Package 'treats'

November 11, 2024

Title Trees and Traits Simulations

Version 1.1

Date 2024-11-11

Description A modular package for simulating phylogenetic trees and species traits jointly. Trees can be simulated using modular birth-death parameters (e.g. changing starting parameters or algorithm rules). Traits can be simulated in any way designed by the user. The growth of the tree and the traits can influence each other through modifiers objects providing rules for affecting each other. Finally, events can be created to modify both the tree and the traits under specific conditions (Guillerme, 2024 <DOI:10.1111/2041-210X.14306>).

Depends R (>= 4.0.0), ape, dispRity, stats

Imports graphics, geiger, MASS, methods, utils, rgl,

Suggests testthat, knitr

License GPL-3

RoxygenNote 7.2.3

URL https://github.com/TGuillerme/treats

NeedsCompilation yes

Author Thomas Guillerme [aut, cre, cph]
(<https://orcid.org/0000-0003-4325-1275>)

Maintainer Thomas Guillerme <guillert@tcd.ie>

Repository CRAN

Date/Publication 2024-11-11 17:20:02 UTC

Contents

`crude.bd.est` *Crudely estimates extinction and speciation*

Description

Crudely estimates the extinction and speciaton rate of a tree based on `geiger::bd.km` and `geiger::bd.ms`

Usage

```
crude.bd.est(tree, method, ...)
```

Arguments

`tree`	a `"phylo"` object.
`method`	either `"count"` or `"estimate"`. See details.
`...`	any additional arguments to be passed to `geiger::bd.km` and `geiger::bd.ms`.

Details

This function calculates the extinction and speciation rates using two methods:

- `"estimate"` estimates the rates using the algorithm from `geiger::bd.km` and `geiger::bd.ms` based on the Magallon and Sanderson 2000 method. Note that this function provides more of a "guestimate" of extinction and speciation rates which can be especially wrong with low sampling (either missing fossil or living data). This can lead to estimating erroneous negative extinction rates.
- `"count"` This function calculates the extinction rate as the logged number of extinction events in the tree divided by the tree age (expressed in tree age units - e.g. million years). The speciation rate is calculated as the logged number of speciation events divided by the tree age. If the input tree has no `$root.time` element, the speciation and extinction rate are just the number of speciation and extinction events. Although *very crude* this method is slightly better at handling under sampled trees.

For more accurate model base approaches see for example `birthdeath` or `bd.ext`.

Value

A `"bd.params"` object to be fed to `treats`.

Author(s)

Thomas Guillerme

References

Magallon S and MJ Sanderson. 2000. Absolute diversification rates in angiosperm clades. Evolution 55:1762-1780.

See Also

`treats` `make.bd.params`

Examples

```
set.seed(1)
## Generating a random tree
my_tree <- rcoal(20)
## Estimate the number of speciations and extinctions events
crude.bd.est(my_tree, method = "estimate")

## Adding a root time
my_tree$root.time <- 5
## Count the number of speciations and extinctions
## per units of time
crude.bd.est(my_tree, method = "count")
```

---

dispRitreats *dispRity interface for treats objects*

Description

Pass a `treats` object to the `dispRity` function.

Usage

```
dispRitreats(data, ..., scale.trees = TRUE)
```

Arguments

data	an output from treats containing tree and traits data.
...	any other arguments to be passed to dispRity, chrono.subsets, custom.subsets, and boot.matrix.
scale.trees	logical, whether to scale the tree ages in all simulations (TRUE; default) or not (FALSE).

Details

This function applies the dispRity package pipeline to the treats output. If multiple simulations are input, the data is scaled for all the simulations.

The scale.trees option allows the trees to have the same depth and root age. This option is recommended if chrono.subsets options are called to make the output results comparable.

Common optional arguments for the following arguments include the following (refer the the specific function for the arguments details):

- custom.subsets: group for the list of elements to be attributed to specific groups;
- chrono.subsets: method for selecting the time binning or slicing method; time for the number of time bins/slices or their specific ages; model for the time slicing method; or inc.nodes for whether to include nodes or not in the time subsets;
- boot.matrix: bootstraps for the number of bootstrap replicates; rarefaction for the number of elements to include in each bootstrap replicate; or boot.type for the bootstrap algorithm;
- dispRity: metric for the disparity, dissimilarity or spatial occupancy metric to apply to the data; or dimensions for the number of dimensions to consider.

Value

Outputs a "dispRity" object that can be plotted, summarised or manipulated with the dispRity package.

