Package 'bite'

April 5, 2020

Type Package

Title Bayesian Integrative Models of Trait Evolution

Version 0.3

Date 2020-03-17

Author Theo Gaboriau <theo.gaboriau@unil.ch>, Anna Kostikova <anna.kostikova@unil.ch>, Daniele Silvestro <daniele.Silvestro@unil.ch>, Sacha Laurent <Sacha.Laurent@unil.ch>, Martha Serrano <martha.serrano@unil.ch>, Glenn Litsios <glenn.litsios@unil.ch>, Wim Hordijk <wim@WorldWideWanderings.net>, Nicolas Salamin <nicolas.salamin@unil.ch>, Simon Joly <joly.simon@gmail.com>

Maintainer Theo Gaboriau <theo.gaboriau@unil.ch>

Description This package implements the jive (Joint Inter and Intra specific model of Variance Evolution) model and other Bayesian models aimed at understanding trait evolution. The goal of the package is to join phylogenetic comparative models (PCM) that tend to integrate various type of data (individual observations, environmental data, fossil data) into a hierarchical Bayesian framework. It contains various PCMs as well as functions to join those models into a hierarchical Bayesian framework in a flexible and user friendly way. It contains MCMC algorithms, methods for model comparison and various plotting function for pre- and post- processing data visualization. Finally, this package integrates functions allowing bridges between R and the BEAST2 implementations of PCMs. Kostikova A, Silvestro D, Pearman PB, Salamin N (2016) (doi: DOI:10.1093/sysbio/syw010). Gaboriau T, Mendes FK, Joly S, Silvestro D, Salamin N (in prep).

Depends R (>= 3.0.3)

Imports ape, MASS, phytools, coda, sm, vioplot, xml2

Suggests mvMORPH

License GPL-2

Repository CRAN

NeedsCompilation no

Encoding UTF-8

RoxygenNote 6.1.1

R topics documented:

`Anolis_map` *Anolis map of regimes*

Description

Anolis map of regimes

Usage

```
data(Anolis_map)
```

Format

A two columns matrix giving the evolutionary time spent by each branch of `Anolis_tree` in the two islands C: Cuba and H: Hispniola

Details

This data contains a map of the Anolis lizards' clades geographic position onto the Anolis phylogeny. A two columns matrix giving the evolutionary time spent by each branch of `Anolis_tree` in the two islands C: Cuba and H: Hispniola. Each row represents a branch of `Anolis_tree`, numbers represent the evolutionary time spent on each island by the clade.

Author(s)

Theo Gaboriau

`Anolis_traits` *Anolis snout to vent length*

Description

Anolis snout to vent length

Usage

```
data(Anolis_traits)
```

Format

A three columns data.frame with species names (species) and snout to vent length (svl) and elevation of individual observations

Details

This data has been extracted from the study of Munoz et al., 2014 It contains individual observations of Anolis lizards' snout to vent length (mm) and elevation (m) from Hispaniola and Cuba islands in a three columns data.frame with species names (species) and snout to vent length (svl) and elevation of individual observations.

Author(s)

Martha Munoz, Johanna Wegener and Adam Algar.

References

Munoz, M. M., Wegener, J. E., and Algar, A. C. (2014). Untangling intra- and interspecific effects on body size clines reveals divergent processes structuring convergent patterns in Anolis lizards. Am. Nat. , 184(5):636-46.

Munoz, Martha M., Wegener, Johanna E., Algar, Adam C. (2014), Data from: Untangling intra- and interspecific effects on body size clines reveals divergent processes structuring convergent patterns in Anolis lizards, Dryad, Dataset, https://doi.org/10.5061/dryad.q39h2

`Anolis_tree` *Anolis phylogenetic tree*

Description

Anolis phylogenetic tree

Usage

```
data(Anolis_tree)
```

Format

A dated phylogenetic tree of class "phylo" including 16 species of Anolis lizards

Details

This phylogenetic tree as been extracted from the Anolis phylogeny of Poe et al., 2017 It contains dated phylogenetic tree of class "phylo" including 16 species of Anolis lizards

Author(s)

Theo Gaboriau

References

Poe, S., Nieto&#45;Montes de Oca, A., Torres&#45;carvajal, O., De Queiroz, K., Velasco, J. A., Truett, B., Gray, L. N., Ryan, M. J., Kohler, G., Ayala&#45;varela, F., and Latella, I. (2017). A Phylogenetic, biogeographic, and taxonomic study of all extant species of Anolis (Squamata, Iguanidae). Syst. Biol., 66(5):663-697

`control_jive` *Control tuning parameters of the jive algorithm*

Description

This function modifies a jive object to tune the jive mcmc algorithm. The output will be different regarding which level of the jive model the user wants to tune ($lik, $priors). This function allows tuning of : initial window size for proposals, starting parameter value, proposal methods, Hyperpriors and update frequencies

Usage

```
control_jive(jive, level = c("lik", "prior"), intvar = NULL,
  pars = NULL, window.size = NULL, initial.values = NULL,
  proposals = NULL, hyperprior = NULL, update.freq = NULL)
```

