with probability root.p[1], and in state 1 with probability root.p[2].
  - ROOT.BOTH: Don't do anything at the root, and return both values. (Note that this will not give you a likelihood!).
- root.p: Root weightings for use when root=ROOT.GIVEN. sum(root.p) should equal 1.
- intermediates: Add intermediates to the returned value as attributes:
  - cache: Cached tree traversal information.
  - intermediates: Mostly branch end information.
  - vals: Root D values.

  At this point, you will have to poke about in the source for more information on these.

starting.point.bisse produces a heuristic starting point to start from, based on the character-independent birth-death model. You can probably do better than this; see the vignette, for example. bisse.starting.point is the same code, but deprecated in favour of starting.point.bisse - it will be removed in a future version.

Unresolved clade information

This must be a `data.frame` with at least the four columns

- `tip.label`, giving the name of the tip to which the data applies
- `Nc`, giving the number of species in the clade
- `n0`, `n1`, giving the number of species known to be in state 0 and 1, respectively.

These columns may be in any order, and additional columns will be ignored. (Note that column names are case sensitive).

An alternative way of specifying unresolved clade information is to use the function `make.clade.tree` to construct a tree where tips that represent clades contain information about which species are contained within the clades. With a `clade.tree`, the `unresolved` object will be automatically constructed from the state information in `states`. (In this case, `states` must contain state information for the species contained within the unresolved clades.)

ODE solver control

The differential equations that define the BiSSE model are solved numerically using deSolve's LSODA ODE solver or the CVODES solver from the SUNDIALS suite of solvers. The `control` argument to `make.bisse` controls the behaviour of the integrator. This is a list that may contain elements:

- `tol`: Numerical tolerance used for the calculations. The default value of `1e-8` should be a reasonable trade-off between speed and accuracy. Do not expect too much more than this from the abilities of most machines!
- `eps`: A value that when the sum of the D values drops below, the integration results will be discarded and the integration will be attempted again (the second-chance integration will divide a branch in two and try again, recursively until the desired accuracy is reached). The default value of `0` will only discard integration results when the parameters go negative. However, for some problems more restrictive values (on the order of `control$tol`) will give better stability.
- `backend`: Select the solver. The three options here are
  - `deSolve`: (the default for now). Use the LSODA solver from the `deSolve` package.
  - `cvodes`: Use the CVODES solver from the SUNDIALS suite.
  - `CVODES`: Use the CVODES solver, but also use an experimental, faster, function for the tree traversal.
- safe(logical) Use a safer, slower, integrator when using the deSolve integrator? Leave this alone (this will be automatically selected when appropriate).

`deSolve` is the only supported backend on Windows.

Author(s)

Richard G. FitzJohn

References

FitzJohn R.G., Maddison W.P., and Otto S.P. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Syst. Biol. 58:595-611.

Maddison W.P., Midford P.E., and Otto S.P. 2007. Estimating a binary character's effect on speciation and extinction. Syst. Biol. 56:701-710.

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.

See Also

constrain for making submodels, find.mle for ML parameter estimation, mcmc for MCMC integration, and make.bd for state-independent birth-death models.

The help pages for find.mle has further examples of ML searches on full and constrained BiSSE models.

Examples

```
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
set.seed(4)
phy <- tree.bisse(pars, max.t=30, x0=0)

## Here is the 52 species tree with the true character history coded.
## Red is state '1', which has twice the speciation rate of black (state
## '0').
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy)

lik <- make.bisse(phy, phy$tip.state)
lik(pars) # -159.71

## Heuristic guess at a starting point, based on the constant-rate
## birth-death model (uses \link{make.bd}).
p <- starting.point.bisse(phy)

## Start an ML search from this point.  This takes some time (~7s)
fit <- find.mle(lik, p, method="subplex")
logLik(fit) # -158.6875

## The estimated parameters aren't too far away from the real ones, even
## with such a small tree
rbind(real=pars,
      estimated=round(coef(fit), 2))

## Test a constrained model where the speciation rates are set equal
## (takes ~4s).
lik.l <- constrain(lik, lambda1 ~ lambda0)
fit.l <- find.mle(lik.l, p[-1], method="subplex")
logLik(fit.l) # -158.7357

## Despite the difference in the estimated parameters, there is no
## statistical support for this difference:
anova(fit, equal.lambda=fit.l)

## Run an MCMC.  Because we are fitting six parameters to a tree with
## only 50 species, priors will be needed.  I will use an exponential
## prior with rate 1/(2r), where r is the character independent
## diversificiation rate:
prior <- make.prior.exponential(1 / (2 * (p[1] - p[3])))

## This takes quite a while to run, so is not run by default
## Not run:
tmp <- mcmc(lik, fit$par, nsteps=100, prior=prior, w=.1, print.every=0)

w <- diff(sapply(tmp[2:7], range))
samples <- mcmc(lik, fit$par, nsteps=1000, prior=prior, w=w,
```

```
                print.every=100)

## See \link{profiles.plot} for more information on plotting these
## profiles.
col <- c("blue", "red")
profiles.plot(samples[c("lambda0", "lambda1")], col.line=col, las=1,
              xlab="Speciation rate", legend="topright")

## End(Not run)

## BiSSE reduces to the birth-death model and Mk2 when diversification
## is state independent (i.e., lambda0 ~ lambda1 and mu0 ~ mu1).
lik.mk2 <- make.mk2(phy, phy$tip.state)
lik.bd <- make.bd(phy)

## 1. BiSSE / Birth-Death
## Set the q01 and q10 parameters to arbitrary numbers (need not be
## symmetric), and constrain the lambdas and mus to be the same for each
## state.  The likelihood function now has just two parameters and
## will be proprtional to Nee's birth-death based likelihood:
lik.bisse.bd <- constrain(lik,
                          lambda1 ~ lambda0, mu1 ~ mu0,
                          q01 ~ .01, q10 ~ .02)
pars <- c(.1, .03)
## These differ by -167.3861 for both parameter sets:
lik.bisse.bd(pars)   - lik.bd(pars)
lik.bisse.bd(2*pars) - lik.bd(2*pars)

## 2. BiSSE / Mk2
## Same idea as above: set all diversification parameters to arbitrary
## values (but symmetric this time):
lik.bisse.mk2 <- constrain(lik,
                           lambda0 ~ .1, lambda1 ~ .1,
                           mu0 ~ .03, mu1 ~ .03)
## Differ by -150.4740 for both parameter sets.
lik.bisse.mk2(pars)   - lik.mk2(pars)
lik.bisse.mk2(2*pars) - lik.mk2(2*pars)

## 3. Sampled BiSSE / Birth-Death
## Pretend that the tree is only .6 sampled:
lik.bd2 <- make.bd(phy, sampling.f=.6)
lik.bisse2 <- make.bisse(phy, phy$tip.state, sampling.f=c(.6, .6))
lik.bisse2.bd <- constrain(lik.bisse2,
                           lambda1 ~ lambda0, mu1 ~ mu0,
                           q01 ~ .01, q10 ~ .01)

## Difference of -167.2876
lik.bisse2.bd(pars)   - lik.bd2(pars)
lik.bisse2.bd(2*pars) - lik.bd2(2*pars)

## 4. Unresolved clade BiSSE / Birth-Death
set.seed(1)
unresolved <- data.frame(tip.label=I(sample(phy$tip.label, 5)),
                         Nc=sample(2:10, 5), n0=0, n1=0)
unresolved.bd <- structure(unresolved$Nc, names=unresolved$tip.label)
lik.bisse3 <- make.bisse(phy, phy$tip.state, unresolved)
lik.bisse3.bd <- constrain(lik.bisse3,
```

```
                            lambda1 ~ lambda0, mu1 ~ mu0,
                            q01 ~ .01, q10 ~ .01)
lik.bd3 <- make.bd(phy, unresolved=unresolved.bd)

## Difference of -167.1523
lik.bisse3.bd(pars) - lik.bd3(pars)
lik.bisse3.bd(pars*2) - lik.bd3(pars*2)

## Not run:
## This only works when diversitree was built with CVODES support
## (currently not on Windows).  This support is still experimental.

## Different backends:
lik.1 <- make.bisse(phy, phy$tip.state, control=list(backend="deSolve"))
lik.2 <- make.bisse(phy, phy$tip.state, control=list(backend="cvodes"))
lik.3 <- make.bisse(phy, phy$tip.state, control=list(backend="CVODES"))

## All the calculations are very similar
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
(ll.1 <- lik.1(pars))
(ll.2 <- lik.2(pars))
(ll.3 <- lik.3(pars))
diff(range(c(ll.1, ll.2, ll.3)))

## However, the CVODES version is quite a bit faster than the other two
## (cvodes is often a bit slower than deSolve).

## Timings are on a 2008 Mac Pro
system.time(fit.1 <- find.mle(lik.1, pars)) # 4.5s
system.time(fit.2 <- find.mle(lik.2, pars)) # 6.0s
system.time(fit.3 <- find.mle(lik.3, pars)) # 1.6s

## All found the same ML point:
diff(range(fit.1$lnLik, fit.2$lnLik, fit.3$lnLik))

## End(Not run)
```

`make.bisse.split` *Binary State Speciation and Extinction Model: Split Models*

Description

Create a likelihood function for a BiSSE model where the tree is partitioned into regions with different parameters.

Usage

```
make.bisse.split(tree, states, nodes, split.t, unresolved=NULL,
                 sampling.f=NULL, nt.extra=10, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.

states	A vector of character states, each of which must be 0 or 1, or NA if the state is unknown. This vector must have names that correspond to the tip labels in the phylogenetic tree (tree$tip.label). For tips corresponding to unresolved clades, the state should be NA.
nodes	Vector of nodes that will be split (see Details).
split.t	Vector of split times, same length as nodes (see Details).
unresolved	Unresolved clade information: see section below for structure.
sampling.f	Vector of length 2 with the estimated proportion of extant species in state 0 and 1 that are included in the phylogeny. A value of c(0.5, 0.75) means that half of species in state 0 and three quarters of species in state 1 are included in the phylogeny. By default all species are assumed to be known.
nt.extra	The number of species modelled in unresolved clades (this is in addition to the largest observed clade).
strict	The states vector is always checked to make sure that the values are 0 and 1 only. If strict is TRUE (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing.
control	List of control parameters for the ODE solver. See details in make.bisse.

Details

Branching times can be controlled with the split.t argument. If this is Inf, split at the base of the branch (as in MEDUSA). If 0, split at the top (closest to the present, as in the new option for MEDUSA). If 0 < split.t < Inf then we split at that time on the tree (zero is the present, with time growing backwards).

TODO: Describe nodes and split.t here.

Author(s)

Richard G. FitzJohn

Examples

```
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
set.seed(546)
phy <- tree.bisse(pars, max.taxa=30, x0=0)

## Here is the phylogeny:
plot(phy, show.node.label=TRUE, label.offset=.1, font=1, cex=.75,
     no.margin=TRUE)

## Here is a plain BiSSE function for comparison:
lik.b <- make.bisse(phy, phy$tip.state)
lik.b(pars) # -93.62479

## Split this phylogeny at three points: nd15, nd18 and nd26
nodes <- c("nd15", "nd18", "nd26")

## This is the index in ape's node indexing:
nodes.i <- match(nodes, phy$node.label) + length(phy$tip.label)

nodelabels(node=nodes.i, pch=19, cex=2, col="#FF000099")
```

```
## To make a split BiSSE function, pass the node locations and times in:
lik.s <- make.bisse.split(phy, phy$tip.state, nodes.i)

## The parameters must be a list of the same length as the number of
## partitions.  Partition '1' is the root partition, and partition i is
## the partition rooted at the node[i-1]
pars4 <- rep(pars, 4)
pars4

## Run the likelihod calculation:
lik.s(pars4) # -93.62479

## These are basically identical (to acceptable tolerance)
lik.s(pars4) - lik.b(pars)

## You can use the labelled nodes rather than indices:
lik.s2 <- make.bisse.split(phy, phy$tip.state, nodes)
identical(lik.s(pars4), lik.s2(pars4))

## This also works where some tips are unresolved clades.  Here are a
## few:
unresolved <-
  data.frame(tip.label=c("sp12", "sp32", "sp9", "sp22", "sp11"),
             Nc=c(2,5,3,2,5), n0=c(1, 4, 3, 2, 4), n1=c(1, 1, 0, 0, 1))

## Plain BiSSE with unresolved clades:
lik.u.b <- make.bisse(phy, phy$tip.state, unresolved=unresolved)
lik.u.b(pars) # -139.3688

## Split BiSSE with unresolved clades:
lik.u.s <- make.bisse.split(phy, phy$tip.state, nodes,
                            unresolved=unresolved)
lik.u.s(pars4) # -139.3688

lik.u.b(pars) - lik.u.s(pars4) # numerical error only
```

`make.bisse.td` *Binary State Speciation and Extinction Model: Time Dependant Models*

Description

Create a likelihood function for a BiSSE model where different chunks of time have different parameters. This code is experimental!

Usage

```
make.bisse.td(tree, states, n.epoch, unresolved=NULL, sampling.f=NULL,
              nt.extra=10, strict=TRUE, control=list())
make.bisse.t(tree, states, functions, unresolved=NULL, sampling.f=NULL,
             nt.extra=10, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be 0 or 1, or `NA` if the state is unknown. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.
`n.epoch`	Number of epochs. 1 corresponds to plain BiSSE, so this will generally be an integer at least 2.
`functions`	A named list of functions of time. See details.
`unresolved`	Unresolved clade information: see `make.bisse`. (Currently this is not supported.)
`sampling.f`	Vector of length 2 with the estimated proportion of extant species in state 0 and 1 that are included in the phylogeny. See `make.bisse`.
`nt.extra`	The number of species modelled in unresolved clades (this is in addition to the largest observed clade).
`strict`	The `states` vector is always checked to make sure that the values are 0 and 1 only. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.