Author(s)

Thomas Guillerme

See Also

treats dispRity chrono.subsets custom.subsets boot.matrix plot.dispRity summary.dispRity

Examples

```
## Simulate a random tree with a 10 dimensional Brownian Motion trait
my_treats <- treats(stop.rule = list("max.taxa" = 20),
                    traits    = make.traits(BM.process, n = 10),
                    bd.params = make.bd.params(speciation = 1))

## Calculating disparity as the sum of variances
disparity <- dispRitreats(my_treats, metric = c(sum, variances))
```

```
summary(disparity)

## Calculating disparity as the mean distance from the centroid of
## coordinates 42 (metric = c(mean, centroids), centroid = 42)
## using 100 bootstrap replicates (bootstrap = 100) and
## chrono.subsets (method = "continuous", model = "acctran", time = 5)
disparity <- dispRitreats(my_treats,
                          metric = c(mean, centroids), centroid = 42,
                          bootstraps = 100,
                          method = "continuous", model = "acctran", time = 5)
plot(disparity)

## Simulate 20 random trees with a 10 dimensional Brownian Motion trait
my_treats <- treats(stop.rule = list("max.taxa" = 20),
                    traits    = make.traits(BM.process, n = 10),
                    bd.params = make.bd.params(speciation = 1))

## Calculating disparity on all these trees as the sum of variance
## on 5 continuous proximity time subsets
disparity <- dispRitreats(my_treats, metric = c(sum, variances),
                          method = "continuous", model = "proximity", time = 5)
plot(disparity)
```

`drop.things` *Drop things from a treats object*

Description

Remove fossils or living species or non-bifurcating nodes (singles) from `treats` objects or `phylo` objects.

Usage

```
drop.things(treats, what)

drop.fossils(treats)

drop.livings(treats)

drop.singles(treats)
```

Arguments

`treats`	`treats` data.
`what`	what to drop. Can be `"fossils"`, `"livings"` or `"singles"` (non-bifurcating nodes).

Details

NOTE that dropping living or fossils species DOES NOT drop associated internal nodes and edge lengths. To drop both fossil/living taxa AND internal nodes, you can use for example: `drop.things(drop.things(my_data what = "fossils"), what = "singles")`.

Value

This function outputs either a `"phylo"` object if no traits where generated or a `treats` object that is a list of at least two elements: `$tree`, a `"phylo"` object and `$data`, a `"matrix"` of the trait values.

Author(s)

Thomas Guillerme

See Also

`treats` `plot.treats`

Examples

```
## A random tree with fossils and traits and internal nodes every 0.5 times
set.seed(3)
my_data <- treats(stop.rule = list(max.taxa = 20),
                  bd.params = list(speciation = 1, extinction = 1/3),
                  traits    = make.traits(), save.steps = 0.5)

## A tree with 20 tips and 54 nodes
my_data$tree
## And a dataset with 74 rows
dim(my_data$data)

## Removing the fossil species
drop.things(my_data, what = "fossils")$tree
dim(drop.fossils(my_data)$data)

## Removing the living species
drop.things(my_data, what = "livings")$tree
dim(drop.livings(my_data)$data)

## Removing the internal nodes
drop.things(my_data, what = "singles")$tree
dim(drop.singles(my_data)$data)

## Removing the internal nodes AND the fossils
drop.singles(drop.fossils(my_data))
```

drop.tip.treats *drop.tip.treats*

Description

Drop or keep tips from a "treats" object.

Usage

```
## S3 method for class 'treats'
drop.tip(phy, tip, ...)
```

Arguments

phy	an object of class "treats".
tip	a vector of mode numeric or character specifying the tips to delete or to keep.
...	any additional argument to be passed to drop.tip.phylo.

Details

This function allows to remove or keep tips from a "treats" object the same way as the drop.tip.phylo function.

Value

This function outputs either a "phylo" object if no traits where generated or a treats object that is a list of at least two elements: $tree, a "phylo" object and $data, a "matrix" of the trait values.

Author(s)

Thomas Guillerme

See Also

treats

Examples

```
## A treats object with one trait and 20 tips
my_treats <- treats(stop.rule = list(max.taxa = 20),
                    traits = make.traits())

## Removing five tips
drop.tip.treats(my_treats, tip = c("t1", "t2", "t3", "t4", "t5"))

## Keeping these five tips
drop.tip.treats(my_treats, tip = c("t1", "t2", "t3", "t4", "t5"))
```

events.conditions *events.conditions*

Description

Inbuilt conditions functions for helping designing events

Usage

```
events.condition(x, condition, ...)
```

Arguments

x	the variable to reach for satisfying a condition (see details)
condition	the logical function for triggering the condition (e.g. '<', '==', '!>', etc...).
...	any optional argument specific for that condition (see details)

Details

The following functions allow to design specific conditions for events:

- age.condition: a conditional function based on the time x. Typically this can be translated into "when time reaches the value x, trigger a condition" (see make.events). There is no optional argument for the function.
- taxa.condition: a conditional function based on the number of taxa x. Typically this can be translated into "when the number of taxa reaches the value x, trigger a condition" (see make.events). This function has one optional argument:
  - living, a logical argument whether to consider the number of taxa alive when the condition is checked (default: living = TRUE) or whether to consider all the taxa simulated so far (living = FALSE).
- trait.condition: a conditional function based on the value x of one or more traits. Typically this can be translated into "when a trait reaches a value x, trigger a condition" (see make.events). This function has three optional argument:
  - trait, one or more integer or numeric value designating the trait(s) to consider. By default, trait = 1, thus considering only the first trait to trigger the condition.
  - what, a function designating what to select from the trait values. By default what = max to select the maximal value of the trait when the condition is triggered (but you can use any function like min, mean, sd, etc. or provide your own function).
  - absolute, a logical designating to consider absolute trait values (TRUE) or not (default; FALSE).