Arguments

`jive`	a jive object obtained from `make_jive`
`level`	character taken in c("lik", "prior") to specify on which level of the jive model, the control will operate (see details)
`intvar`	character taken in names(jive$priors) giving the variable to be edited (see details)
`pars`	vector of character taken in names(jive$priors[[intvar]]$init) giving the names of the hyper parameter to be edited
`window.size`	initial window size for proposals during the mcmc algorithm. matrix or vector depending on the value of level and nreg (see details)
`initial.values`	starting parameter values of the mcmc algorithm. matrix or vector depending on the value of level and nreg (see details)
`proposals`	vector of characters taken in c("slidingWin", "slidingWinAbs", "logSlidingWinAbs","multiplierProposal", "multiplierProposalLakner","logNormal", "absNormal") to control proposal methods during mcmc algorithm (see details)
`hyperprior`	list of hyperprior functions that can be generated with `hpfun`function. Ignored if level == "lik" (see details)
`update.freq`	numeric giving the frequency at which parameters should be updated.

Details

If level == "lik" changes will be applied to the likelihood level of the algorithm. intvar is giving the variable on whic the changes will be operated window.size and initial.values must be entered as a vector of length equal to the number of species. proposal must be a character

If level == "prior" changes will be applied to the prior level of the algorithm. intvar is giving the variable on which te change will be operated. window.size and initial.values must be entered as a vector of size equal to the number of parameters or equal to the length of pars.

Note that if you want to change the tuning at the three levels of the algorithm, you will have to use the control_jive function three times

proposals Has to be one the following : "slidingWin" for Sliding window proposal unconstrained at maximum, "multiplierProposal", for multiplier proposal

Hyperprior list of hyperpriror functions (see `hpfun`). User must provide a list of size equal to the number of parameters or equal to the length of pars

Value

A JIVE (of class "JIVE" and "list") object to parse into mcmc_bite function (see `make_jive`)

Author(s)

Theo Gaboriau

Examples

```
data(Anolis_traits)
data(Anolis_tree)

## Create a jive object
my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3],
model.priors = list(mean = "BM", logvar= c("OU", "root")))

## change starting values for the species means
my.jive$lik$init #default values
new.init <- rep(40,16)
my.jive <- control_jive(my.jive, level = "lik", intvar = "mean", initial.values = new.init)
my.jive$lik$init #mean initial values changed

 ## change hyperpriors for prior.mean
 plot_hp(my.jive) #default values
 new.hprior <- list(hpfun("Gamma", hp.pars = c(2,6)), hpfun("Uniform", c(20,80)))
my.jive <- control_jive(my.jive, level = "prior", intvar = "mean", hyperprior = new.hprior)
 plot_hp(my.jive) #mean initial values changed
```

`format_jive_traits` *Format traits matrix to parse into* `make_jive` *function*

Description

This function takes a dataframe with indivdual observations ans species names and returns a traits matrix to to parse into `make_jive` function

Usage

```
format_jive_traits(obs)
```

Arguments

`obs`	a dataframe with species names in the first column and individual trait values in the second columns

Value

a list of size equal to the number of species. Each element contains a vector of numerical values representing individual observation for that species.

Author(s)

Theo Gaboriau

`hpfun` *Hyper-prior function*

Description

This function creates a hyper-prior density function. Currently supported density function are Uniform, Gamma and Normal. The resulting function is used during MCMC `mcmc_bite` to estimate parameters of priors.

Usage

```
hpfun(hpf = "Uniform", hp.pars = c(1, 2), ...)
```

Arguments

`hpf`	name of a density function. Supported density functions are: Uniform, Gamma and Normal (abbreviations are not supported)
`hp.pars`	a vector of density function parameters
`...`	additional parameters that can be passed to a density function

Details

There are three currently implemented density function: Uniform, Gamma and Normal. Each of these densities requires two input parameters and hp.pars must be a vector of two values and cannot be left empty.

Value

A hyper-prior density function (of class "function")

Author(s)

Anna Kostikova and Daniele Silvestro

Examples

```
my.hp <- hpfun(hpf="Uniform", hp.pars=c(1,2))
```

---

make_jive *Create a list that can be used as an input to mcmc_bite*

Description

This function creates a jive object from a matrix of intraspecific observations and species phylogeny. The obtained jive object is a list that can than be used as an input to `mcmc_bite` function Intraspecific observations should be stored as matrix, where lines are vector of observations for each species, with NA for no data. Phylogenetic tree can be either a simmap object (`make.simmap`) or phylo object (`as.phylo`)

Usage

```
make_jive(phy = NULL, traits, map = NULL, model.priors = list(mean =
  "BM", logvar = "OU"), scale = FALSE, nreg = NULL, lik.f = NULL,
  init = NULL)
```

Arguments

`phy`	phylogenetic tree provided as either a simmap or a phylo object
`traits`	matrix of traits value for every species of phy (see details)
`map`	matrix mapping regimes on every edge of phy (see details)
`model.priors`	list giving model specification for trait evolution preferably given along with variable names. Supported models are "OU", "BM", "WN". The user can also specify if the assumptions of the model should be relaxed and can also enter a function (see details)
`scale`	boolean indicating whether the tree should be scaled to unit length for the model fitting
`nreg`	integer giving the number of regimes for a Beast analysis. Only evaluated if phy == NULL
`lik.f`	alternative likelihood function of the form function(pars.lik, traits, counts) to model intraspecific variation (see details)
`init`	matrix giving initial values for parameters with the variables in rows and the species in columns (see examples)

Details

This function creates a jive object needed for `mcmc_bite` function. Trait values must be stored as a matrix, where lines are vectors of observations for each species, with NA for no data. Rownames are species names that should match exactly tip labels of the phylogenetic tree.