Details

This builds a BiSSE likelihood function where different regions of time (epochs) have different parameter sets. By default, all parameters are free to vary between epochs, so some constraining will probably be required to get reasonable answers.

For `n` epochs, there are `n-1` time points; the first `n-1` elements of the likelihood's parameter vector are these points. These are measured from the present at time zero, with time increasing towards the base of the tree. The rest of the parameter vector are BiSSE parameters; the elements `n:(n+6)` are for the first epoch (closest to the present), elements `(n+7):(n+13)` are for the second epoch, and so on.

For `make.bisse.t`, the `funtions` is a list of functions of time. For example, to have speciation rates be linear functions of time, while the extinction and character change rates be constant with respect to time, one can do

```
functions=rep(list(linear.t, constant.t), c(2, 4))
```

The functions here must have `t` as their first argument, interpreted as time back from the present. See `make.pars.t` for more information, and for some potentially useful time functions.

Author(s)

Richard G. FitzJohn

Examples

```
set.seed(4)
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
phy <- tree.bisse(pars, max.t=30, x0=0)
```

```
## Suppose we want to see if diversification is different in the most
## recent 3 time units, compared with the rest of the tree (yes, this is
## a totally contrived example!):
plot(phy)
axisPhylo()
abline(v=max(branching.times(phy)) - 3, col="red", lty=3)

## For comparison, make a plain BiSSE likelihood function
lik.b <- make.bisse(phy, phy$tip.state)

## Create the time-dependent likelihood function.  The final argument
## here is the number of 'epochs' that are allowed.  Two epochs is one
## switch point.
lik.t <- make.bisse.td(phy, phy$tip.state, 2)

## The switch point is the first argument.  The remaining 12 parameters
## are the BiSSE parameters, with the first 6 being the most recent
## epoch.
argnames(lik.t)

pars.t <- c(3, pars, pars)
names(pars.t) <- argnames(lik.t)

## Calculations are identical to a reasonable tolerance:
lik.b(pars) - lik.t(pars.t)

## It will often be useful to constrain the time as a fixed quantity.
lik.t2 <- constrain(lik.t, t.1 ~ 3)

## Parameter estimation under maximum likelihood.  This is marked "don't
## run" because the time-dependent fit takes a few minutes.
## Not run:
## Fit the BiSSE ML model
fit.b <- find.mle(lik.b, pars)

## And fit the BiSSE/td model
fit.t <- find.mle(lik.t2, pars.t[argnames(lik.t2)],
                  control=list(maxit=20000))

## Compare these two fits with a likelihood ratio test (lik.t2 is nested
## within lik.b)
anova(fit.b, td=fit.t)

## End(Not run)

## The time varying model (bisse.t)  is more general, but substantially
## slower.  Here, I will show that the two functions are equivalent for
## step function models.  We'll constrain all the non-lambda parameters
## to be the same over a time-switch at t=5.  This leaves 8 parameters.
lik.td <- make.bisse.td(phy, phy$tip.state, 2)
lik.td2 <- constrain(lik.td, t.1 ~ 5,
                     mu0.2 ~ mu0.1, mu1.2 ~ mu1.1,
                     q01.2 ~ q01.1, q10.2 ~ q10.1)

lik.t <- make.bisse.t(phy, phy$tip.state,
                        rep(list(stepf.t, constant.t), c(2, 4)))
lik.t2 <- constrain(lik.t, lambda0.tc ~ 5, lambda1.tc ~ 5)
```

```
## Note that the argument names for these functions are different from
## one another.  This reflects different ways that the functions will
## tend to be used, but is potentially confusing here.
argnames(lik.td2)
argnames(lik.t2)

## First, evaluate the functions with no time effect and check that they
## are the same as the base BiSSE model
p.td <- c(pars, pars[1:2])
p.t <- pars[c(1, 1, 2, 2, 3:6)]

## All agree:
lik.b(pars)   # -159.7128
lik.td2(p.td) # -159.7128
lik.t2(p.t)   # -159.7128

## In fact, the time-varying BiSSE will tend to be identical to plain
## BiSSE where the functions to not change:
lik.b(pars) - lik.t2(p.t)

## Slight numerical differences are typical for the time-chunk BiSSE,
## because it forces the integration to be carried out more carefully
## around the switch point.
lik.b(pars) - lik.td2(p.td)

## Next, evaluate the functions with a time effect (5 time units ago,
## speciation rates were twice the contemporary rate)
p.td2 <- c(pars, pars[1:2]*2)
p.t2 <- c(pars[1], pars[1]*2, pars[2], pars[2]*2, pars[3:6])

## Huge drop in the likelihood (from -159.7128 to -172.7874)
lik.b(pars)
lik.td2(p.td2)
lik.t2(p.t2)

## The small difference remains between the two approaches, but they are
## basically the same.
lik.td2(p.td2) - lik.t2(p.t2)

## There is a time cost to both time-dependent methods, but it is more
## heavily paid for the time-varying case:
system.time(lik.b(pars))
system.time(lik.td2(p.td)) # 1.9x slower than plain BiSSE
system.time(lik.t2(p.t))   # 10x  slower than plain BiSSE
```

`make.bisseness` *Binary State Speciation and Extinction (Node Enhanced State Shift) Model*

Description

Prepare to run BiSSE-ness (Binary State Speciation and Extinction (Node Enhanced State Shift)) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.bisseness(tree, states, unresolved=NULL, sampling.f=NULL,
               nt.extra=10, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be 0 or 1, or `NA` if the state is unknown. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.
`unresolved`	Unresolved clade information: see section below for structure.
`sampling.f`	Vector of length 2 with the estimated proportion of extant species in state 0 and 1 that are included in the phylogeny. A value of `c(0.5, 0.75)` means that half of species in state 0 and three quarters of species in state 1 are included in the phylogeny. By default all species are assumed to be known.
`nt.extra`	The number of "extra" species to include in the unresolved clade calculations. This is in addition to the largest included unresolved clade.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.
`strict`	The `states` vector is always checked to make sure that the values are 0 and 1 only. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present at least once in the tree. The likelihood models tend to be poorly behaved where a state is not represented on the tree.

Details

`make.bisse` returns a function of class `bisse`. This function has argument list (and default values) [RICH: Update to BiSSEness?]

```
f(pars, condition.surv=TRUE, root=ROOT.OBS, root.p=NULL,
  intermediates=FALSE)
```

The arguments are interpreted as

- `pars` A vector of 10 parameters, in the order `lambda0`, `lambda1`, `mu0`, `mu1`, `q01`, `q10`, `p0c`, `p0a`, `p1c`, `p1a`.
- `condition.surv` (logical): should the likelihood calculation condition on survival of two lineages and the speciation event subtending them? This is done by default, following Nee et al. 1994. For BiSSE-ness, equation (A5) in Magnuson-Ford and Otto describes how conditioning on survival alters the likelihood of observing the data.
- `root`: Behaviour at the root (see Maddison et al. 2007, FitzJohn et al. 2009). The possible options are
  - `ROOT.FLAT`: A flat prior, weighting D_0 and D_1 equally.
  - `ROOT.EQUI`: Use the equilibrium distribution of the model, as described in Maddison et al. (2007) using equation (A6) in Magnuson-Ford and Otto.
  - `ROOT.OBS`: Weight D_0 and D_1 by their relative probability of observing the data, following FitzJohn et al. 2009:

$$D = D_0 \frac{D_0}{D_0 + D_1} + D_1 \frac{D_1}{D_0 + D_1}$$

  - ROOT.GIVEN: Root will be in state 0 with probability root.p[1], and in state 1 with probability root.p[2].
  - ROOT.BOTH: Don't do anything at the root, and return both values. (Note that this will not give you a likelihood!).
- root.p: Root weightings for use when root=ROOT.GIVEN. sum(root.p) should equal 1.
- intermediates: Add intermediates to the returned value as attributes:
  - cache: Cached tree traversal information.
  - intermediates: Mostly branch end information.
  - vals: Root D values.

  At this point, you will have to poke about in the source for more information on these.

Unresolved clade information

This must be a data.frame with at least the four columns

- tip.label, giving the name of the tip to which the data applies
- Nc, giving the number of species in the clade
- n0, n1, giving the number of species known to be in state 0 and 1, respectively.

These columns may be in any order, and additional columns will be ignored. (Note that column names are case sensitive).

An alternative way of specifying unresolved clade information is to use the function make.clade.tree to construct a tree where tips that represent clades contain information about which species are contained within the clades. With a clade.tree, the unresolved object will be automatically constructed from the state information in states. (In this case, states must contain state information for the species contained within the unresolved clades.)

Author(s)

Karen Magnuson-Ford

References

FitzJohn R.G., Maddison W.P., and Otto S.P. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Syst. Biol. 58:595-611.

Maddison W.P., Midford P.E., and Otto S.P. 2007. Estimating a binary character's effect on speciation and extinction. Syst. Biol. 56:701-710.

Magnuson-Ford, K., and Otto, S.P. 2012. Linking the investigations of character evolution and species diversification. American Naturalist, in press.

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.

See Also

make.bisse for the model with no state change at nodes.

tree.bisseness for simulating trees under the BiSSE-ness model.

constrain for making submodels, find.mle for ML parameter estimation, mcmc for MCMC integration, and make.bd for state-independent birth-death models.

The help pages for find.mle has further examples of ML searches on full and constrained BiSSE models.

Examples

```
## First we simulat a 50 species tree, assuming cladogenetic shifts in
## the trait (i.e., the trait only changes at speciation).
## Red is state '1', black is state '0', and we let red lineages
## speciate at twice the rate of black lineages.
## The simulation starts in state 0.
set.seed(3)
pars <- c(0.1, 0.2, 0.03, 0.03, 0, 0, 0.1, 0, 0.1, 0)
phy <- tree.bisseness(pars, max.taxa=50, x0=0)
phy$tip.state

h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy)

## This builds the likelihood of the data according to BiSSEness:
lik <- make.bisseness(phy, phy$tip.state)
## e.g., the likelihood of the true parameters is:
lik(pars) # -174.7954

## ML search:  First we make hueristic guess at a starting point, based
## on the constant-rate birth-death model assuming anagenesis (uses
## \link{make.bd}).
startp <- starting.point.bisse(phy)

## We then take the total amount of anagenetic change expected across
## the tree and assign half of this change to anagenesis and half to
## cladogenetic change at the nodes as a heuristic starting point:
t <- branching.times(phy)
tryq <- 1/2 * startp[["q01"]] * sum(t)/length(t)
p <- c(startp[1:4], startp[5:6]/2, p0c=tryq, p0a=0.5, p1c=tryq, p1a=0.5)

## Start an ML search from this point.  This takes some time (~12s)
fit <- find.mle(lik, p, method="subplex")
logLik(fit) # -174.0104

## Compare the fit to a constrained model that only allows the trait
## to change along a lineage (anagenesis).  This takes some time (~12s)
lik.no.clado <- constrain(lik, p0c ~ 0, p1c ~ 0)
fit.no.clado <- find.mle(lik.no.clado,p[argnames(lik.no.clado)])
logLik(fit.no.clado) # -174.0577

## This is consistent with what BiSSE finds:
likB <- make.bisse(phy, phy$tip.state)
fitB <- find.mle(likB, startp, method="subplex")
logLik(fitB) # -174.0576

## With only this 50-species tree, there is no statistical support
## for the more complicated BiSSE-ness model that allows cladogenesis:
anova(fit, no.clado=fit.no.clado)
## Note that anova() performs a likelihood ratio test here.

## If the above is repeated with max.taxa=250, BiSSE-ness rejects the
## constrained model in favor of one that allows cladogenetic change.

## Not run:
## MCMC run: We use the ML estimate from the full model
```

```
## as a starting point.
##
## We shift all very small numbers up to 1e-4 to allow the derivatives
## to be calculated.
ml.start.pt <- pmax(coef(fit), 1e-4)

## Make exponential priors for the rate parameters and uniform priors
## for the cladogenetic change probability prarameters.
make.prior.exp_ness <- function(r, min=0, max=1) {
  function(pars) {
    sum(dexp(pars[1:6], rate=r, log=TRUE)) +
      sum(dunif(pars[7:10], min, max, log=TRUE))
  }
}

## Choosing the slice sampling parameter, w (affects speed):
library(numDeriv)
hess <- hessian(lik, ml.start.pt)
vcv <- -solve(hess)
sehess <- sqrt(abs(diag(vcv)))
w <- 2 * pmin(sehess, .2)

## Setting the priors
r <- log(length(phy$tip.label))/max(branching.times(phy))
prior <- make.prior.exp_ness(1/(2*r))
prior(ml.start.pt)

## Running the mcmc chain (only 10 steps are shown for illustration)
steps <- 10
set.seed(1) # For reproducibility
output <- mcmc(lik, ml.start.pt, nsteps=steps, w=w, prior=prior)

## Unresolved tip clade: Here we collapse one clade in the 50 species
## tree (involving sister species sp70 and sp71) and illustrate the use
## of BiSSEness with unresolved tip clades.
slimphy <- drop.tip(phy,c("sp71"))
states <- slimphy$tip.state[slimphy$tip.label]
states["sp70"] <- NA
unresolved <- data.frame(tip.label=c("sp70"), Nc=2, n0=2, n1=0)

## This builds the likelihood of the data according to BiSSEness:
lik.unresolved <- make.bisseness(slimphy, states, unresolved)
## e.g., the likelihood of the true parameters is:
lik.unresolved(pars) # -174.6575

## ML search from the heuristic starting point used above:
fit.unresolved <- find.mle(lik.unresolved, p, method="subplex")
logLik(fit.unresolved) # -173.9136

## End(Not run)
```

---

make.bm *Brownian Motion and Ornstein-Uhlenbeck Character Evolution*

Description

Create a likelihood function for models of simple Brownian Motion or Ornstein-Uhlenbeck (OU) character evolution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.bm(tree, states, states.sd=0, control=list())
make.ou(tree, states, states.sd=0, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of continuous valued character states. This vector must be named with the tip labels of `tree`.
`states.sd`	An optional vector of measurement errors, as standard deviation around the mean. If a single value is given it is used for all tips, otherwise the vector must be named as for `states`.
`control`	A list of control parameters. See details below.