More details about the events functions is explained in the treats manual: http://tguillerme.github.io/treats.

Value

This function outputs a "function" to be passed to make.events.

Author(s)

Thomas Guillerme

See Also

treats make.events events.modifications

Examples

```
## Generating a mass extinction
## 80% mass extinction at time 4
mass_extinction <- make.events(
                     target       = "taxa",
                     condition    = age.condition(4),
                     modification = random.extinction(0.8))

## Set the simulation parameters
stop.rule <- list(max.time = 5)
bd.params <- list(extinction = 0, speciation = 1)

## Run the simulations
set.seed(123)
results <- treats(bd.params = bd.params,
                  stop.rule = stop.rule,
                  events    = mass_extinction)
## Plot the results
plot(results, show.tip.label = FALSE)
axisPhylo()

## Changing the trait process
## The 95% upper quantile value of a distribution
upper.95 <- function(x) {
    return(quantile(x, prob = 0.95))
}
## Create an event to change the trait process
change_process <- make.events(
                   target       = "traits",
                   ## condition is triggered if(upper.95(x) > 3)
                   condition    = trait.condition(3, condition = `>`, what = upper.95),
                   modification = traits.update(process = OU.process))

## Set the simulation parameters
bd.params <- list(extinction = 0, speciation = 1)
stop.rule <- list(max.time = 6)
traits    <- make.traits()

## Run the simulations
set.seed(1)
no_change <- treats(bd.params = bd.params,
                   stop.rule = stop.rule,
                   traits    = traits)
set.seed(1)
```

```
process_change <- treats(bd.params = bd.params,
                         stop.rule = stop.rule,
                         traits    = traits,
                         events    = change_process)
## Plot the results
oldpar <- par(mfrow = c(1,2))
plot(no_change, ylim = c(-7, 7))
plot(process_change, ylim = c(-7, 7))
par(oldpar)
```

`events.modifications` *Events modifications*

Description

Inbuilt modifications functions for helping designing events

Usage

```
events.modification(x, ...)
```

Arguments

`x`	a numerical value to update.
`...`	any specific argument for the modification (see details).

Details

The following functions allow to design specific modifications for events:

- modifications for the target `"taxa"`
  - `random.extinction`: this function removes (makes extinct) a proportion of living taxa when the event is triggered. The proportion of taxa to remove can be changed with the argument `x`.
  - `trait.extinction`: this function removes (makes extinct) a number of living taxa based on their trait(s) values when the event is triggered. The trait value is specified with the argument `x`.This function has one optional argument:
    - condition to specify the condition in relation to that trait value (the default is `condition = `<`` meaning taxa with a trait value lower that `x` will go extinct).
    - trait to specify which trait will be affected (the default is `trait = 1`, meaning it will only consider the first trait).
- modifications for the target `"bd.params"`
  - `bd.params.update`: this function updates a `"bd.params"` object within the birth death process. It takes any unambiguous named argument to be passed to `make.bd.params`. For example, to update the speciation from any current rate to a new rate of 42, you can use `bd.params.update(speciation = 42)`.

- modifications for the target "traits"
  - traits.update: this function updates a "traits" object within the birth death process. It takes any unambiguous named argument to be passed to make.traits. For example, to update the trait process from the current one to an OU process, you can use traits.update(process = OU.process).
- modifications for the target "modifiers"
  - modifiers.update: this function updates a "modifiers" object within the birth death process. It takes any unambiguous named argument to be passed to make.modifiers. For example, to update the speciation from the current process to be dependent to trait values, you can use modifiers.update(speciation = speciation.trait).
- modifications for the target "founding"
  - founding.event: this function runs an independent birth-death process when the condition is met. This function takes any of the arguments normally passed to treats ("bd.params", "traits", "modifiers" and "events"). The stop.rule and other arguments are handled internally: namely the stop.rule argument is updated to match the time and number of taxa when the founding event is triggered. *Note that this can lead to the simulation stopping just before reaching the* max.taxa *or* max.living *stop rule.*

More details about the events functions is explained in the treats manual: http://tguillerme.github.io/treats.

Value

This function outputs a "function" to be passed to make.events.