Phylogenetic tree must be provided as either simmap object or as a phylo object. If the phylogenetic tree is a phylo object but model specification indicates multiple regimes, user must provide a mapping of the regime in map. If you keep the phy = NULL options the JIVE object can only be parsed to the `xml_bite` function.

map is a matrix giving the mapping of regimes on phy edges. Each row correspond to an edge in phy and each column correspond to a regime. If map is provided the map from the simmap object is ignored.

trait evolution can be modeled with Ornstein-Uhlenbeck (OU), Brownian Motion (BM) or White Noise (WN) processes. Multiple regimes can be defined for both models and will apply on thetas: c("OU", "theta"), sigmas: c("OU", "sigma") or alphas: c("OU", "alpha") for OU and on sigmas only for WN: c("WN", "sigma") and BM: c("BM", "sigma"). While using the OU model, the user can also relax the stationarity of the root: c("OU", "root") and relax several assumptions at the same time c("OU", "root", "theta") Species-specific distributions are modeled as multivariate normal distributions. User defined functions of trait evolution can be used in model.priors. The function should be of the form: function(tree, x, pars) and return a loglikelihood value with "tree" being the phylogenetic tree, x being a vector of trait value of size equal to the number of species and ordered as tree$tip.label and pars should be a vector of model parameters (see examples)

parameters used in the different pre-defined models:

White Noise model (WN):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in model.priors

Brownian Motion model (BM):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in model.priors

Ornstein Uhlenbeck model (OU):

- root: root value. Only used if "root" is specified in model.priors
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in model.priors
- theta: optimal value, n regimes if "theta" is specified in model.priors
- alpha: strength of selection, n regimes if "alpha" is specified in model.priors

Value

A list of functions and tuning parameters (of class "JIVE" and "list") representing the plan of the hierarchical model to parse into `mcmc_bite`.

Author(s)

Theo Gaboriau, Anna Kostikova, Daniele Silvestro and Simon Joly

See Also

`xml_bite`, `mcmc_bite`

Examples

```
## Load test data
data(Anolis_traits)
data(Anolis_tree)
data(Anolis_map)

## JIVE object to run jive with single regimes
```

```
my.jive <- make_jive(phy = Anolis_tree, traits = Anolis_traits[,-3],
 model.priors = list(mean = "BM", logvar= c("OU", "root")))

## JIVE object to run jive with multiple regimes
my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3], map = Anolis_map,
 model.priors =list(mean = "BM", logvar = c("OU", "theta", "alpha")))

## JIVE object to run jive from an ancestral state reconstruction (stochastic mapping)
# First generate simmap object
library(phytools)
n= length(Anolis_tree$tip.label)
trait = rep(0,n)
trait[c(4,3,14,16, 6,5)] = 1
names(trait) =  Anolis_tree$tip.label

mapped_tree=make.simmap(Anolis_tree, trait, model='SYM')
plotSimmap(mapped_tree)

my.jive <- make_jive(mapped_tree, Anolis_traits[,-3]
, model.priors = list(mean = "OU" , logvar = c("OU", "theta")))

 ## Jive object using another model of trait evolution (EB from mvMORPH)
 library(mvMORPH)
 early_burst <- function(tree, x, pars){
  suppressMessages(mvEB(tree, x, method = "inverse", optimization = "fixed",
   echo = FALSE)$llik(pars, root.mle = FALSE))
 }

 my.jive <- make_jive(phy = Anolis_tree, traits = Anolis_traits[,-3]
, model.priors = list(mean = early_burst , logvar = c("OU", "root")))
 initial.values <- c(0.1, 1, 50)
 window.size <- c(0.1, 0.2, 1)
 proposals <- list("slidingWin", "slidingWin", "slidingWin")
 hyperprior <- list(hpfun("Gamma", hp.pars = c(1.1, 5)), hpfun("Gamma", hp.pars = c(3, 5)),
                    hpfun("Uniform", hp.pars = c(30, 80)))
 names(initial.values) <- names(window.size) <- c("sigma_sq", "beta", "root")
 names(proposals) <- names(hyperprior) <- c("sigma_sq", "beta", "root")
 my.jive <- control_jive(jive = my.jive, level = "prior", intvar = "mean",
  pars = names(initial.values), window.size = window.size,
  initial.values = initial.values, proposals = proposals, hyperprior = hyperprior)

 ## Jive object using another model of intraspecific variation (uniform model)
 lik_unif <- function(pars.lik, traits, counts){
  if(!"mid" %in% names(pars.lik)) stop("'mid' parameter cannot be found in model.priors")
   if(!"logrange" %in% names(pars.lik)){
    stop("'logrange' parameter cannot be found in model.priors")
   }

   min.sp <- pars.lik$mid - 1/2*exp(pars.lik$logrange)
   max.sp <- pars.lik$mid + 1/2*exp(pars.lik$logrange)

   log.lik.U <- sapply(1:length(traits), function(i){
   sum(dunif(traits[[i]], min.sp[i], max.sp[i], log = TRUE))
   })

   if (is.na(sum(log.lik.U))) {
     return(-Inf)
   } else {
     return(log.lik.U)
   }
 }

 init_unif <- sapply(Anolis_tree$tip.label, function(sp){
  logrange <- log(diff(range(Anolis_traits[Anolis_traits[,1] == sp, 3])) + 2)
  mid <- mean(range(Anolis_traits[Anolis_traits[,1] == sp, 3]))
  c(mid = mid, logrange = logrange)
 })

 my.jive <- make_jive(phy = Anolis_tree, traits = Anolis_traits[,-2],
 model.priors = list(mid = "BM" , logrange = c("OU", "root")),
 lik.f = lik_unif, init = init_unif)
```