Details

The `control` argument is a named list of options.

Currently the only option is `method`. Specifying `control=list(method="vcv")` uses a variance-covariance matrix based approach to compute the likelihood. This is similar to the approach used by geiger, and is the default for `make.bm` (but is not available for `make.ou`). The alternative is to specify `control=list(method="direct")` which uses the transition density function for brownian motion along each branch, similar to how most methods in diversitree are computed. This second approach is much faster for very large trees.

The VCV-based `make.bm` function is heavily based on `fitContinuous` in the `geiger` package.

Author(s)

Richard G. FitzJohn

Examples

```
## Random data (following APE)
data(bird.orders)
set.seed(1)
x <- structure(rnorm(length(bird.orders$tip.label)),
               names=bird.orders$tip.label)

## With the VCV approach
fit1 <- find.mle(make.bm(bird.orders, x), .1)

## With the direct calculations
lik.direct <- make.bm(bird.orders, x, control=list(method="direct"))
fit2 <- find.mle(lik.direct, .1)

## All the same (need to drop the function from this though)
all.equal(fit1[-7], fit2[-7])
```

```
## If this is the same as the estimates from Geiger, to within the
## tolerances expected for the calculation and optimisation:
## Not run:
fit3 <- fitContinuous(bird.orders, x)
all.equal(fit3$Trait1$lnl, fit1$lnLik)
all.equal(fit3$Trait1$beta, fit1$par, check.attributes=FALSE)

## End(Not run)
```

`make.clade.tree` *Make a "Clade Tree"*

Description

This function makes a "clade tree", where tips represent clades. It is designed to make working with unresolved clade information in `make.bisse` more straightforward. `clade.tree` objects have their own plotting methods.

Usage

```
make.clade.tree(tree, clades)
clades.from.polytomies(tree)
clades.from.classification(tree, class, check=TRUE)
```

Arguments

`tree`	A phylogenetic tree, in `phylo` (Ape) format.
`clades`	A list, where the name of each element represents a tip in `tree` and each element is a character vector containing the names of species contained within that clade.
`class`	A vector along `tree$tip.label`, indicating the class to which each tip belongs.
`check`	Logical, indicating whether a (potentially slow) check will be done to ensure that all resulting clades are reciprocally monophyletic within the tree.

Details

The idea here is that `make.bisse` takes a tree and a named character state vector. If the phylogenetic tree contains information about the membership of clades, then the unresolved clade information can be constructed automatically. The names chosen should therefore reflect the names used in the state information.

Note

Currently, `clade.tree` objects work poorly with some `ape` functions.

Author(s)

Richard G. FitzJohn

`make.classe` *Cladogenetic State change Speciation and Extinction Model*

Description

Prepare to run ClaSSE (Cladogenetic State change Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.classe(tree, states, k, sampling.f=NULL, strict=TRUE,
            control=list())
starting.point.classe(tree, k, eps=0.5)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be an integer between 1 and `k`. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`).
`k`	The number of states. (The maximum now is 31, but that can easily be increased if necessary.)
`sampling.f`	Vector of length `k` where `sampling.f[i]` is the proportion of species in state `i` that are present in the phylogeny. A value of `c(0.5, 0.75, 1)` means that half of species in state 1, three quarters of species in state 2, and all species in state 3 are included in the phylogeny. By default all species are assumed to be known
`strict`	The `states` vector is always checked to make sure that the values are integers on `1:k`. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.
`eps`	Ratio of extinction to speciation rates to be used when choosing a starting set of parameters. The procedure used is based on Magallon & Sanderson (2001).

Details

The ClaSSE model with `k = 2` is equivalent to but a different parameterization than the BiSSE-ness model. The GeoSSE model can be constructed from ClaSSE with `k = 3`; see the example below.

`make.classe` returns a function of class `classe`. The arguments and default values for this function are:

```
f(pars, condition.surv=TRUE, root=ROOT.OBS, root.p=NULL,
  intermediates=FALSE)
```

The arguments are of this function are explained in make.bisse. The speciation rate parameters are lambda_ijk, ordered with k changing fastest and insisting on j < k.

With more than 9 states, lambda_ijk and q_ij can be ambiguous (e.g. is q113 1->13 or 11->3?). To avoid this, the numbers are zero padded (so that the above would be q0113 or q1103 for 1->13 and 11->3 respectively). It might be easier to rename the arguments in practice though. More human-friendly handling of large speciation rate arrays is in the works.

`starting.point.classe` produces a first-guess set of parameters, ignoring character states.

Unresolved clade methods are not available for ClaSSE.

Tree simulation methods are not yet available for ClaSSE.

Author(s)

Emma E. Goldberg

References

FitzJohn R.G., Maddison W.P., and Otto S.P. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Syst. Biol. 58:595-611.

Goldberg E.E. and Igic B. Tempo and mode in plant breeding system evolution. In review.

Maddison W.P., Midford P.E., and Otto S.P. 2007. Estimating a binary character's effect on speciation and extinction. Syst. Biol. 56:701-710.

Magallon S. and Sanderson M.J. 2001. Absolute diversification rates in angiospem clades. Evol. 55:1762-1780.

Magnuson-Ford, K., and Otto, S.P. 2012. Linking the investigations of character evolution and species diversification. American Naturalist, in press.

See Also

`constrain` for making submodels, `find.mle` for ML parameter estimation, and `mcmc` for MCMC integration. The help page for `find.mle` has further examples of ML searches on full and constrained BiSSE models. Things work similarly for ClaSSE, just with different speciation parameters.

`make.bisse`, `make.bisseness`, `make.geosse`, `make.musse` for similar models and further relevant examples.

Examples

```
## GeoSSE equivalence
data("geosse")

lik.g <- make.geosse(geosse.phy, geosse.phy$tip.state)
pars.g <- c(1.5, 0.5, 1.0, 0.7, 0.7, 1.4, 1.3)
names(pars.g) <- argnames(lik.g)

lik.c <- make.classe(geosse.phy, geosse.phy$tip.state+1, 3)
pars.c <- 0 * starting.point.classe(geosse.phy, 3)
pars.c['lambda222'] <- pars.c['lambda112'] <- pars.g['sA']
pars.c['lambda333'] <- pars.c['lambda113'] <- pars.g['sB']
pars.c['lambda123'] <-  pars.g['sAB']
pars.c['mu2'] <- pars.c['q13'] <- pars.g['xA']
pars.c['mu3'] <- pars.c['q12'] <- pars.g['xB']
pars.c['q21'] <- pars.g['dA']
```

```
pars.c['q31'] <- pars.g['dB']

lik.g(pars.g)   # -309.9812
lik.c(pars.c)   # -309.9812
```

`make.geosse` *Geographic State Speciation and Extinction Model*

Description

Prepare to run GeoSSE (Geographic State Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.geosse(tree, states, sampling.f=NULL, strict=TRUE,
  control=list())
starting.point.geosse(tree, eps=0.5)
```

Arguments

`tree`	A phylogenetic tree, in `ape` "phylo" format.
`states`	A vector of character states, each of which must be 0 (in both regions/widespread; AB), 1 or 2 (endemic to one region; A or B), or `NA` if the state is unknown. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`).
`sampling.f`	Vector of length 3 with the estimated proportion of extant species in states 0, 1 and 2 that are included in the phylogeny. A value of `c(0.5, 0.75, 1)` means that half of species in state 0, three quarters of species in state 1, and all the species in state 2 are included in the phylogeny. By default all species are assumed to be known.
`strict`	The `states` vector is always checked to make sure that the values are 0, 1 and 2 only. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.
`eps`	Ratio of extinction to speciation rates to be used when choosing a starting set of parameters. The procedure used is based on Magallon & Sanderson (2001).

Details

`make.geosse` returns a function of class `geosse`. The arguments and default values for this function are:

```
f(pars, condition.surv=TRUE, root=ROOT.OBS, root.p=NULL,
  intermediates=FALSE)
```

The arguments of this function are explained in make.bisse. The parameter vector `pars` is ordered `sA`, `sB`, `sAB`, `xA`, `xB`, `dA`, `dB`.

Unresolved clade methods are not available for GeoSSE.

Author(s)

Emma E. Goldberg

References

FitzJohn R.G., Maddison W.P., and Otto S.P. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Syst. Biol. 58:595-611.

Goldberg E.E., Lancaster L.T., and Ree R.H. 2011. Phylogenetic inference of reciprocal effects between geographic range evolution and diversification. Syst. Biol. 60:451-465.

Maddison W.P., Midford P.E., and Otto S.P. 2007. Estimating a binary character's effect on speciation and extinction. Syst. Biol. 56:701-710.

Magallon S. and Sanderson M.J. 2001. Absolute diversification rates in angiospem clades. Evol. 55:1762-1780.

See Also

`constrain` for making submodels, `find.mle` for ML parameter estimation, `mcmc` for MCMC integration, `make.bisse` for further relevant examples.

The help page for `find.mle` has further examples of ML searches on full and constrained BiSSE models. Things work similarly for GeoSSE, just with different parameters.

Examples

```
## Tree simulation under GeoSSE is currently only available through the
## SimTreeSDD program (\url{http://tigger.uic.edu/~eeg/code/code.html}).
## Here is an example, created with the values in "pars".

pars <- c(1.5, 0.5, 1.0, 0.7, 0.7, 2.5, 0.5)
data("geosse")

## The likelihood function
lik <- make.geosse(geosse.phy, geosse.phy$tip.state)

## See the data
statecols <- c("AB"="purple", "A"="blue", "B"="red")
plot(geosse.phy, tip.color=statecols[geosse.phy$tip.state+1], cex=0.5)

## With "true" parameter values
names(pars) <- argnames(lik)
lik(pars) # -297.6749

## A guess at a starting point.
p <- starting.point.geosse(geosse.phy)

## Start an ML search from this point (takes several minutes to run).
## Not run:
fit <- find.mle(lik, p, method="subplex")
logLik(fit) # -295.0275

## Compare with sim values.
rbind(real=pars, estimated=round(coef(fit), 2))

## A model with constraints on the dispersal rates.
lik.d <- constrain(lik, dA ~ dB)
```

```
fit.d <- find.mle(lik.d, p[-7])
logLik(fit.d) # -296.5956

## A model with constraints on the speciation rates.
lik.s <- constrain(lik, sA ~ sB, sAB ~ 0)
fit.s <- find.mle(lik.s, p[-c(2,3)])
logLik(fit.s) # -302.5542

## End(Not run)

## "Skeletal tree" sampling is supported.  For example, if your tree
## includes all AB species, half of A species, and a third of B species,
## create the likelihood function like this:
lik.f <- make.geosse(geosse.phy, geosse.phy$tip.state,
                     sampling.f=c(1, 0.5, 1/3))

## If you have external evidence that the base of your tree must have
## been in state 1, say (endemic to region A), you can fix the root
## when computing the likelihood, like this:
lik(pars, root=ROOT.GIVEN, root.p=c(0,1,0))
```

`make.geosse.split` *Geographic State Speciation and Extinction Model: Split Models*

Description

Create a likelihood function for a GeoSSE model where the tree is partitioned into regions with different parameters.

Usage

```
make.geosse.split(tree, states, nodes, split.t,
                  sampling.f=NULL, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be an integer between 0 and 2: see `make.geosse`. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). Unresolved clades are not supported.
`nodes`	Vector of nodes that will be split (see Details).
`split.t`	Vector of split times, same length as `nodes` (see Details).
`sampling.f`	Vector of length 3 where `sampling.f[i]` is the proportion of species in state `i` that are present in the phylogeny. A value of `c(0.5, 0.75, 1)` means that half of species in state 0, three quarters of species in state 1, and all species in state 2 are included in the phylogeny. By default all species are assumed to be known.
`strict`	The `states` vector is always checked to make sure that the values are integers on `0:2`. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.

Details

Branching times can be controlled with the `split.t` argument. If this is `Inf`, split at the base of the branch (as in MEDUSA). If `0`, split at the top (closest to the present, as in the new option for MEDUSA). If `0 < split.t < Inf` then we split at that time on the tree (zero is the present, with time growing backwards).

The `nodes` at the top of the split location are specified as a vector of node names. For example, a value of `c("nd10", "nd12")` means that the splits are along the branches leading from each of these nodes towards the root.