Author(s)

Thomas Guillerme

See Also

treats make.events events.conditions

Examples

```
## Generating a mass extinction
## 80% mass extinction at time 4
mass_extinction <- make.events(
                      target       = "taxa",
                      condition    = age.condition(4),
                      modification = random.extinction(0.8))

## Set the simulation parameters
stop.rule <- list(max.time = 5)
bd.params <- list(extinction = 0, speciation = 1)

## Run the simulations
set.seed(123)
results <- treats(bd.params = bd.params,
```

```
                stop.rule = stop.rule,
                events    = mass_extinction)
## Plot the results
plot(results, show.tip.label = FALSE)
axisPhylo()

## Changing the trait process
## The 95% upper quantile value of a distribution
upper.95 <- function(x) {
    return(quantile(x, prob = 0.95))
}
## Create an event to change the trait process
change_process <- make.events(
                  target       = "traits",
                  ## condition is triggered if(upper.95(x) > 3)
                  condition    = trait.condition(3, condition = `>`, what = upper.95),
                  modification = traits.update(process = OU.process))

## Set the simulation parameters
bd.params <- list(extinction = 0, speciation = 1)
stop.rule <- list(max.time = 6)
traits    <- make.traits()

## Run the simulations
set.seed(1)
no_change <- treats(bd.params = bd.params,
                  stop.rule = stop.rule,
                  traits    = traits)
set.seed(1)
process_change <- treats(bd.params = bd.params,
                       stop.rule = stop.rule,
                       traits    = traits,
                       events    = change_process)
## Plot the results
oldpar <- par(mfrow = c(1,2))
plot(no_change, ylim = c(-7, 7))
plot(process_change, ylim = c(-7, 7))
par(oldpar)
```

`link.traits` *link.traits*

Description

Linking traits objects together to simulate simulate them sequentially.

Usage

```
link.traits(base.trait, next.trait, link.type, link.args, trait.name)
```

Arguments

base.trait	One or more "treats" "traits" object(s) to be considered first.
next.trait	One or more "treats" "traits" object(s) to be considered sequentially.
link.type	The type of link between the traits. Can be "conditional".
link.args	Optional arguments to interpret the link between the objects (based on the link.type).
trait.name	Optional, a character, the name the resulting trait.

Details

This function allows to link several traits together in the simulations. The current link types implemented are:

- "conditional": this allows to link the next.trait traits conditionally to the base.trait one. For example if base.trait is a discrete.process with two states 0 and 1 and next.trait is a list of two traits with two different processes OU.process and BM.process. The simulations generates a first trait using base.trait and then a second one using one of the two processes in next.trait depending on the results of base.trait. The link arguments link.args must be a list of logical functions to interpret x1, the results of the base.trait. For example, list(function(x1){x1 == 0}, function(x1){x1 == 1}) will generate a trait using the first next.trait if x1 is equal to 0 or using the second next.trait if x1 is equal to 1.

Value

This function outputs a treats object that is a named list of elements handled internally by the treats function.

Author(s)

Thomas Guillerme

See Also

treats trait.process make.traits

Examples

```
## Setting up a discrete trait
discrete_trait <- make.traits(discrete.process,
       process.args = list(transitions = matrix(c(3, 0.2, 0.05, 3), 2, 2)),
       trait.names  = "discrete")

## Setting up one dummy trait (always outputs 1)
always_one <- make.traits(process = function(x0 = 0, edge.length = 1) {return(1)},
                          trait.names = "one")
## Setting up a Brownian motion trait
BM_trait <- make.traits(trait.names = "BM")

## Setting a condition list to link all traits
```

```
## (if discrete trait is 0, simulate a BM trait)
## (if discrete trait is 1, simulate the always one trait)
conditions <- list("choose.BM"  = function(x1) {x1 == 0},
                   "choose.one" = function(x1) {x1 == 1})

## Creating the linked trait
conditional <- link.traits(base.trait = discrete_trait,
                           next.trait = list(BM_trait, always_one),
                           link.type  = "conditional",
                           link.args  = conditions)

## Simulating a tree using this trait
treats(stop.rule = list(max.living = 200),
       traits    = conditional)
```

`make.bd.params` *Make birth death parameters*

Description

Making bd.params objects for treats.

Usage

```
make.bd.params(
  speciation = NULL,
  extinction = NULL,
  joint = NULL,
  absolute = NULL,
  speciation.args = NULL,
  extinction.args = NULL,
  test = TRUE,
  update = NULL
)
```

Arguments

`speciation`	The speciation parameter. Can be a single `numeric` value, a `numeric` vector or a `function` (default is `1`).
`extinction`	The extinction parameter. Can be a single `numeric` value, a `numeric` vector or a `function` (default is `0`).
`joint`	Logical, whether to estimate both birth and death parameter jointly with speciation > extinction (`TRUE`) or not (`FALSE`; default).
`absolute`	Logical, whether always return absolute values (`TRUE`) or not (`FALSE`; default).
`speciation.args`	If `speciation` is a function, any additional arguments to passed to the `speciation` function.

`extinction.args`
: If `speciation` is a function, any additional arguments to passed to the `speciation` function.

`test`
: Logical whether to test if the bd.params object will work (default is `TRUE`).

`update`
: Optional, another previous `"treats"` `"bd.params"` object to update (see details).

Details

When using `update`, the provided arguments (to `make.bd.params`) will be the ones updated in the `"bd.params"` object.

Value

This function outputs a `treats` object that is a named list of elements handled internally by the `treats` function.