marginal_lik   *Calculate marginal likelihood by thermodynamic integration (LTI)*

Description

Calculate the marginal likelihood from a logfile generated by `mcmc_bite` with thermodynamic integration (Lartillot and Philippe, 2006) or stepping stone (Xie et al., 2011).

Usage

```
marginal_lik(mcmc.log, burnin = 0, method = "SS")
```

Arguments

`mcmc.log`	the output file of a `mcmc_bite` run
`burnin`	number or proportion of iteration to delete
`method`	one of "TI" for thermodynamic integration and "SS" for stepping stone integration (the default)

Value

a length one numeric double giving the marginal likelihood of the model.

Author(s)

Theo Gaboriau and Simon Joly

Examples

```
## Load test data
data(Anolis_traits)
data(Anolis_tree)
data(Anolis_map)

## Run a MCMC chain with thermodynamic Integration
set.seed(300)
my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3],
  model.priors = list(mean="BM", logvar="OU"))
mcmc_bite(my.jive, log.file="my.jive_MCMC_TI.log", ncat=10, sampling.freq=10,
 print.freq=100, ngen=1000, burnin=0)

## import the results in R
logfile <- "my.jive_MCMC_TI.log"
res <- read.csv(logfile, header = TRUE, sep = "\t")

mlikTI <- marginal_lik(res, burnin = 0.1, method = "TI")
mlikTI

mlikSS <- marginal_lik(res, burnin = 0.1, method = "SS")
mlikSS
```

mcmc_bite *MCMC algorithm*

Description

Implements Markov chain Monte Carlo sampling for trait evolution models

Usage

```
mcmc_bite(model, log.file = "bite_mcmc.log", sampling.freq = 1000,
  print.freq = 1000, ncat = 1, beta.param = 0.3, ngen = 5e+06,
  burnin = 0)
```

Arguments

`model`	an object of class "jive" or other objects from the bite package (see details)
`log.file`	name of the output file that will store the log of MCMC chain
`sampling.freq`	sampling frequency of the MCMC chain (how often chain will be saved into output file
`print.freq`	printing frequency of the MCMC chain (how often chain will be printed in the R console). Setting it to 0 will suppress every printed message
`ncat`	number of classes for thermodynamic integration (see details)
`beta.param`	beta value to define classes for thermodynamic integration (see details)
`ngen`	number of generation in MCMC chain
`burnin`	a burning phase of MCMC chain (has to be specified for thermodynamic integration)

Details

This function runs MCMC sampling on jive object `make_jive` or objects describing other models of the bite package. The jive object contains both the dataset and set of model to be used in MCMC. This function implements both a conventional MCMC and an MCMC with thermodynamic integration. The latter option is turned off by default and can be changed by setting ncat to values > 1. The recommended ncat for TI is 10. When setting ncat > 1, make sure to specify burning. As a rule of thumb set burning to 1/10 fraction of ngen.

Value

Generates a log file in the users filespace at the path defined by log.file

Author(s)

Theo Gaboriau, Anna Kostikova, Daniele Silvestro, and Simon Joly

Examples

```
## Load test data
data(Anolis_traits)
data(Anolis_tree)
data(Anolis_map)

## Run a simple MCMC chain
set.seed(300)
my.jive <- make_jive(phy = Anolis_tree, traits = Anolis_traits[,-3],
 model.priors = list(mean = "BM", logvar= c("OU", "root")))
mcmc_bite(model = my.jive, log.file="my.jive_MCMC.log",
sampling.freq=10, print.freq=0, ngen=1000)

## Run an MCMC chain with thermodynamic integration
mcmc_bite(my.jive, log.file="my.jive_MCMC_TI.log", ncat=10,
 sampling.freq=10, print.freq=100, ngen=1000, burnin=10)
```

`plot_bf` *Plots summary of Bayes Factors calculations*

Description

Lolipop plot representing values of BF (Bayes Factor) scores for different models