Author(s)

Emma E. Goldberg

`make.mkn` *Mk2 and Mk-n Models of character evolution*

Description

Prepare to run a Mk2/Mk-n model on a phylogenetic tree and binary/discrete trait data. This fits the Pagel 1994 model, duplicating the `ace` function in ape. Differences with that function include (1) alternative root treatments are possible, (2) easier to tweak parameter combinations through `constrain`, and (3) run both MCMC and MLE fits to parameters. Rather than exponentiate the Q matrix, this implementation solves the ODEs that this matrix defines. This may or may not be robust on trees leading to low probabilities.

Usage

```
make.mk2(tree, states, strict=TRUE, control=list())
make.mkn(tree, states, k, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in `ape` "phylo" format.
`states`	A vector of character states, each of which must be 0 or 1 for `make.mk2` or 1 to `k` for `make.mkn`.
`k`	Number of states to model.
`strict`	The `states` vector is always checked to make sure that the values are integers on `0:1` (`mk2`) or `1:k` (`mkn`). If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
`control`	List of control parameters for the ODE solver. See Details below.

Details

`make.mk2` and `make.mkn` return functions of class `mkn`. These functions have argument list (and default values)

```
f(pars, pars, prior=NULL, root=ROOT.OBS, root.p=NULL, fail.value=NULL)
```

The arguments are interpreted as

- `pars` For `make.mk2`, a vector of two parameters, in the order `q01`, `q10`. For `make.mkn`, a vector of `k(k-1)` parameters, in the order `q12,q13,...q1k, q21,q23,...,q2k,...qk(k-1)`, corresponding to the off-diagonal elements of the `Q` matrix in row order. The order of parameters can be seen by running `argnames(f)`.
- `prior`: a valid prior. See `make.prior` for more information.
- `root`: Behaviour at the root (see Maddison et al. 2007, FitzJohn et al. 2009). The possible options are
  - `ROOT.FLAT`: A flat prior, weighting all variables equally.
  - `ROOT.EQUI`: Use the equilibrium distribution of the model (not yet implemented).
  - `ROOT.OBS`: Weight D_0 and D_1 by their relative probability of observing the data, following FitzJohn et al. 2009:
  $$D = \sum_i D_i \frac{D_i}{\sum_j D_j}$$
  - `ROOT.GIVEN`: Root will be in state i with probability `root.p[i]`.
  - `ROOT.BOTH`: Don't do anything at the root, and return both values. (Note that this will not give you a likelihood for use with ML or MCMC functions!).
- `root.p`Vector of probabilities/weights to use when `ROOT.GIVEN` is specified. Must be of length `k` (2 for `make.mk2`).
- `intermediates`: Add intermediates to the returned value as attributes. Currently undocumented.

With more than 9 states, qij can be ambiguous (e.g. is q113 1->13 or 11->3?). To avoid this, the numbers are zero padded (so that the above would be q0113 or q1103 for 1->13 and 11->3 respectively). It might be easier to rename the arguments in practice though.

The `control` argument controls how the calculations will be carried out. It is a list, which may contain elements in `make.bisse`. In addition, the list element `method` may be present, which selects between three different ways of computing the likelihood:

- `method="exp"`: Uses a matrix exponentiation approach, where all transition probabilities are computed (i.e., for a rate matrix Q and time interval t, it computes $P = exp(Qt)$).
- `method="mk2"`: As for `exp`, but for 2 states only. Faster, direct, calculations are available here, rather than numerically computing the exponentiation.
- `method="ode"`: Uses an ODE-based approach to compute only the k variables over time, rather than the k^2 transition probabilities in the `exp` approach. This will be much more efficient when `k` is large. The `CVODES` backend is also available here.

Author(s)

Richard G. FitzJohn

See Also

`constrain` for making submodels, `find.mle` for ML parameter estimation, `mcmc` for MCMC integration, and `make.bisse` for state-dependent birth-death models.

Examples

```
## Simulate a tree and character distribution.  This is on a birth-death
## tree, with high rates of character evolution and an asymmetry in the
## character transition rates.
pars <- c(.1, .1, .03, .03, .1, .2)
set.seed(3)
phy <- trees(pars, "bisse", max.taxa=25, max.t=Inf, x0=0)[[1]]

## Here is the 25 species tree with the true character history coded.
## Red is state '1', which has twice the character transition rate of
## black (state '0').
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy)

## Maximum likelihood parameter estimation:
p <- c(.1, .1) # initial parameter guess

lik <- make.mk2(phy, phy$tip.state)
fit.mk2 <- find.mle(lik, p)
coef(fit.mk2)   # q10 >> q01
logLik(fit.mk2) # -10.9057

## This can also be done using the more general Mk-n.
## This uses an approximation for the likelihood calculations.  make.mkn
## assumes that states are numbered 1, 2, ..., k, so 1 needs to be added
## to the states returned by trees.
lik.mkn <- make.mkn(phy, phy$tip.state + 1, 2)
fit.mkn <- find.mle(lik.mkn, p)
fit.mkn[1:2]

## These are the same (except for the naming of arguments)
all.equal(fit.mkn[-7], fit.mk2[-7], check.attr=FALSE, tolerance=1e-7)

## Equivalence to ape's \link{ace} function:
model <- matrix(c(0, 2, 1, 0), 2)
fit.ape <- ace(phy$tip.state, phy, "discrete", model=model, ip=p)

## To do the comparison, we need to rerun the diversitree version with
## the same root conditions as ape.
fit.mk2 <- find.mle(lik, p, root=ROOT.GIVEN, root.p=c(1,1))

## These are the same to a reasonable degree of accuracy, too (the
## matrix exponentiation is slightly less accurate than the ODE
## solving approach.  The make.mk2 version is exact)
all.equal(fit.ape[c("rates", "loglik")], fit.mk2[1:2],
          check.attributes=FALSE, tolerance=1e-4)

## The ODE calculation method may be useful when there are a large
## number of possible states (say, over 20).
lik.ode <- make.mkn(phy, phy$tip.state + 1, 2,
                    control=list(method="ode"))
fit.ode <- find.mle(lik.ode, p)
fit.ode[1:2]

all.equal(fit.ode[-7], fit.mkn[-7], tolerance=1e-7)
```

make.mkn.deriv — *Mk2 and Mk-n Models of character evolution [temporary extension with derivatives]*

Description

Not currently working

Currently a restricted-feature-set version of `make.mkn`, but including gradient (first derivative) information along with the likelihood. Currently lacks several features of the full `mkn` model: no support for missing trait data, no support for different root treatments, no optimised 2-state method.

In the future, this will roll back into `make.mkn`, so do not use or depend on this!

Arguments

`tree`	A phylogenetic tree, in `ape` "phylo" format.
`states`	A vector of character states, each of which must be integers from 1 to `k`.
`k`	Number of states to model.

Author(s)

Richard G. FitzJohn

See Also

`make.mkn`, which is more general, and from which this inherits support methods for ML and MCMC inference.

make.musse — *MuSSE: Multi-State Speciation and Extinction*

Description

Prepare to run MuSSE (Multi-State Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

MuSSE is agnostic as to whether multiple states or multiple traits are modelled (following Pagel 1994). Instead, a transition rate matrix amongst possible trait/state combinations is constructed and the analysis is conducted on this.

The helper function `make.musse.multitrait` wraps the basic MuSSE model for the case of a combination of several binary traits; its argument handling are a little different; please see the help page for more information.

Usage

```
make.musse(tree, states, k, sampling.f=NULL, strict=TRUE,
           control=list())
starting.point.musse(tree, k, q.div=5, yule=FALSE)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be an integer between 1 and `k`. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.
`k`	The number of states.
`sampling.f`	Vector of length `k` where `sampling.f[i]` is the proportion of species in state `i` that are present in the phylogeny. A value of `c(0.5, 0.75, 1)` means that half of species in state 1, three quarters of species in state 2, and all species in state 3 are included in the phylogeny. By default all species are assumed to be known
`strict`	The `states` vector is always checked to make sure that the values are integers on `1:k`. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.
`q.div`	Ratio of diversification rate to character change rate. Eventually this will be changed to allow for Mk2 to be used for estimating q parameters.
`yule`	Logical: should starting parameters be Yule estimates rather than birth-death estimates?

Details

With more than 9 states, qij can be ambiguous (e.g. is q113 1->13 or 11->3?). To avoid this, the numbers are zero padded (so that the above would be q0113 or q1103 for 1->13 and 11->3 respectively). It might be easier to rename the arguments in practice though.

Author(s)

Richard G. FitzJohn

See Also

`make.bisse` for the basic binary model, and `make.musse.multitrait` for the case where the data are really combinations of binary traits.

Examples

```
## 1: BiSSE equivalence
pars <- c(.1, .2, .03, .04, 0.05, 0.1)
set.seed(2)
phy <- tree.musse(pars, 20, x0=1)

## Show that the likelihood functions give the same answers.  Ignore the
## warning when creating the MuSSE function.
lik.b <- make.bisse(phy, phy$tip.state-1)
lik.m <- make.musse(phy, phy$tip.state, 2)
all.equal(lik.b(pars), lik.m(pars), tolerance=1e-7)

## Notice that default argument names are different between BiSSE and
## MuSSE, but that the order is the same.
```

```
argnames(lik.b) # BiSSE: 0/1
argnames(lik.m) # MuSSE: 1/2

## 2: A 3-state example where movement is only allowed between
## neighbouring states (1 <-> 2 <-> 3), and where speciation and
## extinction rates increase moving from 1 -> 2 -> 3:

## You can get the expected argument order for any number of states
## this way (sorry - clunky).  The help file also lists the order.
diversitree:::default.argnames.musse(3)

## Here are the parameters:
pars <- c(.1,  .15,  .2,  # lambda 1, 2, 3
          .03, .045, .06, # mu 1, 2, 3
          .05, 0,         # q12, q13
          .05, .05,       # q21, q23
          0,   .05)       # q31, q32

set.seed(2)
phy <- tree.musse(pars, 30, x0=1)

## Extract history from simulated tree and plot
## (colours are 1: black, 2: red, 3: blue)
col <- c("blue", "orange", "red")
h <- history.from.sim.discrete(phy, 1:3)
plot(h, phy, cex=.7, col=col)

## The states are numbered 1:3, rather than 0:1 in bisse.
states <- phy$tip.state
table(states)

## 2: Likelihood
## Making a likelihood function is basically identical to bisse.  The
## third argument needs to be the number of states.  In a future
## version this will probably be max(states), but there are some
## pitfalls about this that I am still worried about.
lik <- make.musse(phy, states, 3)

## Here are the arguments.  Even with three states, this is getting
## ridiculous.
argnames(lik)

## Start with a fully constrained model, but still enforcing stepwise
## changes (disallowing 1 <-> 3 shifts)
lik.base <- constrain(lik, lambda2 ~ lambda1, lambda3 ~ lambda1,
                      mu2 ~ mu1, mu3 ~ mu1,
                      q13 ~ 0, q21 ~ q12, q23 ~ q12, q31 ~ 0, q32 ~ q12)

p <- starting.point.musse(phy, 3)
fit.base <- find.mle(lik.base, p[argnames(lik.base)])

## Now, allow the speciation rates to vary:
lik.lambda <- constrain(lik, mu2 ~ mu1, mu3 ~ mu1,
                        q13 ~ 0, q21 ~ q12, q23 ~ q12, q31 ~ 0, q32 ~ q12)
fit.lambda <- find.mle(lik.lambda, p[argnames(lik.lambda)])

## Very little improvement in fit (this *is* a small tree)
```

```
anova(fit.base, lambda=fit.lambda)

## Run an MCMC with this set.  Priors will be necessary (using the
## usual exponential with mean of 2r)
prior <- make.prior.exponential(1 / (2 * (p[1] - p[4])))
## Not run:
samples <- mcmc(lik.lambda, coef(fit.lambda), nstep=1000, w=1,
                prior=prior, print.every=50)

## Posterior probability profile plots for the three speciation rates.
profiles.plot(samples[2:4], col)
abline(v=c(.1, .15, .2), col=col)

## End(Not run)
```

`make.musse.multitrait` *MuSSE: Multi-State Speciation and Extinction (Multiple Binary Traits Version)*

Description

Prepare to run MuSSE or Mkn (Multi-State Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

This is a helper function that wraps the basic MuSSE/Mkn models for the case of a combination of several binary traits; its parametrisation and argument handling are a little different to the other models in diversitree.