Author(s)

Thomas Guillerme

See Also

`treats`

Examples

```
## A default set of birth death parameters
make.bd.params()

## Speciation is randomly picked between 1, 10 and 100
## and extinction is always 2
make.bd.params(speciation = c(1,10,100), extinction = 2)

## Speciation is a normal distribution(with sd = 0.75)
## and extinction is a lognormal distribution always lower than
## speciation (joint). Both are always positive values (absolute)
my_bd_params <- make.bd.params(speciation = rnorm,
                               speciation.args = list(sd = 0.75),
                               extinction = rlnorm,
                               joint = TRUE,
                               absolute = TRUE)
my_bd_params

## Visualising the distributions
plot(my_bd_params)
```

make.events *make.events*

Description

Making events objects for treats

Usage

```
make.events(
  target,
  condition,
  modification,
  add,
  test = TRUE,
  event.name,
  replications = 0,
  additional.args
)
```

Arguments

target	What to modify, can be "taxa", "bd.params", "traits" or "modifiers" (see details).
condition	A function returning a logical to trigger the event (see details).
modification	A function bringing the modification to the event (see details).
add	Another "events" to object to add this event.
test	A logical, whether to test if the events object will work (default is TRUE)
event.name	Optional, a "character" string to name the event.
replications	A numeric or integer value for repeating the event (by default, the event is not repeated: replications = 0).
additional.args	Optional, a named list of additional arguments to be used in the event.

Details

target is a character to designate what will be affected by the event. It can be either "taxa", "bd.params", "traits" or "modifiers". This means that the condition and modification functions will target this specific part of the algorithm.

condition must be a function that returns a logical value and intakes any of the following arguments: bd.params, lineage, traits and time. See events.conditions for examples.

modification must be a function that intakes a first argument named "x" an returns any specific type of class that can be handled internally by treats. For example, if target = "bd.params" the modification function should typically return an updated bd.params object (see make.bd.params). See events.modifications for examples.

Value

This function outputs a `treats` object that is a named list of elements handled internally by the `treats` function.

Author(s)

Thomas Guillerme

See Also

`treats` `make.bd.params` `make.traits` `make.modifiers` `events.conditions` `events.modifications`

Examples

```
## Generating a mass extinction
## 80% mass extinction at time 4
mass_extinction <- make.events(
                     target       = "taxa",
                     condition    = age.condition(4),
                     modification = random.extinction(0.8))

## Set the simulation parameters
stop.rule <- list(max.time = 5)
bd.params <- list(extinction = 0, speciation = 1)

## Run the simulations
set.seed(123)
results <- treats(bd.params = bd.params,
                  stop.rule = stop.rule,
                  events    = mass_extinction)
## Plot the results
plot(results, show.tip.label = FALSE)
axisPhylo()

## Changing the trait process
## The 95% upper quantile value of a distribution
upper.95 <- function(x) {
    return(quantile(x, prob = 0.95))
}
## Create an event to change the trait process
change_process <- make.events(
                    target       = "traits",
                    ## condition is triggered if(upper.95(x) > 3)
                    condition    = trait.condition(3, condition = `>`, what = upper.95),
                    modification = traits.update(process = OU.process))

## Set the simulation parameters
bd.params <- list(extinction = 0, speciation = 1)
stop.rule <- list(max.time = 6)
traits    <- make.traits()

## Run the simulations
```

```
set.seed(1)
no_change <- treats(bd.params = bd.params,
                    stop.rule = stop.rule,
                    traits    = traits)
set.seed(1)
process_change <- treats(bd.params = bd.params,
                         stop.rule = stop.rule,
                         traits    = traits,
                         events    = change_process)
## Plot the results
oldpar <- par(mfrow = c(1,2))
plot(no_change, ylim = c(-7, 7))
plot(process_change, ylim = c(-7, 7))
par(oldpar)
```

`make.modifiers` *make.modifiers*

Description

Making modifiers objects for treats based on an ancestor's (parent) trait.

Usage

```
make.modifiers(
  branch.length = NULL,
  selection = NULL,
  speciation = NULL,
  condition = NULL,
  modify = NULL,
  add = NULL,
  update = NULL,
  test = TRUE
)
```

Arguments

`branch.length`	A function for the waiting time generating branch length (can be left empty for the defeault branch length function; see details).
`selection`	A function for selecting the lineage(s) affected by speciation (can be left empty for the default selection function; see details).
`speciation`	A function for triggering the speciation events (can be left empty for the default speciation function; see details).
`condition`	A function giving the condition on which to modify the output of `branch.length` or `speciation` (see details). If `NULL` the condition is always met.
`modify`	A function giving the rule of how to modify the output of `branch.length` or `speciation` (see details). If `NULL` no modification is used.

add	Whether to add this modifier to a "treats" "modifier" object.
update	Optional, another previous "treats" modifiers object to update (see details).
test	Logical whether to test if the modifiers object will work (default is TRUE).

Details

branch.length, selection and speciation must be a functions that intakes the following arguments: bd.params, lineage, trait.values, modify.fun. If left empty, any of these arguments is considered as NULL.

The default branch.length function is drawing a random number from the exponantial distribution with a rate equal to the current number of taxa multiplied by the speciation and extinction (rexp(1, n_taxa * (speciation + extinction))).