Usage

```
plot_bf(m.liks, thr = 2, dir = c("vertical", "horizontal"),
  col = c("#d32f23", "#2e86ab", "#000000"), col.thr = c("#a6e1fa"),
  ax.lab = "log(BF)", main = "", rank = TRUE, dec = TRUE,
  group.pattern = NULL, cex = c(1.2, 1, 1), mod.lab = NULL,
  srt.lab = 0, adj.lab = 0, space = c(1.2, 0.8))
```

Arguments

m.liks	matrix of marginal likelihoods (see details)
thr	value of BF threshold for model selection (default is 2), Several thresholds can be given in the form of a vector
dir	string giving the direction of the plot ("vertical", "horizontal")
col	color of the lines and dots. Could be of size one or more. The first element applies to the best model, the second to the ones below thr[1] and so on...
col.thr	color for the threshold area (must be at the sime length as thr)
ax.lab	label of the axis.

`main`	an overall title for the plot
`rank`	logical: should models be ranked by BF scores? (default is TRUE)
`dec`	logical: should models be displayed in decreasing order? (default is TRUE)
`group.pattern`	Regular expression given a pattern to be matched in names(m.liks). The values matching that pattern will be grouped in the plot
`cex`	size of the dots and width of the lines. Could be of size one or more. The first element applies to the best model, the second to the ones below thr[1] and so on...
`mod.lab`	name of the models. If mod.lab = NULL the names of m.liks are used
`srt.lab, adj.lab`	rotation and justification of model names, see srt and adj in `par`
`space`	only evaluated if group.pattern != NULL, numeric(2) indicating the space between groups and the space between members of the same groups

Author(s)

Theo Gaboriau

Examples

```
## Fake marginal likelihood data
m.liks <- c(20 ,33, 56, 51, 55, 12)
names(m.liks) <- c("MBM-VBM", "MBM-VWN", "MBM-VOU", "MOU-VBM", "MOU-VWN", "MOU-VOU")

#Does not go well with default margin sizes
oldmar <- par()$mar
par(mar = c(5,1,4,6))
plot_bf(m.liks)
plot_bf(m.liks, thr = c(2,6), col.thr = c("#a2c5ac", "#ade1e5"),
 col = c("#d32f23", "#468189","#2e86ab","#000000"), cex = c(1.2,1,0.8,0.8))
plot_bf(m.liks, group.pattern = "MBM", rank = FALSE)
plot_bf(m.liks, group.pattern = "MOU", rank = TRUE)
plot_bf(m.liks, group.pattern = c("VWN", "VOU", "VBM"), rank = TRUE)
par(mar = c(6,5,1,2))
plot_bf(m.liks, dir = "horizontal", srt.lab = -60, adj.lab = c(0,0.8))
par(mar = oldmar)
```

`plot_hp` *plot Hyper-prior function*

Description

This function plots a hyper-prior density function. Currently supported density function are Uniform, Gamma, Normal, Loggamma and Lognormal. The resulting function is used during MCMC `mcmc_bite` to estimate parameters of priors.

Usage

```
plot_hp(hpf, col = c("#bfdbf7", "#f49e4c"), border = c("#2e86ab",
  "#a31621"), bty = "n", ...)
```

Arguments

`hpf`	name of a density function. Supported density functions are: Uniform, Gamma and Normal
`col`	color of the density area. Can be of size 2 (hpriors for the means, hpriors for the logvars) if a jive object is plotted
`border`	color of the density curve. Can be of size 2 (hpriors for the means, hpriors for the logvars) if a jive object is plotted
`bty, ...`	additional parameters that can be passed to a density function and `par`

Details

There are three currently implemented density function: Uniform, Gamma and Normal. Each of these densities requires two input parameters and hp.pars must be a vector of two values and cannot be left empty.

Author(s)

Theo Gaboriau

Examples

```
 ## Load test data
 data(Anolis_traits)
 data(Anolis_tree)

 my.hp <- hpfun(hpf="Uniform", hp.pars=c(1,2))
 plot_hp(my.hp)

my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3], model.priors = list(mean="BM", logvar="OU"))
 plot_hp(my.jive, cex.main = .8)
```

`plot_jive` *plot input data from a jive object*

Description

This function plots the phylogenetic tree, the trait data and the map used as an input for a jive analysis

Usage

```
plot_jive(jive, col.map = NULL, col = "lightgrey",
  show.tip.label = TRUE, show.models = TRUE,
  direction = "rightwards", trait.lab = "x", trait.lim = NULL,
  srt.label = 0, c.reg = NULL, tip.color = "#000000", ...)
```

Arguments

`jive`	a jive object built with the function `make_jive`
`col.map`	a vector of mode character indicating colors of the edges for the map ploting. It should be of same size than the number of regimes
`col`	a character indicating the color of the vioplots
`show.tip.label`	a logical indicating whether to show the tip labels on the phylogeny (defaults to TRUE, i.e. the labels are shown).
`show.models`	a logical indicating whether to show details about model specification in the jive object.
`direction`	a character string specifying the direction of the tree. Two values are possible: "rightwards" (the default) and "upwards".
`trait.lab`	a charachter specifying the axis label for the traits
`trait.lim`	a vector of mode numeric indicating the limits for trait ploting
`srt.label`	an integer indicating the string rotation in degrees for the tip labels
`c.reg`	a real number indicating where to plot the names of the regimes. The names are not plotted if c.reg == NULL.
`tip.color`	the colours used for the tip labels, eventually recycled.
`...`	additional parameters that can be passed to `vioplot`