Usage

```
make.musse.multitrait(tree, states, sampling.f=NULL,
                      depth=NULL, allow.multistep=FALSE,
                      strict=TRUE, control=list())
make.mkn.multitrait(tree, states,
                    depth=NULL, allow.multistep=FALSE,
                    strict=TRUE, control=list())

musse.multitrait.translate(n.trait, depth=NULL, names=NULL,
                           allow.multistep=FALSE)
mkn.multitrait.translate(n.trait, depth=NULL, names=NULL,
                         allow.multistep=FALSE)

starting.point.musse.multitrait(tree, lik, q.div=5, yule=FALSE)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A `data.frame` of character states, each column of which represents a different binary state (with values 0 or 1), and each row of which represents a taxon. The row names of `states` must be the names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). The column names must be unique and a single character long. The character "0" (zero) is reserved and may not be

used. NA values are allowed in one or more columns when one or more traits is unknown for a taxon.

depth
: A scalar or vector of length 3 indicating the depth of interactions to include in the model. See Details.

allow.multistep
: Should transition rates be included that imply simultaneous changes in more than one trait? By default this is not allowed, but if set to TRUE these rates are included at the end of the parameter vector. Warning: treatment of these will change in future versions!

sampling.f
: Scalar with the estimated proportion of extant species that are included in the phylogeny. A value of 0.75 means that three quarters of extant species are included in the phylogeny. By default all species are assumed to be known. In the future, this will expand to allow state-specific sampling rates.

strict
: Each column in states is always checked to make sure that the values are 0 or 1. If strict is TRUE (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful. Note that this model may misbehave even if this check is met, due to combinations of traits being absent.

control
: List of control parameters for the ODE solver. See details in make.bisse.

lik
: A likelihood function created by make.musse.multitrait.

q.div
: Ratio of diversification rate to character change rate. Eventually this will be changed to allow for Mk2 to be used for estimating q parameters.

yule
: Logical: should starting parameters be Yule estimates rather than birth-death estimates?

n.trait
: Number of binary traits.

names
: Vector of names for the traits when using musse.multitrait.translate (optional).

Details

Suppose that you have two binary traits that may affect speciation and extinction. In previous versions of diversitree, you had to code the possible combinations as states 1, 2, 3, 4, which makes the interpretation of the speciation rates (lambda1, lambda2, etc) unintuitive.

Let states is a data.frame with columns "A" and "B", representing the two binary traits. We can write the speciation rate as

$$\lambda_0 + \lambda_A X_A + \lambda_B X_B + \lambda_{AB} X_A X_B$$

where X_A and X_B are indicator variables that take the value of trait A and B respectively (with values 0 or 1). In this form, λ_0 is the intercept, λ_A and λ_B are "main effects" of traits A and B, and λ_{AB} is the "interaction" between these. We can do a similar trick for the extinction rates.

For character transition rates, we first consider changes only in a single trait. For our two trait case we have four "types" of character change allowed (A 0->1, A 1->0, B 0->1, and B 1->0), but the rates of change for trait A might depend on the current state of trait B (and vice versa). So we have, for the A0->1 trait change $q_{A01,0} + q_{A01,B} \times X_B$. Note that one fewer levels of interaction are possible for these character changes than for the speciation/extinction parameters.

It may sometimes be desirable to have the multi-trait changes in the model. At present, if allow.multistep is TRUE, all the multiple change transitions are included at the end of the parameter vector. For the

two trait case these are labelled q00.11, q10.01, q01.10, and q11.00, where qij.kl represents a change from (A=i, B=j) to (C=k, D=l). The argument name, and treatment, of these may change in future.

This approach generalises out to more than two traits. For N traits, interactions are possible up to the Nth order for lambda and mu, and up to the N-1th order for q. The depth argument controls how many of these are returned. If this is a scalar, then the same level is used for lambda, mu and q. If it is a vector of length 3, then different depths are used for these three types of parameters. By default, all possible interactions are returned and the model has the same number of degrees of freedom as the models returned by make.musse (except for a reduction in the possible q parameters when allow.multistep is FALSE). Parameters can then be further refined with constrain.

Author(s)

Richard G. FitzJohn

See Also

make.bisse for the basic binary model, and make.musse for the basic multistate model.

Examples

```
## The translation between these two bases is fairly straightforward; if
## we have a vector of parameters in our new basis 'p' we can convert it
## into the original MuSSE basis ('q') through this matrix:
tr <- musse.multitrait.translate(2)
tr

## Notice that the rows that correspond to transitions in multiple
## traits are all zero by default; this means that these q values will
## be zero regardless of the parameter vector used.
tr["q00.11",]

## And here is the section of the transition matrix corresponding to the
## lambda values; every rate gets a contribution from the intercept term
## (lambda0), lambda10 and lambda11 get a contribution from lambdaA, etc.
tr[1:4,1:4]

## There is currently no nice simulation support for this, so bear with
## an ugly script to generate the tree and traits.
pars <- c(.10, .15, .20, .25, # lambda 00, 10, 01, 11
          .03, .03, .03, .03, # mu 00, 10, 01, 11
          .05, .05, .0,       # q00.10, q00.01, q00.11
          .05, .0,  .05,      # q10.00, q10.01, q10.11
          .05, .0,  .05,      # q01.00, q01.10, q01.11
          .0,  .05, .05)      # q11.00, q11.10, q11.01
set.seed(2)
phy <- tree.musse(pars, 60, x0=1)

states <- expand.grid(A=0:1, B=0:1)[phy$tip.state,]
rownames(states) <- phy$tip.label

## Here, states has row names corresponding to the different taxa, and
## the states of two traits "A" and "B" are recorded in the columns.
head(states)

## Note that transition from the original MuSSE basis to this basis is
```

```
## only possible in general when depth=n.trait and allow.multistep=TRUE
## (as only this generates a square matrix that is invertible).
## However, when it is possible to express the set of parameters in the
## new basis (as it is above), this can be done through a pseudoinverse
## (here, a left inverse).
pars2 <- drop(solve(t(tr) %*% tr) %*% t(tr) %*% pars)

## Going from our new basis to the original MuSSE parameters is always
## straightforward.  This is done automatically in the likelihood
## function.
all.equal(tr %*% pars2, pars, check.attributes=FALSE)

## This shows that the two traits act additively on speciation rate
## (lambdaAB is zero), that there is no effect of any trait on
## extinction (the only nonzero mu parameter is mu0) and transition
## rates for one trait are unaffected by other traits (the only nonzero
## q parameters are the qXij.0 parameters; qXij.Y parameters are all
## zero).

## Here is our new MuSSE function parametrised as a multi-trait
## function:
lik <- make.musse.multitrait(phy, states)

## Here are the argument names for the likelihood function.
argnames(lik)

## Basic MuSSE function for comparison
lik.m <- make.musse(phy, phy$tip.state, 4)
argnames(lik.m)

## Rather than fit this complicated model first, let's start with a
## simple model with no state dependent diversification.  This model
## allows the forwards and backwards transition rates to vary, but the
## speciation and extinction rates do not depend on the character
## state:
lik0 <- make.musse.multitrait(phy, states, depth=0)
argnames(lik0)

p <- starting.point.musse.multitrait(phy, lik0)
fit0 <- find.mle(lik0, p)

## Now, allow the speciation rates to vary additively with both
## character states (extinction and character changes are left as in the
## previous model)
lik1 <- make.musse.multitrait(phy, states, depth=c(1, 0, 0))

## Start from the previous ML point:
p <- starting.point.musse.multitrait(phy, lik1)
p[names(coef(fit0))] <- coef(fit0)

fit1 <- find.mle(lik1, p)

## The likelihood improves, but the difference is not statistically
## significant (p = 0.35).
anova(fit1, fit0)

## We can fit an interaction for the speciation rates, too:
```

```
lik2 <- make.musse.multitrait(phy, states, depth=c(2, 0, 0))
p <- starting.point.musse.multitrait(phy, lik2)
p[names(coef(fit1))] <- coef(fit1)
fit2 <- find.mle(lik2, p)

## There is next to no support for the interaction term (which is good,
## as the original model did not have any interaction!)
anova(fit2, fit1)

## Constraining also works with these models.  For example, constraining
## the lambdaA parameter to zero:
lik1b <- constrain(lik1, lambdaA ~ 0)
argnames(lik1b)
p <- starting.point.musse.multitrait(phy, lik1b)
p[names(coef(fit0))] <- coef(fit0)
fit1b <- find.mle(lik1b, p)
anova(fit1b, fit0)

## Or constraining both main effects to take the same value:
lik1c <- constrain(lik1, lambdaB ~ lambdaA)
argnames(lik1c)
p <- starting.point.musse.multitrait(phy, lik1c)
p[names(coef(fit0))] <- coef(fit0)
fit1c <- find.mle(lik1c, p)
anova(fit1c, fit0)
```

`make.musse.split` *Multiple State Speciation and Extinction Model: Split Models*

Description

Create a likelihood function for a MuSSE model where the tree is partitioned into regions with different parameters.

Usage

```
make.musse.split(tree, states, k, nodes, split.t,
                 sampling.f=NULL, strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in `ape` "phylo" format.
`states`	A vector of character states, each of which must be an integer between 1 and `k`. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.
`k`	The number of states.
`nodes`	Vector of nodes that will be split (see Details).
`split.t`	Vector of split times, same length as `nodes` (see Details).
`sampling.f`	Vector of length `k` where `sampling.f[i]` is the proportion of species in state `i` that are present in the phylogeny. A value of `c(0.5, 0.75, 1)` means that half of species in state 1, three quarters of species in state 2, and all species in state 3 are included in the phylogeny. By default all species are assumed to be known

`strict`	The `states` vector is always checked to make sure that the values are integers on `1:k`. If `strict` is `TRUE` (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
`control`	List of control parameters for the ODE solver. See details in `make.bisse`.

Details

Branching times can be controlled with the `split.t` argument. If this is `Inf`, split at the base of the branch (as in MEDUSA). If `0`, split at the top (closest to the present, as in the new option for MEDUSA). If `0 < split.t < Inf` then we split at that time on the tree (zero is the present, with time growing backwards).

Author(s)

Richard G. FitzJohn

Examples

```
## This example picks up from the tree used in the ?make.musse example.

## First, simulate the tree:
set.seed(2)
pars <- c(.1,  .15,  .2,  # lambda 1, 2, 3
          .03, .045, .06, # mu 1, 2, 3
          .05, 0,         # q12, q13
          .05, .05,       # q21, q23
          0,   .05)       # q31, q32
phy <- tree.musse(pars, 30, x0=1)

## Here is the phylogeny, with true character history superposed:
h <- history.from.sim.discrete(phy, 1:3)
plot(h, phy, show.node.label=TRUE, font=1, cex=.75, no.margin=TRUE)

## Here is a plain MuSSE function for later comparison:
lik.m <- make.musse(phy, phy$tip.state, 3)
lik.m(pars) # -110.8364

## Split this phylogeny at three points: nd16 and nd25, splitting it
## into three chunks
nodes <- c("nd16", "nd25")
nodelabels(node=match(nodes, phy$node.label) + length(phy$tip.label),
           pch=19, cex=2, col="#FF000099")

## To make a split BiSSE function, pass the node locations and times
## in.  Here, we'll use 'Inf' as the split time to mimick MEDUSA's
## behaviour of placing the split at the base of the branch subtended by
## a node.
lik.s <- make.musse.split(phy, phy$tip.state, 3, nodes, split.t=Inf)

## The parameters must be a list of the same length as the number of
## partitions.  Partition '1' is the root partition, and partition i is
## the partition rooted at the node[i-1]:
argnames(lik.s)
```

```
## Because we have two nodes, there are three sets of parameters.
## Replicate the original list to get a starting point for the analysis:
pars.s <- rep(pars, 3)
names(pars.s) <- argnames(lik.s)

lik.s(pars.s) # -110.8364

## This is basically identical (to acceptable tolerance) to the plain
## MuSSE version:
lik.s(pars.s) - lik.m(pars)

## The resulting likelihood function can be used in ML analyses with
## find.mle.  However, because of the large number of parameters, this
## may take some time (especially with as few species as there are in
## this tree - getting convergence in a reasonable number of iterations
## is difficult).
## Not run:
fit.s <- find.mle(lik.s, pars.s, control=list(maxit=20000))

## End(Not run)

## Bayesian analysis also works, using the mcmc function.  Given the
## large number of parameters, priors will be essential, as there will
## be no signal for several parameters.  Here, I am using an exponential
## distribution with a mean of twice the state-independent
## diversification rate.
## Not run:
prior <- make.prior.exponential(1/(-2*diff(starting.point.bd(phy))))
samples <- mcmc(lik.s, pars.s, 100, prior=prior, w=1, print.every=10)

## End(Not run)
```

`make.musse.td` *Multiple State Speciation and Extinction Model: Time Dependent Models*

Description

Create a likelihood function for a MuSSE model where different chunks of time have different parameters. This code is experimental!