The default selection function is randomly drawing a single lineage among the ones present at the time of the speciation (sample(n_taxa, 1)).

The default speciation function is drawing a random number from a uniform distribution (0,1) and starts a speciation event if this random number is lower than the ration of speciation on speciation and extinction (runif(1) < (speciation/(speciation + extinction))). If the random number is greater, the lineage goes extinct.

condition must be a function with unambiguous input (the inputs listed about for branch.length and speciation) and must output a single logical value.

For example a conditional on the number of taxa:

condition = function(lineage) return(lineage$n < 1)

or a conditional on the trait values:

condition = function(trait.values, lineage) { parent.traits(trait.values, lineage) < mean(trait.values) }

modify must be a function with at least one input named x (which will be the branch length or the speciation trigger to value depending on the modifier) and must return a numeric value. For example a constant modification of the input:

modify = function(x) return(x * 2)

or a modifier depending on the number of taxa:

modify = function(x, lineage) return(x/lineage$n)

When using update, the provided arguments (to make.modifiers) will be the ones updated in the "modifiers" object. If the "modifiers" object contains multiple modifiers (branch.length, selection or speciation), only the called arguments will be updated (e.g. make.modifiers(update = previous_modifiers, speciation = new_speciation) will only update the speciation process).

More details about the modifiers functions is explained in the treats manual: http://tguillerme.github.io/treats.

Value

This function outputs a treats object that is a named list of elements handled internally by the treats function.

Author(s)

Thomas Guillerme

See Also

treats modifiers

Examples

```
## These functions should be fed to the make.modifiers function to create
## modifiers for treats objects. For example, the following sets specifies that
## the branch length should be generated using the branch.length.trait function
## the selection using the selection function and the speciation using the
## speciation.trait function:
my_modifiers <- make.modifiers(branch.length = branch.length.trait,
                               selection     = selection,
                               speciation    = speciation.trait)

## Creating a treats simulation using these modifiers
treats(stop.rule = list(max.taxa = 20),
     traits = make.traits(),
     modifiers = my_modifiers)
```

---

make.traits *make.traits*

Description

Making traits objects for treats

Usage

```
make.traits(
  process = BM.process,
  n = NULL,
  start = NULL,
  process.args = NULL,
  trait.names = NULL,
  add = NULL,
  update = NULL,
  test = TRUE,
  background
)
```

Arguments

process	The trait process(es) (default is BM.process).
n	Optional, the number of traits per process (default is 1).
start	Optional, the starting values for each traits (default is 0).
process.args	Optional, a named list of optional arguments for the trait process.
trait.names	Optional, the name(s) of the process(s).
add	Optional, another previous "treats" traits object to which to add the trait.
update	Optional, another previous "treats" traits object to update (see details).
test	Logical, whether to test if the traits object will work with treats (TRUE - default).
background	Optional, another "treats" "traits" object to simulate background trait evolution (see details).

Details

When using update, the provided arguments (to make.traits) will be the ones updated in the "traits" object. If the "traits" object contains multiple processes, you can specify which ones should be affected with the trait.names argument. Note that you cannot update the traits.names or the number of traits per processes (n) not use the add argument when updating a "traits" object.

If a background "traits" object is given, this object is then applied to all living edges at the same in the background while the main "traits" is computed.

More details about the "treats" "traits" objects is explained in the treats manual: http://tguillerme.github.io/treats.

Value

This function outputs a treats object that is a named list of elements handled internally by the treats function.

Author(s)

Thomas Guillerme

See Also

treats trait.process

Examples

```
## A simple Brownian motion trait (default)
make.traits()

## Two independent Brownian motion traits
make.traits(n = 2)

## Two different traits with different process
## (Brownian motion and Ornstein-Uhlenbeck)
```

```
make.traits(process = list(BM.process, OU.process))

## A multidimensional Brownian motion trait with correlation
## and different starting points
my_correlations <- matrix(1/3, ncol = 3, nrow = 3)
(my_traits <- make.traits(n = 3, start = c(0, 1, 3),
                          process.args = list(Sigma = my_correlations)))

## Adding a Ornstein-Uhlenbeck trait to the previous trait object
make.traits(process = OU.process, trait.names = "OU_trait",
            add = my_traits)
```

`make.treats` *Make a treats object*

Description

Combines a tree and some associated data into a treats object (e.g. for plotting)

Usage

```
make.treats(tree, data)
```

Arguments

`tree`	a phylogenetic tree.
`data`	a dataset of traits, either a `matrix` with column names or a named `vector`.

Value

This function outputs a `treats` object that is a list of at least two elements: `$tree`, a `"phylo"` object and `$data`, a `"matrix"` of the trait values.