Author(s)

Theo Gaboriau

Examples

```
data(Anolis_traits)
data(Anolis_tree)
data(Anolis_map)

colnames(Anolis_map) <- c("Hispaniola", "Cuba")
my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3],
 model.priors = list(m="BM", v = "OU"))
par(cex.lab = .8, cex.axis = .8, las = 1, mgp = c(2,0.5,0))
plot_jive(jive = my.jive, show.tip.label = TRUE,
trait.lab = "Snout to vent length (cm)", srt.label = 0, c.reg = 2)

my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3], Anolis_map,
 model.priors = list(m = "BM", v = c("OU", "theta")))
par(cex.lab = .8, cex.axis = .8, las = 1, mgp = c(2,0.5,0))
plot_jive(jive = my.jive, show.tip.label = TRUE, c.reg = 2,
 trait.lab = "Snout to vent length (cm)", srt.label = 70, direction = "upwards")
```

plot_mcmc_bite *Plot trace and density from a log file*

Description

This function plots the trace and/or density of each mcmc sample.

Usage

```
plot_mcmc_bite(mcmc.log, type = c("trace", "density"), burnin = 0,
  variable = NA, label = NA, col = "#000000", cex.est = 1,
  bty = "n", kp.burn = FALSE, ...)
```

Arguments

`mcmc.log`	Any mcmc sample with the saved iterations in rows and the variables in columns
`type`	Character taken in c("trace", "density"). If both are specified, they are plotted side by side in the same graphical device
`burnin`	The size of the burnin in number of iterations or the proportion of iteration you want to remove
`variable`	The name or number of the variable to plot. If is.na(variable), all columns of mcmc.log will be plotted except "iter" and "temperature"
`label`	Full variable name to be plotted
`col, bty, ...`	Other graphical parameters to parse to `par`
`cex.est`	The magnification to be used for estimates display
`kp.burn`	Logical specifying whether the plot window should adjust to the pre-burnin values (only evaluated if "trace" in type)

Author(s)

Theo Gaboriau

Examples

```
 ## Load test data
 data(Anolis_traits)
 data(Anolis_tree)
 data(Anolis_map)

 ## Run a simple MCMC chain
my.jive <- make_jive(Anolis_tree, Anolis_traits[-3],  model.priors = list(mean="BM", logvar="OU"))
mcmc_bite(my.jive, log.file="my.jive_mcmc.log", sampling.freq=10, print.freq=10, ngen=1000)

 ## import the results in R
 logfile <- "my.jive_mcmc.log"
 res <- read.csv(logfile, header = TRUE, sep = "\t")

 ## plot the results
 plot_mcmc_bite(res, burnin = 0.2, variable = NA, cex.est = .7)
 plot_mcmc_bite(res, burnin = 0.2, variable = "prior.mean", cex.est = .7)
```

plot_post_beast *Plot posterior probabilities from beast*

Description

This function plots the phylogenetic tree along with mean posterior probabilities of the chosen parameter. The default plots paramenter value under multiple regimes estimated with the beast implementation of JIVE, OU, BM and WN. Options are included to plot node support and age bars.

Usage

```
plot_post_beast(mcc, post = TRUE, post.var = "rate",
  post.cex = par("cex"), post.col = "#ee964b", post.alpha = 1,
  post.border = "#000000", post.lwd = par("lwd"), leg = TRUE,
  leg.frac = c(0.2, 0.5), leg.lab = "rate", leg.cex = par("cex")/2,
  sup = TRUE, sup.var = "posterior", sup.cex = par("cex")/2,
  bars = FALSE, bar.var = "height_95%_HPD", bar.col = "#2e86ab",
  bar.alpha = 0.8, bar.thc = par("cex")/10, bar.border = "#000000",
  bar.lwd = par("lwd"), ...)
```

Arguments

`mcc`	A character containing the path to the Maximum credibility tree from the Beast analysis and extracted from treeAnnotator
`post`	A logical specifying whether the mean posterior value of a parameter should be plot along the tree
`post.var`	A character containing the name of the parameter (only evaluated if post == TRUE)
`post.cex`	A numeric giving the size of the dots (only evaluated if post == TRUE)
`post.col`	Color of the dots (only evaluated if post == TRUE)
`post.alpha`	A numeric giving the transparency of the dots (only evaluated if post == TRUE)
`post.border`	Color of the dots' border (only evaluated if post == TRUE)
`post.lwd`	Line width of the dots' border (only evaluated if post == TRUE)
`leg`	A logical specifying whether a legend should be plotted (only evaluated if post == TRUE)
`leg.frac`	A vector giving the proportion of the plot window taken by the legend (only evaluated if post == TRUE)
`leg.lab`	Label of the legend (only evaluated if post == TRUE)
`leg.cex`	A numeric giving the size of the legend (only evaluated if post == TRUE)
`sup`	A logical specifying whether node support information should be displayed
`sup.var`	A character containing the name of the support variable (only evaluated if sup == TRUE)
`sup.cex`	A numeric giving the size of node support information (only evaluated if sup == TRUE)
`bars`	A logical specifying whether node age bars should be plotted