Usage

```
make.musse.td(tree, states, k, n.epoch, sampling.f=NULL,
              strict=TRUE, control=list())
make.musse.t(tree, states, k, functions, sampling.f=NULL,
              strict=TRUE, control=list())
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be an integer between 1 and `k`. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`). For tips corresponding to unresolved clades, the state should be `NA`.

k	The number of states.
n.epoch	Number of epochs. 1 corresponds to plain MuSSE, so this will generally be an integer at least 2.
functions	A named list of functions of time. See details.
sampling.f	Vector of length k where sampling.f[i] is the proportion of species in state i that are present in the phylogeny. A value of c(0.5, 0.75, 1) means that half of species in state 1, three quarters of species in state 2, and all species in state 3 are included in the phylogeny. By default all species are assumed to be known
strict	The states vector is always checked to make sure that the values are integers on 1:k. If strict is TRUE (the default), then the additional check is made that *every* state is present. The likelihood models tend to be poorly behaved where states are missing, but there are cases (missing intermediate states for meristic characters) where allowing such models may be useful.
control	List of control parameters for the ODE solver. See details in make.bisse.

Details

Please see make.bisse for further details.

Author(s)

Richard G. FitzJohn

Examples

```
## Here we will start with the tree and three-state character set from
## the make.musse example.  This is a poorly contrived example.
pars <- c(.1,  .15,  .2,  # lambda 1, 2, 3
          .03, .045, .06, # mu 1, 2, 3
          .05, 0,         # q12, q13
          .05, .05,       # q21, q23
          0,   .05)       # q31, q32
set.seed(2)
phy <- tree.musse(pars, 30, x0=1)

## Suppose we want to see if diversification is different in the most
## recent 3 time units, compared with the rest of the tree (yes, this is
## a totally contrived example!):
plot(phy)
axisPhylo()
abline(v=max(branching.times(phy)) - 3, col="red", lty=3)

## For comparison, make a plain MuSSE likelihood function
lik.m <- make.musse(phy, phy$tip.state, 3)

## Create the time-dependent likelihood function.  The final argument
## here is the number of 'epochs' that are allowed.  Two epochs is one
## switch point.
lik.t <- make.musse.td(phy, phy$tip.state, 3, 2)

## The switch point is the first argument.  The remaining 12 parameters
## are the MuSSE parameters, with the first 6 being the most recent
## epoch.
argnames(lik.t)
```

```
pars.t <- c(3, pars, pars)
names(pars.t) <- argnames(lik.t)

## Calculations are identical to a reasonable tolerance:
lik.m(pars) - lik.t(pars.t)

## It will often be useful to constrain the time as a fixed quantity.
lik.t2 <- constrain(lik.t, t.1 ~ 3)

## Parameter estimation under maximum likelihood.  This is marked "don't
## run" because the time-dependent fit takes a few minutes.
## Not run:
## Fit the MuSSE ML model
fit.m <- find.mle(lik.m, pars)

## And fit the MuSSE/td model
fit.t <- find.mle(lik.t2, pars.t[argnames(lik.t2)],
                  control=list(maxit=20000))

## Compare these two fits with a likelihood ratio test (lik.t2 is nested
## within lik.m)
anova(fit.m, td=fit.t)

## End(Not run)
```

`make.prior` *Simple Prior Functions*

Description

Functions for generating prior functions for use with `mcmc`, etc.

Usage

```
make.prior.exponential(r)
make.prior.uniform(lower, upper)
```

Arguments

`r`	Scalar or vector of rate parameters
`lower`	Lower bound of the parameter
`upper`	Upper bound of the parameter

Details

The exponential prior probability distribution has probability density

$$\sum_i r_i e^{-r_i x_i}$$

where the i denotes the ith parameter. If `r` is a scalar, then the same rate is used for all parameters.

These functions each return a function that may be used as the `prior` argument to `mcmc()`.

Author(s)

Richard G. FitzJohn

`make.quasse` *Quantitative State Speciation and Extinction Model*

Description

Prepare to run QuaSSE (Quantitative State Speciation and Extinction) on a phylogenetic tree and character distribution. This function creates a likelihood function that can be used in maximum likelihood or Bayesian inference.

Usage

```
make.quasse(tree, states, states.sd, lambda, mu, control,
            sampling.f=NULL)
starting.point.quasse(tree, states, states.sd=NULL)
```

Arguments

`tree`	A phylogenetic tree, in `ape` "phylo" format.
`states`	A vector of character states, each of which must be a numeric real values. Missing values (`NA`) are not yet handled. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`).
`states.sd`	A scalar or vector corresponding to the standard error around the mean in states (the initial probability distribution is assumed to be normal).
`lambda`	A function to use as the speciation function. The first argument of this must be `x` (see Details).
`mu`	A function to use as the extinction function. The first argument of this must be `x` (see Details.)
`control`	A list of parameters for tuning the performance of the integrator. A guess at reasonble values will be made here. See Details for possible entries.
`sampling.f`	Scalar with the estimated proportion of extant species that are included in the phylogeny. A value of `0.75` means that three quarters of extant species are included in the phylogeny. By default all species are assumed to be known.

Details

The `control` list may contain the following elements:

- `method`: one of `fftC` or `fftR` to switch between `C` (fast) and `R` (slow) backends for the integration. Both use non-adaptive fft-based convolutions. Eventually, an adaptive methods-of-lines approach will be available.
- `dt.max`: Maximum time step to use for the integration. By default, this will be set to 1/1000 of the tree depth. Smaller values will slow down calculations, but improve accuracy.
- `nx`: The number of bins into which the character space is divided (default=1024). Larger values will be slower and more accurate. For the `fftC` integration method, this should be an integer power of 2 (512, 2048, etc).

- r: Scaling factor that multiplies nx for a "high resolution" section at the tips of the tree (default=4, giving a high resolution character space divided into 4096 bins). This helps improve accuracy while possibly tight initial probability distributions flatten out as time progresses towards the root. Larger values will be slower and more accurate. For the fftC integration method, this should be a power of 2 (2, 4, 8, so that nx*r is a power of 2).
- tc: where in the tree to switch to the low-resolution integration (zero corresponds to the present, larger numbers moving towards the root). By default, this happens at 10% of the tree depth. Smaller values will be faster, but less accurate.
- xmid: Mid point to center the character space. By default this is at the mid point of the extremes of the character states.
- tips.combined: Get a modest speed-up by simultaneously integrating all tips? By default, this is FALSE, but speedups of up to 25% are possible with this set to TRUE.
- w: Number of standard deviations of the normal distribution induced by Brownian motion to use when doing the convolutions (default=5). Probably best to leave this one alone.

Warning

In an attempt at being computationally efficient, a substantial amount of information is cached in memory so that it does not have to be created each time. However, this can interact poorly with the multicore package. In particular, likelihood functions should not be made within a call to mclapply, or they will not share memory with the main R thread, and will not work (this will cause an error, but should no longer crash R).

The method has less general testing than BiSSE, and is a little more fragile. In particular, because of the way that I chose to implement the integrator, there is a very real chance of likelihood calculation failure when your data are a poor fit to the model; this can be annoyingly difficult to diagnose (you will just get a -Inf log likelihood, but the problem is often just caused by two sister species on short branches with quite different states). There are also a large number of options for fine tuning the integration, but these aren't really discussed in any great detail anywhere.

Author(s)

Richard G. FitzJohn

Examples

```
## Example showing simple integration with two different backends,
## plus the splits.
lambda <- function(x) sigmoid.x(x, 0.1, 0.2,  0, 2.5)
mu <- function(x) constant.x(x, 0.03)
char <- make.brownian.with.drift(0, 0.025)

set.seed(1)
phy <- tree.quasse(c(lambda, mu, char), max.taxa=15, x0=0,
                   single.lineage=FALSE, verbose=TRUE)

nodes <- c("nd13", "nd9", "nd5")
split.t <- Inf

pars <- c(.1, .2, 0, 2.5, .03, 0, .01)
pars4 <- unlist(rep(list(pars), 4))

sd <- 1/200
control.C.1 <- list(dt.max=1/200)
```

```
control.R.1 <- list(dt.max=1/200, method="fftR")
lik.C.1 <- make.quasse(phy, phy$tip.state, sd, sigmoid.x, constant.x, control.C.1)
(ll.C.1 <- lik.C.1(pars)) # -62.06409




## Not run:
## slow...
lik.R.1 <- make.quasse(phy, phy$tip.state, sd, sigmoid.x, constant.x, control.R.1)
(ll.R.1 <- lik.R.1(pars)) # -62.06409

## End(Not run)

lik.s.C.1 <- make.quasse.split(phy, phy$tip.state, sd, sigmoid.x, constant.x,
                               nodes, split.t, control.C.1)
(ll.s.C.1 <- lik.s.C.1(pars4)) # -62.06409
```

`make.quasse.split` *Quantitative State Speciation and Extinction Model: Split Models*

Description

Create a likelihood function for a QuaSSE model where the tree is partitioned into regions with different parameters.

Usage

```
make.quasse.split(tree, states, states.sd, lambda, mu, nodes, split.t,
                  control=NULL, sampling.f=NULL)
```

Arguments

`tree`	A phylogenetic tree, in ape "phylo" format.
`states`	A vector of character states, each of which must be a numeric real values. Missing values (`NA`) are not yet handled. This vector must have names that correspond to the tip labels in the phylogenetic tree (`tree$tip.label`).
`states.sd`	A scalar or vector corresponding to the standard error around the mean in states (the initial probability distribution is assumed to be normal).
`lambda`	A function to use as the speciation function. The first argument of this must be `x` (see Details).
`mu`	A function to use as the extinction function. The first argument of this must be `x` (see Details.)
`nodes`	Vector of nodes that will be split (see Details).
`split.t`	Vector of split times, same length as `nodes` (see Details).
`control`	A list of parameters for tuning the performance of the integrator. A guess at reasonble values will be made here. See Details in `make.quasse` for possible entries.

`sampling.f` Scalar with the estimated proportion of extant species that are included in the phylogeny. A value of `0.75` means that three quarters of extant species are included in the phylogeny. By default all species are assumed to be known.

Details

Branching times can be controlled with the `split.t` argument. If this is `Inf`, split at the base of the branch (as in MEDUSA). If `0`, split at the top (closest to the present, as in the new option for MEDUSA). If `0 < split.t < Inf` then we split at that time on the tree (zero is the present, with time growing backwards).

TODO: Describe `nodes` and `split.t` here.

Author(s)

Richard G. FitzJohn

`mcmc` *Simple Markov Chain Monte Carlo with Slice Sampling*

Description

Run a simple-minded MCMC using slice samples (Neal 2003) for independent updating of each variable.

Usage

```
mcmc(lik, x.init, nsteps, ...)
## Default S3 method:
mcmc(lik, x.init, nsteps, w, prior=NULL,
             sampler=sampler.slice, fail.value=-Inf, lower=-Inf,
             upper=Inf, print.every=1, control=list(),
             save.every=0, save.file, ...)

sampler.slice(lik, x.init, y.init, w, lower, upper, control)
sampler.norm(lik, x.init, y.init, w, lower, upper, control)
```

Arguments

`lik` Likelihood function to run MCMC on. This must return the log likelihood (or the log of a value proportional to the likelihood).

`x.init` Initial parameter location (vector).

`nsteps` Number of MCMC steps to take.

`w` Tuning parameter for the sampler. See Details below for more information.

`prior` An optional prior probability distribution function. This must be a function that returns the log prior probability, given a parameter vector. See `make.prior` for more information.

`sampler` Sampler to use for the MCMC. There are currently only two implemented; `sampler.slice` (the default, and generally recommended), and `sampler.norm` (Gaussian updates, and for illustrative purposes mostly).

`lower`	Lower bounds on parameter space (scalar or vector of same length as `x.init`).
`upper`	Upper bounds on parameter space (scalar or vector of same length as `x.init`).
`fail.value`	Value to use where function evaluation fails. The default (negative infinity) corresponds to zero probability. Most values that fail are invalid for a given model (negative rates, etc) or have negligble probability, so this is reasonable. Set to `NULL` to turn off checking.
`print.every`	The position and its probability will be printed every `print.every` generations. Set this to 0 to disable printing.
`control`	List with additional control parameters for the sampler. Not currently used.
`save.every`	Number of steps to save progress into `save.file`. By default this is 0, which prevents saving occuring. Low nonzero values of this will slow things down, but may be useful during long runs.
`save.file`	Name of csv file to save temporary output in. Contents will be rewritten at each save.
`...`	Arguments passed to the function `lik`
`y.init`	Likelihood evaluated at `x.init`.

Details

There are two samplers implemented: a slice sampler (Neal 2003) and a basic Gaussian sampler. In general, only the slice sampler should be used; the Gaussian sampler is provided for illustration and as a starting point for future samplers.

For slice sampling (`sampler.slice`), the tuning parameter `w` affects how many function evaluations are required between sample updates, but in almost all cases **it does not affect how fast the MCMC "mixes"** (Neal 2003). In particular, `w` is not analagous to the step sizes used in conventional Metropolis-Hastings updaters that use some fixed kernel for updates (see below). Ideally, `w` would be set to approximately the width of the high probability region. I find that chosing the distance between the 5% and 95% quantiles of the marginal distributions of each parameter works well, computed from this preliminary set of samples (see Examples). If a single value is given, this is shared across all parameters.

For the Gaussian updates (`sampler.norm`), the tuning parameter `w` is the standard deviation of the normal distribution centred on each parameter as it is updated.

For both samplers, if a single value is given, this is shared across all parameters. If a vector is given, then it must be the same length as `w`, and parameter `i` will use `w[i]`.

If the MCMC is stopped by an interrupt (Escape on GUI versions of R, Control-C on command-line version), it will return a truncated chain with as many points as completed so far.

This is far from the most efficient MCMC function possible, as it was designed to work with likelihood functions that are relatively expensive to compute. The overhead for 10,000 slice samples is on the order of 5s on a 2008 Mac Pro (0.0005 s / sample).

The sampler function `sampler.norm` and `sampler.slice` should not generally be called directly (though this is possible), but exist only to be passed in to `mcmc`.

Author(s)

Richard G. FitzJohn

References

Neal R.M. 2003. Slice sampling. Annals of Statistics 31:705-767.

See Also

make.bd, make.bisse, make.geosse, and make.mkn, all of which provide likelihood functions that are suitable for use with this function. The help page for make.bd has further examples of using MCMC, and make.bisse has examples of using priors with MCMC.

Examples

```
## To demonstrate, start with a simple bivariate normal.  The function
## 'make.mvn' creates likelihood function for the multivariate normal
## distribution given 'mean' (a vector) and 'vcv' (the variance
## covariance matrix).  This is based on mvnorm in the package
## mvtnorm, but will be faster where the vcv does not change between
## calls.
make.mvn <- function(mean, vcv) {
  logdet <- as.numeric(determinant(vcv, TRUE)$modulus)
  tmp <- length(mean) * log(2 * pi) + logdet
  vcv.i <- solve(vcv)

  function(x) {
    dx <- x - mean
    -(tmp + rowSums((dx %*% vcv.i) * dx))/2
  }
}

## Our target distribution has mean 0, and a VCV with positive
## covariance between the two parameters.
vcv <- matrix(c(1, .25, .25, .75), 2, 2)
lik <- make.mvn(c(0, 0), vcv)

## Sample 10,000 points from the distribution, starting at c(0, 0).
set.seed(1)
samples <- mcmc(lik, c(0, 0), 10000, 1, print.every=1000)

## The marginal distribution of V1 (the first axis of the
## distribution) should be a normal distribution with mean 0 and
## variance 1:
curve(dnorm, xlim=range(samples$V1), ylim=c(0, .5), col="red")
hist(samples$V1, 30, add=TRUE, freq=FALSE)

plot(V2 ~ V1, samples, pch=19, cex=.2, col="#00000055", asp=1)

## The estimated variance here matches nicely with the true VCV:
var(samples[2:3])

## The above uses slice sampling.  We can use simple Gaussian updates
## instead.  This performs updates with standard deviation '1' in each
## direction.  Unlike slice sampling, the 'w' parameter here will
## matter a great deal in determining how fast the chain will mix.
samples.norm <- mcmc(lik, c(0, 0), 10000, 1, print.every=1000,
                     sampler=sampler.norm)

## This *appears* to run much faster than the slice sampling based
## approach above, but the effective sample size of the second
## approach is much lower.  The 'effectiveSize' function in coda says
## that for 10,000 samples using slice sampling, the effective sample
## size (equivalent number of independent samples) is about 8,500, but
```

```
## for the Gaussian updates is only 1,200.  This can be seen by
## comparing the autocorrelation between samples from the two
## different runs.
op <- par(oma=c(0, 0, 2, 0))
acf(samples[2:3])
title(main="Slice sampling", outer=TRUE)

acf(samples.norm[2:3])
title(main="Gaussian updates", outer=TRUE)

## The autocorrelation is negligable after just 2 samples under slice
## sampling, but remains significant for about 15 with Gaussian
## updates.