Author(s)

Thomas Guillerme

See Also

`treats` `plot.treats`

Examples

```
## Creating a random tree
my_tree <- rtree(5)
## Adding node labels
my_tree$node.label <- letters[1:4]
## Creating a random dataset
my_data <- matrix(rnorm(9),
    dimnames = list(c(my_tree$tip.label, my_tree$node.label)))
## Creating the treats object
my_treats <- make.treats(tree = my_tree, data = my_data)
plot(my_treats)
```

`map.traits` *Maps a trait on a tree*

Description

Simulates one or more trait specified through a "traits" onto one or multiple trees.

Usage

```
map.traits(traits, tree, replicates)
```

Arguments

`traits`	A `"traits"` object (see `make.traits`).
`tree`	A `"phylo"` or `"multiPhylo"` object.
`replicates`	Optional, a number of replicated traits to map.

Details

This function simulates the trait(s) on the tree using the tree's branch length.

Value

A `"treats"` object containing the tree and the traits.

Examples

```
## Simulating a random tree with branch length
my_tree <- rtree(20)

## Creating three different traits objects:
## A Brownian Motion
bm_process <- make.traits(process = BM.process)
## An Ornstein-Uhlenbeck process
ou_process <- make.traits(process = OU.process)
```

```
## No process (just randomly drawing values from a normal distribution)
no_process <- make.traits(process = no.process)

## Mapping the three traits on the phylogeny
bm_traits <- map.traits(bm_process, my_tree)
ou_traits <- map.traits(ou_process, my_tree)
no_traits <- map.traits(no_process, my_tree)

## Plotting the topology and the different traits
oldpar <- par(mfrow = c(2,2))
plot(my_tree, main = "Base topology")
plot(bm_traits, main = "Mapped BM")
plot(ou_traits, main = "Mapped OU")
plot(no_traits, main = "Mapped normal trait")
par(oldpar)
```

`modifiers` *Modifiers*

Description

Different modifiers for the birth death process implemented in treats.

Usage

```
modifiers(bd.params = NULL, lineage = NULL, trait.values = NULL,
                    modify.fun = NULL)
```

Arguments

`bd.params`	A named list of birth death parameters (see details).
`lineage`	A named list containing the lineage data (see details).
`trait.values`	A matrix containing the trait values (see details).
`modify.fun`	A list of internals functions that can modified by `events` (see details).

Details

`bd.params` can be either a named list of parameter values (e.g. `list("extinction" = 0, "speciation" = 1)`) but it is typically handled internally from a `"treats" "bd.params"` object.

`modifiers` are functions passed to the birth death process in `treats` to either generate the branch length (named `branch.length` and similar) or to decide whether to speciate or go extinct (named `speciation` and similar).

For user defined functions, the `modifiers` *must* have at least the arguments described above. For safety, we suggest setting these arguments to `NULL`.

The pre-build `modifiers` in the `treats` package are (so far):

- `branch.length` the simple branch length generator that randomly gets a numeric value drawn from the exponential distribution (`rexp`) with a rate equal to the number of taxa (`lineage$n * bd.params$speciation + bd.params$extinction`).
- `branch.length.trait` a modification of the `branch.length` `modifier` where the resulting branch length is changed by `modify.fun$modify` if the parent trait(s) meet the condition `modify.fun$condition`.
- `selection` a function returning a randomly sampled integer among the number of taxa available.
- `speciation` a function returning `TRUE` (speciation) if a random uniform number (`runif`) is smaller than the ratio of speciation by speciation and extinction (`bd.params$speciation / (bd.params$speciation) + bd.params$extinction`). If it's bigger, the function returns `FALSE` (exinction).
- `speciation.trait` a modification of the `speciation` `modifier` where the random uniform number is changed by `modify.fun$modify` if the parent trait(s) meet the condition `modify.fun$condition`.

More details about the `modifiers` functions is explained in the `treats` manual: http://tguillerme.github.io/treats.

Value

These functions returns either `"numeric"` or `"logical"` values to be passed to `make.modifiers` and `treats`.

Author(s)

Thomas Guillerme

See Also

`treats` `make.modifiers`

Examples

```
## These functions should be fed to the make.modifiers function to create
## modifiers for treats objects. For example, the following sets specifies that
## the branch length should be generated using the branch.length.trait function
## the selection using the selection function and the speciation using the
## speciation.trait function:
my_modifiers <- make.modifiers(branch.length = branch.length.trait,
                               selection     = selection,
                               speciation    = speciation.trait)

## Creating a treats simulation using these modifiers
treats(stop.rule = list(max.taxa = 20),
     traits = make.traits(),
     modifiers = my_modifiers)
```

parent.traits *Get parent traits*

Description

An internal utility function for `modifiers`, `traits` or `events` to access the value(s) of the parent traits in the `treats` algorithm

Usage

```
parent.traits(trait.values, lineage, current = TRUE)
```

Arguments

`trait.values`	The internal table of trait values
`lineage`	The internal lineage data list
`current`	Whether to consider only the current lineage (`TRUE` - default) or all the living lineages (`FALSE`).

Details

This function is designed to be used internally in `treats` to help `modifiers`, `traits` or `events` objects to access the parent traits of the lineages simulated through the internal birth death algorithm.

Value

Returns one or more `"numeric"` values.