`bar.var`	A character containing the name of the node age variable. MIN and MAX should be available for this variable (only evaluated if bars == TRUE)
`bar.col`	Color of the bars (only evaluated if bars == TRUE)
`bar.alpha`	A numeric giving the transparency of the bars (only evaluated if bars == TRUE)
`bar.thc`	A numeric giving the thickness of the bars (only evaluated if bars == TRUE)
`bar.border`	Color of the bars' border (only evaluated if bars == TRUE)
`bar.lwd`	Line width of the bars' border (only evaluated if bars == TRUE)
`...`	additional parameters that can be parsed to `plot.phylo`

Details

leg = TRUE changes the proportion of the plot.window taken by the plot. Use `par(fig = c(0,1,0,1))` to restore default parameters

Value

plot

Author(s)

Theo Gaboriau

`plot_pvo` *Plots estimates of species traits distribution*

Description

Density plot representing estimated species trait distributions under a jive model. This function plots the mean or median density distribution and the HPD distributions assuming that the trait is normally distributed

Usage

```
plot_pvo(phy, traits, map = NULL, mcmc.log, tip = NA, burnin = 0.1,
  conf = 0.95, stat = "median", trait.lab = "x", col = NULL,
  lab = TRUE, lolipop = c(0.4, 0.4), cex.tip = par("cex"),
  var.f = NULL, ...)
```

Arguments

`phy`	phylogenetic tree provided as either a simmap or a phylo object
`traits`	trait data used to perform the jive analysis. This has to be of the same form as the one used in `make_jive`
`map`	map used to perform the jive analysis. This has to be of the same form as the one used in `make_jive`
`mcmc.log`	the output file of a `mcmc_bite` run
`tip`	An integer giving the row corresponding to the species to be plotted. If tip == NA, the posterior distribution of every tip is plotted along with the phylogenetic tree

burnin	The size of the burnin in number of iterations or the proportion of iteration you want to remove
conf	A number of [0,1] giving the confidence level desired.
stat	A character giving the function to be used to estimate species mean and variance from the posterior distributions. Must be one of be "mean" and "median"
trait.lab	a charachter specifying the axis label for the traits
col	color of the density filling. Must be of size two for estimates and HPD. If col and border are NULL, two random colors are assigned
lab	logical indicating whether to show species name in the plot. Only evaluated if tip =! NA
lolipop	size and width of the lolipops representing samples
cex.tip	size of the tips
var.f	alternative distribution used to model intraspecific variation of the form function(n, pars). The function must return n samples from the given distribution.
...	Additional parameters that can be parsed to plot

Author(s)

Theo Gaboriau

Examples

```
 ## Load test data
 data(Anolis_traits)
 data(Anolis_tree)
 data(Anolis_map)

 # Run a simple MCMC chain
my.jive <- make_jive(Anolis_tree, Anolis_traits[,-3], model.priors=list(mean="BM", logvar = "OU"))
mcmc_bite(my.jive, log.file="my.jive_mcmc.log", sampling.freq=10, print.freq=10, ngen=5000)

 ## import the results in R
 logfile <- "my.jive_mcmc.log"
 res <- read.csv(logfile, header = TRUE, sep = "\t")

 plot_pvo(phy = Anolis_tree, traits = Anolis_traits, mcmc.log = res)
```

sim_jive *Simulate JIVE process*

Description

Generate random values of trait mean and variance simulated under a JIVE process along a phylogenetic tree

Usage

```
sim_jive(phy, map = NULL, models = list(mean = c("BM"), logvar =
  c("OU")), pars = list(mean = c(root = 0, sigma_sq = 0.1), logvar =
  c(root = 2, theta = 1, sigma_sq = 0.1, alpha = 1)), sampling = c(1, 7),
  bounds = list(c(-Inf, Inf), c(-Inf, Inf)), var.f = NULL)
```

Arguments

`phy`	Phylogenetic tree
`map`	list containing the mapping of regimes over each edge (see details)
`models`	list giving model specification for the simulation of trait evolution. Supported models are c("OU", "BM", "WN"). The user can also specify if the assumptions of the model should be relaxed (see details)
`pars`	list giving parameters used for the simulation of trait evolution (see details). Name of parameters must be explecitly entered (see details)
`sampling`	vector of size 2 giving the min and max number of individual per species
`bounds`	list giving traits bounds
`var.f`	alternative function to model intraspecific variation of the form: function(n, pars) with n, number of individuals and pars, parameters of the model