## Not run:
## Next, a diversitree likelihood example.  This example uses a 203
## species phylogeny evolved under the BiSSE model.  This takes a
## more substantial amount of time, so is not evaluated by default.
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)

## First, create a likelihood function:
lik <- make.bisse(phy, phy$tip.state)
lik(pars)

## This produces about a sample a second, so takes a while.  The "upper"
## limit is a hard upper limit, above which the sampler will never let
## the parameter going (in effect, putting a uniform prior on the range
## lower..upper, and returning the joint distribution conditional on the
## parameters being in this range).
tmp <- mcmc(lik, pars, nsteps=100, w=.1)

## The argument 'w' works best when it is about the width of the "high
## probability" region for that parameter.  This takes the with of the
## 90
## than the first guess.  Samples are generated about 1/s; allow 15
## minutes to generate 1000 samples.
w <- diff(sapply(tmp[2:7], quantile, c(.05, .95)))
out <- mcmc(lik, pars, nsteps=1000, w=w)

## You can do several things with this.  Look for candidate ML points:
out[which.max(out$p),]

## Or look at the marginal distribution of parameters
profiles.plot(out["lambda0"], col.line="red")

## Or look at the joint marginal distribution of pairs of parameters
plot(lambda0 ~ mu0, out)

## End(Not run)
```

`plot.history` *Plot Character History*

Description

Both stochastic character mapping and simulation may create character histories. This function plots these histories

Usage

```
## S3 method for class 'history'
plot(x, phy, cols=seq_along(states),
             states=x$states,
             xlim=NULL, ylim=NULL, show.tip.label=TRUE,
             show.node.label=FALSE, show.tip.state=TRUE,
             show.node.state=TRUE, no.margin=FALSE, cex=1, font=3,
             srt=0, adj=0, label.offset=NA, lwd=1, ...)
```

Arguments

`x`	An object of class `history.discrete` containing a discrete character history. This could be made by `history.from.sim.discrete`.
`phy`	The phylogeny used to generate the history. Few checks are made to make sure that this is really correct, and all manner of terrible things might happen if these are not compatible. This may change in future.
`cols`	A vector of colours.
`states`	The different state types. Probably best to leave alone.
`xlim`	Plot x-limits (optional).
`ylim`	Plot y-limits (optional).
`show.tip.label`	Logical: show the species tip labels?
`show.node.label`	Logical: show the species node labels?
`show.tip.state`	Logical: draw a symbol at the tips to indicate tip state?
`show.node.state`	Logical: draw a symbol at the nodes to indicate node state?
`no.margin`	Supress drawing of margins around the plot
`cex`	Font and symbol scaling factor.
`font`	Font used for tip and node labels (see `par`).
`srt`	String rotation for tip and node labels.
`adj`	Label adjustment (see `par`).
`label.offset`	Horizontal offset of tip and node labels, in branch length units.
`lwd`	Line width
`...`	Additional arguments (currently ignored)

Details

This attempts to be as compatible with ape's plotting functions as possible, but currently implements only right-facing cladegrams.

Author(s)

Richard G. FitzJohn

Examples

```
## Simulate a tree, but retain extinct species.
pars <- c(.1, .2, .03, .04, 0.05, 0.1) # BiSSE pars
set.seed(2)
phy <- tree.bisse(pars, 20, x0=0, include.extinct=TRUE)

## Create a 'history' from the information produced by the simulation
## and plot this
h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy, cex=.7)

## Prune the extinct taxa.
phy2 <- prune(phy)

## The history must be recreated for this pruned tree:
h2 <- history.from.sim.discrete(phy2, 0:1)
plot(h2, phy2, cex=.7)
```

---

profiles.plot *Plot Marginal Distributions from MCMC*

Description

Simple plotting assistance for plotting output from MCMC runs

Usage

```
profiles.plot(y, col.line, col.fill, xlim=NULL, ymax=NULL, n.br=50,
              opacity=.5, xlab="Parameter estimate",
              ylab="Probability density", legend.pos=NULL, ...)
```

Arguments

`y`	Data frame, columns of which will be plotted as separate profiles.
`col.line`	Vector of colours for the lines.
`col.fill`	Vector of colours for the fill of the 95% most probable region of the distribution. If ommited, this will be a semi-transparent version of `col.line`.
`xlim`	X-axis limits - calculated automatically if omitted.
`ymax`	Y-axis upper limit - calculated automatically if omitted.
`n.br`	Number of breaks along the range of the data.
`opacity`	Opacity of the filled region (0 is transparent, 1 is fully opaque).
`xlab,ylab`	Axis labels for the plot.
`legend.pos`	String to pass to `legend` to position the legend (for automatic legend building based on the names of `y`).
`...`	Additional arguments passed through to `plot`.

Author(s)

Richard G. FitzJohn

Examples

```
## For usage, see the example in ?make.bd
```

---

quasse-common *Support Functions for QuaSSE Models*

Description

Utility functions for working with QuaSSE models. These provide a minimal set of state-varying functions, suitable for use with `make.quasse`, and simulation assistance functions for use with `tree.quasse`.

This is currently poorly explained!

Usage

```
constant.x(x, c)
sigmoid.x(x, y0, y1, xmid, r)
stepf.x(x, y0, y1, xmid)
noroptimal.x(x, y0, y1, xmid, s2)

make.linear.x(x0, x1)

make.brownian.with.drift(drift, diffusion)
```

Arguments

`x`	Character state
`c`	Constant.
`y0`	y value at very small `x` (limit as `x` tends to negative infinity)
`y1`	y value at very large `x` (limit as `x` tends to infinity). For `noroptimal.x`, this is the y value at `xmid`.
`xmid`	Midpoint (inflection point) of sigmoid or step function
`r`	Rate at which exponential decay occurs or sigmoid changes - higher values are steeper
`s2`	Variance of the normal distribution specified by `noroptimal.x`.
`x0`	Lower x limit for the linear function: y will take value at x0 for all x smaller than this
`x1`	Upper x limit for the linear function: y will take value at x1 for all x greater than this
`drift`	Rate of drift
`diffusion`	Rate of diffusion (positive)

Details

The linear function returned by (`make.linear.x`) will go to zero wherever negative. This may not always be desired, but is required for valid likelihood calculations under QuaSSE.

Author(s)

Richard G. FitzJohn

`set.defaults` *Set Default Arguments of a Function*

Description

Set the default values of formal arguments of a function.

Usage

```
set.defaults(f, ..., defaults)
```

Arguments

`f`	A function
`...`	Named arguments to be set
`defaults`	A named list of arguments

Details

The repetitive argument lists of many of diversitree's likelihood functions are the motivation for this function.

For example, the likelihood function that `make.bisse` produces takes arguments `condition.surv`, `root`, and `root.p`, each with default values. If you dislike the defaults, you can change them by passing in alternative values when computing likelihoods, or when doing an ML search. However, this can get tedious if you are using a function a lot, and your code will get cluttered with lots of statements like `condition.surv=FALSE`, some of which you may forget. See the example below for how to avoid this.

Author(s)

Richard G. FitzJohn

Examples

```
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
set.seed(4)
phy <- tree.bisse(pars, max.t=30, x0=0)
lik <- make.bisse(phy, phy$tip.state)

## default arguments:
args(lik)

lik.no.cond <- set.defaults(lik, condition.surv=FALSE)
args(lik.no.cond)

## Multiple arguments at once:
lik2 <- set.defaults(lik, root=ROOT.GIVEN, root.p=c(0, 1))
args(lik2)

## Equivalently (using alist, not list -- see ?alist)
defaults <- alist(root=ROOT.GIVEN, root.p=c(0, 1))
lik3 <- set.defaults(lik, defaults=defaults)
identical(lik2, lik3)
```

sim.character *Simulate a Character Distribution on a Tree*

Description

Simulate a character distribution (state of each species) under some simple models of trait evolution. Currently this does not return the full history (node states, and state changes) but this may be added in a future version.

Usage

```
sim.character(tree, pars, x0=0, model="bm", br=NULL)
```

Arguments

tree	A phylogenetic tree, in ape "phylo" format.
pars	A set of model parameters (see Details below), as the order and interpretation depends on the model.
x0	Root state. The default is zero, which may not make sense in all models.
model	Character string specifying which model to evolve the character under. Possible values are "bm", "ou", "bbm", "mk" and "meristic"; see Details.
br	For cases where none of the models specifiable through the model argument are sufficient, you can provide your own function. The function must have arguments x0, t, which are the state at the base of a branch and the length of time to simulate over. It must return a scalar state at the tip of the branch. Future versions may change requirements of this function, especially to allow retention of character histories along branches.

Details

This function duplicates functionality in other packages; see sim.char in geiger in particular. The main difference here is that for continuous characters, this does not use the variance-covariance matrix, which can make it much faster for very large trees. I believe that this approach is similar to fastBM in phytools.

- model="bm": Brownian Motion. Takes a single parameter, representing the rate of diffusion (must be positive)
- model="ou": Ornstein-Uhlenbeck process. Takes a vector of three parameters, representing the rate of diffusion, strength of restoring force, and the "optimum" value. The first two parameters must be non-negative, and the rate of diffusion must be positive.
- model="bbm": Bounded Brownian Motion. Takes a vector of three parameters (s2, c, d), representing the rate of diffusion, lower and upper bound, respectively. The rate of diffusion must be positive.

  model="mk": Mk model (see make.mkn). Takes a Q matrix as its argument. The element Q[i,j] represents the rate of transition from state i to state j, and the diagonal elements must be such that rowSums(Q) is zero.

  model="meristic": A special case of the Mk model, where the trait is meristic and character transitions are only possible between adjacent states. There are three parameters (k, up, down), representing the number of states, and rate of character change up (from state i to i+1) and down.

Author(s)

Richard G. FitzJohn

`simulate` *Evolve Birth-Death Trees*

Description

Evolves one or more trees under the BiSSE (Binary State Speciation and Extinction) birth-death model, MuSSE (Multi-State Speciation and Extinction) model, or a simple character independent birth-death model. For BiSSE and MuSSE trees, it simultaneously evolves a binary character that affects speciation and/or extinction, and the tree itself.

Usage

```
trees(pars, type=c("bisse", "bisseness", "bd", "musse", "quasse", "yule"),
      n=1, max.taxa=Inf, max.t=Inf, include.extinct=FALSE, ...)

tree.bisse(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE,
           x0=NA)
tree.musse(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE,
           x0=NA)
tree.musse.multitrait(pars, n.trait, depth, max.taxa=Inf, max.t=Inf,
                      include.extinct=FALSE, x0=NA)

tree.quasse(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE, x0=NA,
            single.lineage=TRUE, verbose=FALSE)

tree.bisseness(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE,
               x0=NA)

tree.bd(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE)
tree.yule(pars, max.taxa=Inf, max.t=Inf, include.extinct=FALSE)

prune(phy, to.drop=NULL)
```

Arguments

`pars`	Vector of parameters. The parameters must be in the same order as an unconstrained likelihood function returned by `make.x`, for tree type `x`. The MuSSE simulator automatically detects the appropriate number of states, given a parameter vector.
`type`	Type of tree to generate: May be "bisse" or "bd".
`n`	How many trees to generate?
`max.taxa`	Maximum number of taxa to include in the tree. If `Inf`, then the tree will be evolved until `max.t` time has passed.
`max.t`	Maximum length to evolve the phylogeny over. If `Inf` (the default), then the tree will evolve until `max.taxa` extant taxa are present.

`include.extinct`
: Logical: should extinct taxa be included in the final phylogeny? And should extinct trees be returned by `trees`?

`x0`
: Initial character state at the root (state 0 or 1). A value of `NA` will randomly choose a state from the model's equilibrium distribution for a BiSSE model, but a non-NA value must be specified for MuSSE and QuaSSE.

`n.trait, depth`
: For `tree.musse.multitrait` only, these specify the number of binary traits and the style of parameters (with the same meaning as in `make.musse.multitrait`). The `pars` argument then needs to be in the same order as a likelihood function created by `make.musse.multitrait` with these arguments (this interface may be improved in future – email me if you find this annoying).

`single.lineage`
: (`tree.quasse` only): Start simulation with a single lineage? If `FALSE`, then the simulation starts with two lineages in state `x0` (i.e., immediately following a speciation event).

`verbose`
: (`tree.quasse` only): print verbose details about tree simuations. This can be reassuring for really large trees.

`...`
: Additional arguments

`phy`
: A phylogeny, possibly with extinct species, produced by one of the tree evolving functions.

`to.drop`
: Optional vector with the species names to drop.

Details

The phylogeny will begin from a single lineage in state `x0`, but the final phylogeny will include only branches above the first split.

`tree.bisse` may return an extinct phylogeny, and `trees` might return extinct phylogenies if `include.extinct` is `TRUE`.

Value

A `phylo` phylogenetic tree (ape format), or for `bisse.trees`, a `list` of `phylo` trees.

The trees will have an element `tip.state` that contains the binary state information.

Note

There are some logic problems around the creation of zero and one species trees; this will cause occasional errors when running the above functions. Things will change to fix this soon. All these functions may change in the near future.

Author(s)

Richard G. FitzJohn

Examples