Details

map : the list must be ordered in the same order than phy$edge. Each element represents an edge and contains a vector indicating the time spent under each regime in the branch. The name of the regimes must appear on the map models : trait evolution can be simulated using Ornstein-Uhlenbeck (OU), Brownian Motion (BM) or White Noise (WN) processes. Multiple regimes can be defined for both models and will apply on thetas: c("OU", "theta"), sigmas: c("OU", "sigma") or alphas: c("OU", "alpha") for OU and on sigmas only for WN: c("WN", "sigma") and BM: c("BM", "sigma"). While using the OU model, the user can also relax the stationarity of the root: c("OU", "root") and relax several assumptions at the same time c("OU", "root", "theta") pars : list containing parameters depending on the chosen model. Elements of that lists must be vectors of size 1 or n, with n = number of regimes in the map. Each element of pars must be named with the corresponding parameter abbreviation. Parameters used in the different models:

White Noise model (WN):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models

Brownian Motion model (BM):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models

Ornstein Uhlenbeck model (OU):

- root: root value. Only used if "root" is specified in models
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models
- theta: optimal value, n regimes if "theta" is specified in models
- alpha: strength of selection, n regimes if "alpha" is specified in models

Value

A list of length two containing a numeric matrix named "evo_traits" giving simulated traits representing intraspecific variation and a data.frame called "obs" containing species name in the fisrt column and individual observation of the trait in the second.

Author(s)

Theo Gaboriau

Examples

```
library(phytools)
phy <- pbtree(n = 50)
Q <- cbind(c(-.002, .002), c(.002, -.002))
phy <- sim.history(phy, Q = Q)
# MBM and VOU
jive_phy <- sim_jive(phy = phy, map = phy$maps)

# MWN + sigma and VOU + theta + root + alpha
jive_phy <- sim_jive(phy = phy, map = phy$maps,
   models = list(mean= c("WN", "sigma"), logvar = c("OU")),
   pars = list(mean = c(root = 0, sigma_sq1 = 0.1, sigma_sq2 = 0.5),
              logvar = c(root = 10, theta1 = 5, theta2 = 10,
                         sigma_sq = 0.1, alpha1 = 0.2, alpha2 = 0.8))
 )


# With a different model of intraspecific variation:
unif.f <- function(n, pars){
 runif(n, pars[1] - exp(pars[2])/2, pars[1] + exp(pars[2])/2)
}

unif_phy <- sim_jive(phy = phy, map = phy$maps, models = list(mid=c("BM"), logrange=c("OU")),
 pars = list(mid = c(root = 0, sigma_sq = 0.1),
             logrange = c(root = 2, theta = 1, sigma_sq = 0.1, alpha = 1)),
 var.f = unif.f)
```

---

sim_mte *Simulate MTE process*

Description

Generate random values of trait mean simulated under a MTE process along a phylogenetic tree

Usage

```
sim_mte(phy, map = NULL, model = "OU", pars = c(root = 2, theta = 1,
  sigma_sq = 0.1, alpha = 1), sampling = c(1, 7), bounds = c(-Inf,
  Inf))
```

Arguments

phy	Phylogenetic tree
map	list containing the mapping of regimes over each edge (see details).
model	model specification for the simulation of trait mean evolution. Supported models are c("OU", "BM", "WN")
pars	parameters used for the simulation of trait mean evolution (see details).
sampling	vector of size 2 giving the min and max number of individual per species
bounds	vector of size 2 giving the bounds of the mean

Details

map : the list must be ordered in the same order than phy$edge. Each element represents an edge and contains a vector indicating the time spent under each regime in the branch. The name of the regimes must appear on the map pars : list containing parameters depending on the chosen model. Elements of that lists must be vectors of size 1 or n, with n = number of regimes in the map. Each element of pars must be named with the corresponding parameter abbreviation. Parameters used in the different models:

White Noise model (WN):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models

Brownian Motion model (BM):

- root: root value
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models

Ornstein Uhlenbeck model (OU):

- root: root value. Only used if "root" is specified in models
- sigma_sq: evolutionary rate, n regimes if "sigma" is specified in models
- theta: optimal value, n regimes if "theta" is specified in models
- alpha: strength of selection, n regimes if "alpha" is specified in models

Value

returns a numeric vector giving the simulated mean value of the trait for each species of the tree.

Author(s)

Theo Gaboriau

Examples

```
library(phytools)
phy <- pbtree(n = 50)
Q <- cbind(c(-.002, .002), c(.002, -.002))
phy <- sim.history(phy, Q = Q)
# MBM and VOU
mte_phy <- sim_mte(phy, phy$maps)
```

xml_bite *Write xml file with model*

Description

Modifies a .xml file from beauti to include a model model in the Beast 2 analysis

Usage

```
xml_bite(model, xml, out = sprintf("%s_edited.xml", gsub(".xml", "",
  xml)))
```

Arguments

`model`	an object of class "model" (see details)
`xml`	name of the output file that will store the log of MCMC chain
`out`	where to write the edited xml

Details

This function takes a .xml file generated with Beauti and a model object generated with `make_jive` Only model objects that use models supported by the Beast implementation of model ("BM", c("BM", "sigma"), "WN", "OU", c("OU", "theta"), c("OU", "root"), c("OU", "root", "theta"))

Value

no return value, called for side effects

No return value: Modifies the .xml file given in xml in the user's filespace.

Author(s)

Theo Gaboriau

Index