```
pars <- c(0.1, 0.2, 0.03, 0.03, 0.01, 0.01)
set.seed(3)
phy <- tree.bisse(pars, max.taxa=30, x0=0)
phy$tip.state

h <- history.from.sim.discrete(phy, 0:1)
plot(h, phy)
```

```
## Retain extinct species:
set.seed(3)
phy2 <- tree.bisse(pars, max.taxa=30, x0=0, include.extinct=TRUE)
h2 <- history.from.sim.discrete(phy2, 0:1)
plot(h2, phy2)

#### MuSSE:
## Two states
pars <- c(.1, .2, .03, .04, 0.05, 0.1)
set.seed(2)
phy <- tree.musse(pars, 20, x0=1)

h <- history.from.sim.discrete(phy, 1:2)
plot(h, phy)

## A 3-state example where movement is only allowed between neighbouring
## states (1 <-> 2 <-> 3), and where speciation and extinction rates
## increase moving from 1 -> 2 -> 3:
pars <- c(.1,  .15,  .2,  # lambda 1, 2, 3
          .03, .045, .06, # mu 1, 2, 3
          .05, 0,         # q12, q13
          .05, .05,       # q21, q23
          0,   .05)       # q31, q32

set.seed(2)
phy <- tree.musse(pars, 30, x0=1, include.extinct=TRUE)

h <- history.from.sim.discrete(phy, 1:3)
plot(h, phy, cex=.7)

## And with extinct taxa pruned:
phy2 <- prune(phy)
h2 <- history.from.sim.discrete(phy2, 1:3)
plot(h2, phy2, cex=.7)

## This can all be done in one step (and is by default):
set.seed(2)
phy <- tree.musse(pars, 30, x0=1)
h <- history.from.sim.discrete(phy, 1:3)
plot(h, phy, cex=.7)
```

`time` *Support Functions for Time-varying Models*

Description

Utility functions for working with time-dependent models.

Usage

```
constant.t(t, c)
linear.t(t, c, m)
exponential.t(t, y0, y1, r)
```

```
sigmoid.t(t, y0, y1, tmid, r)
stepf.t(t, y0, y1, tc)

make.pars.t(functions, cache=NULL, check.negative.const=TRUE,
            check.negative.var=TRUE)
```

Arguments

`t`	Time
`c`	Constant. For `linear.t` this is the value at t=0.
`m`	Slope
`y0`	y value at time 0 (`exp.t`) or very early times (limit as t tends to negative infinity in `sigmoid.t`)
`y1`	y value approached as t tends to negative infinity
`r`	Rate at which exponential decay occurs or sigmoid changes - higher values are steeper
`tmid`	Midpoint (inflection point) of sigmoid
`tc`	"Critical time" at which y changes from `y0` to `y1`.
`functions`	A list of functions with `t` as their first argument.
`cache`	Internal use – please leave `NULL`.
`check.negative.const, check.negative.var`	Logicals, indicating if negative parameters should be checked for, for time-constant and time-varying parameters respectively.

Details

The linear function (`linear.t`) will go to zero wherever negative. This may not always be desired, but is often useful for models of rates.

Author(s)

Richard G. FitzJohn

`trait.plot` *Plot a Phylogeny and Traits*

Description

Plot a phylogeny and label the tips with binary traits. This function is experimental, and may change soon. Currently it is limited to radial trees and binary traits.

Usage

```
trait.plot(tree, dat, cols, lab=names(cols), str=0:1, class=NULL,
           type="f", w=1/50, legend=length(cols) > 1, cex.lab=.5,
           font.lab=3, cex.legend=.75, margin=1/4)
```

Arguments

tree	Phylogenetic tree, in ape format.
dat	A data.frame of trait values. The row names must be the same names as the tree (tree$tip.label), and each column contains the states (0, 1, or NA). The column names must give the trait names.
cols	A list, each element of which is a vector of length 2 being the colours for state 0 and state 1 for each trait. The list must be named so that all names appear in names(dat). Traits will be plotted in the order given by cols.
lab	Alternative names for the legend (perhaps longer or more informative). Must be in the same order as cols..
str	Strings used for the states in the legend.
class	A vector along phy$tip.label giving a higher level classification (e.g., genus or family). No checking is done to ensure that such classifications are not polyphyletic.
type	Plot type (same as type in ?plot.phylo). Currently only f (fan) is implemented.
w	Width of the trait plot, as a fraction of the tree depth.
legend	Logical: should a legend be plotted?
cex.lab, font.lab	Font size and type for the tip labels.
cex.legend	Font size for the legend.
margin	How much space, relative to the total tree depth, should be reserved when plotting a higher level classification.

Author(s)

Richard G. FitzJohn

Examples

```
## These are the parameters: they are a single speciation and extinction
## rate, then 0->1 (trait A), 1->0 (A), 0->1 (B) and 1->0 (B).
colnames(musse.multitrait.translate(2, depth=0))

## Simulate a tree where trait A changes slowly and B changes rapidly.
set.seed(1)
phy <- tree.musse.multitrait(c(.1, 0, .01, .01, .05, .05),
                             n.trait=2, depth=0, max.taxa=100,
                             x0=c(0,0))
## Here is the matrix of tip states (each row is a species, each column
## is a trait).
head(phy$tip.state)

trait.plot(phy, phy$tip.state,
           cols=list(A=c("pink", "red"), B=c("lightblue", "blue")))

nodes <- c("nd5", "nd4", "nd7", "nd11", "nd10", "nd8")
descendants <- function(node, phy) {
  ans <- node <- match(node, phy$node.label) + length(phy$tip.label)
  repeat {
    node <- phy$edge[phy$edge[,1] %in% node,2]
```

```
    if ( length(node) > 0 )
      ans <- c(ans, node)
    else
      break
  }

  ans <- unlist(ans)
  ans[ans <= length(phy$tip.label)]
}

grp <- lapply(nodes, descendants, phy)
class <- rep(NA, 100)
for ( i in seq_along(grp) )
  class[grp[[i]]] <- paste("group", LETTERS[i])

## Now, 'class' is a vector along phy$tip.label indicating which of six
## groups each species belongs.

## Plotting the phylogeny with these groups:
trait.plot(phy, phy$tip.state,
           cols=list(A=c("pink", "red"), B=c("lightblue", "blue")),
           class=class, font=1, cex.lab=1, cex.legend=1)
```

Index

∗Topic **datasets**
geosse.phy, 15
∗Topic **hplot**
plot.history, 65
profiles.plot, 67
trait.plot, 74
∗Topic **manip**
make.clade.tree, 38
∗Topic **models**
find.mle, 12
history.from.sim, 16
make.bd, 16
make.bd.split, 19
make.bd.t, 20
make.bisse, 22
make.bisse.split, 27
make.bisse.td, 29
make.bisseness, 32
make.bm, 36
make.classe, 39
make.geosse, 41
make.geosse.split, 43
make.mkn, 44
make.mkn.deriv, 47
make.musse, 47
make.musse.multitrait, 50
make.musse.split, 54
make.musse.td, 56
make.prior, 58
make.quasse, 59
make.quasse.split, 61
mcmc, 62
quasse-common, 68
sim.character, 70
simulate, 71
time, 73
∗Topic **model**
asr, 3
asr-bisse, 4
asr-mkn, 6
∗Topic **programming**
argnames, 2
combine, 9
constrain, 9
set.defaults, 69
∗Topic **utilities**
check, 8

ace, *13*
AIC, *14*
anova, *14*
anova.fit.mle *(find.mle)*, 12
argnames, 2, *10*, *14*, *45*
argnames<- *(argnames)*, 2
asr, 3, *4*
asr-bisse, 4
asr-mkn, 6
asr.bisse, *4*
asr.bisse *(asr-bisse)*, 4
asr.marginal.bisse *(asr-bisse)*, 4
asr.marginal.musse *(asr-bisse)*, 4
asr.mkn, *4*
asr.mkn *(asr-mkn)*, 6
asr.musse *(asr-bisse)*, 4

Bayesian, *22*, *32*, *37*, *39*, *41*, *47*, *50*, *59*
birthdeath, *13*
BiSSE-ness, *39*

check, 8
clades.from.classification *(make.clade.tree)*, 38
clades.from.polytomies *(make.clade.tree)*, 38
coef.fit.mle *(find.mle)*, 12
combine, 9
constant.t *(time)*, 73
constant.x *(quasse-common)*, 68
constrain, *2*, 9, *13*, *17*, *25*, *34*, *40*, *42*, *44*, *45*

data.frame, *24*, *34*

exponential.t *(time)*, 73

find.mle, *10*, 12, *17*, *25*, *34*, *40*, *42*, *45*

GeoSSE, *39*
geosse.phy, 15

history.from.sim, 16
history.from.sim.discrete, *66*

legend, *67*
linear.t (time), 73
logLik.fit.mle (find.mle), 12

make.asr.joint (asr), 3
make.asr.joint.mk2 (asr-mkn), 6
make.asr.joint.mkn (asr-mkn), 6
make.asr.marginal (asr), 3
make.asr.marginal.bisse (asr-bisse), 4
make.asr.marginal.mk2 (asr-mkn), 6
make.asr.marginal.mkn (asr-mkn), 6
make.asr.marginal.musse (asr-bisse), 4
make.asr.stoch (asr), 3
make.asr.stoch.mk2 (asr-mkn), 6
make.asr.stoch.mkn (asr-mkn), 6
make.bd, 16, *21*, *25*, *34*, *64*
make.bd.split, 19
make.bd.t, 20
make.bisse, *17*, *21*, 22, *28*, *30*, *33*, *34*, *38–43*, *45*, *48*, *51*, *52*, *55*, *57*, *64*
make.bisse.split, *19*, 27
make.bisse.t (make.bisse.td), 29
make.bisse.td, 29
make.bisseness, 32, *40*
make.bm, 36
make.brownian.with.drift (quasse-common), 68
make.clade.tree, *24*, *34*, 38
make.classe, 39
make.geosse, *40*, 41, *43*, *64*
make.geosse.split, 43
make.linear.x (quasse-common), 68
make.mk2 (make.mkn), 44
make.mkn, 44, *47*, *64*, *70*
make.mkn.deriv, 47
make.mkn.multitrait (make.musse.multitrait), 50
make.musse, *40*, 47, *52*
make.musse.multitrait, *47*, *48*, 50, *72*
make.musse.split, 54
make.musse.t (make.musse.td), 56
make.musse.td, 56
make.ou (make.bm), 36
make.pars.t, *30*
make.pars.t (time), 73
make.prior, *17*, *45*, 58, *62*
make.quasse, 59, *61*, *68*
make.quasse.split, 61
make.yule (make.bd), 16
maximum likelihood, *22*, *32*, *37*, *39*, *41*, *47*, *50*, *59*
mcmc, *17*, *25*, *34*, *40*, *42*, *45*, *58*, 62
mkn.multitrait.translate (make.musse.multitrait), 50
musse.multitrait.translate (make.musse.multitrait), 50

nlm, *13*
nlminb, *13*
noroptimal.x (quasse-common), 68

optim, *13*

par, *66*
plot, *67*
plot.history, 65
profiles.plot, 67
prune (simulate), 71

quasse-common, 68

sampler.norm (mcmc), 62
sampler.slice (mcmc), 62
set.defaults, 69
sigmoid.t (time), 73
sigmoid.x (quasse-common), 68
sigmoid2.x (quasse-common), 68
sim.character, 70
simulate, 71
starting.point.bd (make.bd), 16
starting.point.bisse (make.bisse), 22
starting.point.classe (make.classe), 39
starting.point.geosse (make.geosse), 41
starting.point.musse (make.musse), 47
starting.point.musse.multitrait (make.musse.multitrait), 50
starting.point.quasse (make.quasse), 59
stepf.t (time), 73
stepf.x (quasse-common), 68
subplex, *13*

time, 73
trait.plot, 74
tree.bd (simulate), 71
tree.bisse, *16*
tree.bisse (simulate), 71
tree.bisseness, *34*
tree.bisseness (simulate), 71
tree.musse (simulate), 71
tree.quasse, *68*
tree.quasse (simulate), 71
tree.yule (simulate), 71
trees (simulate), 71