Author(s)

Thomas Guillerme

See Also

`treats` `make.modifiers`

Examples

```
## Speciation event is more likely if lineage's ancestor is further away from the mean trait value
distance.modify <- function(x, trait.values, lineage) {
     ## Distance to the parent's trait
     parent_trait_val <- parent.traits(trait.values, lineage)[1]
     mean_trait_val <- mean(trait.values[, 1])
     distance <- abs(parent_trait_val - mean_trait_val)
     ## Scales x with the distance
     return(x + x * distance)
}
```

```
## Make a distance modifier (speciation more likely with distance)
distance.speciation <- make.modifiers(speciation = speciation,
                                      modify = distance.modify)
```

`plot.treats` *Plot treats objects*

Description

Plotting treats objects (either a simulated tree and trait(s) or a process for traits objects)

Usage

```
## S3 method for class 'treats'
plot(
  x,
  col,
  ...,
  trait = 1,
  edges = "grey",
  tips.nodes = NULL,
  use.3D = FALSE,
  simulations = 20,
  cent.tend = mean,
  quantiles = c(95, 50),
  legend = FALSE,
  transparency,
  add = FALSE
)
```

Arguments

`x`	`treats` data.
`col`	Optional, a `vector` of colours that can be named or a `function` (see details).
`...`	Any additional options to be passed to plot functions (from `graphics` or `rgl` if `use.3D = TRUE`).
`trait`	which trait to plot (default is `1`; see details).
`edges`	either a colour name to attribute to the edges or `NULL` to not display the edges (default is `"grey"`).
`tips.nodes`	optional, a colour to circle tips and nodes (only used if `use.3D = FALSE`). By default `tips.nodes = NULL`.
`use.3D`	logical, whether to use a 3D plot or not (default is `FALSE`; see details).
`simulations`	if the input is a `treats` `traits` or `bd.params` object, how many replicates to run (default is `50`).

cent.tend	if the input is a treats traits, which central tendency to plot (default is mean).
quantiles	if the input is a treats traits, which quantiles to plot (default are c(95, 50))).
legend	logical, whether to display the legend in 2D plots (TRUE) or not (FALSE; default)
transparency	Optional, a transparency factor (1 = not transparent, 0 = invisible). If left empty, and multiple plots are called, the transparency is set to 1 / number of plots + 0.1.
add	logical, whether to add to a previous plot.

Details

The col option can be either:

- a vector of colours to be applied to "treats" "traits" objects (for respectively the median, 50
- a vector of colours to be applied to "treats" objects for the colours of different elements of the plot. This vector is applied to all the elements in the tree using the order in tree$tip.label and tree$node.label.
- an unambiguous named vector for colouring each specific elements. These can be any of the following (with default colours) col = c("nodes" = "orange", "fossils" = "lightblue", "livings" = "blue") or "tips" to designate both livings and fossils and "singletons" to designate non-bifurcating nodes.
- a function from which to sample the colours to match the time gradient for each element.

The trait option can intake from 1 to 3 traits (if use.3D = TRUE). If two traits are given (e.g. c(1, 2)), the default plots a correlation plot between both traits (same for 3 traits if use.3D = TRUE).

The use.3D option uses the rgl library to create a 3D plot. The plot displays either a time on the Z axis with two traits on the X and Y axis (if two traits are requested via trait) or three traits on the X Y and Z (if three traits a requested via trait).

Value

No return value, plot x's content.

Author(s)

Thomas Guillerme

See Also

treats

Examples

```
## Specifying a trait process
my_trait <- make.traits()
## Plotting a trait process
plot(my_trait, main = "A Brownian Motion")

## Simulating a tree with ten taxa
```

```
my_tree <- treats(stop.rule = list(max.taxa = 10))
## Plotting a simple birth death tree (using ape::plot.phylo)
plot(my_tree, main = "A pure birth tree")

## Simulating a tree with traits
my_data <- treats(stop.rule = list(max.taxa = 100),
                  traits    = my_trait)
## Plotting the tree and traits
plot(my_data)

## Specifying a 3D trait process
my_3D_trait <- make.traits(n = 3)
## Simulating a birth death tree with that trait
my_data <- treats(bd.params = list(extinction = 0.2),
                  stop.rule = list(max.living = 50),
                  traits    = my_3D_trait)

## Plotting the second trait and the tree (default)
## The colours are purple for nodes and blue for tips
## with a black circle for highlighting the tips
plot(my_data, trait = 2,
     col = c("nodes" = "purple", "tips" = "blue"),
     edges = "pink", tips.nodes = "black")

## Plotting the first and third trait correlation
## The colours are a heat map based on the elements age
plot(my_data, trait = c(1,3), col = terrain.colors,
     edges = "grey", tips.nodes = "black")

## Plotting the first and third trait correlation in 3D
plot(my_data, trait = c(1,3), col = rainbow,
     edges = "grey", tips.nodes = "black", use.3D = TRUE)
#rglwidget() # to display the plot with non-default OpenRGL

## Plotting all traits in 3D (without branch lengths)
plot(my_data, trait = c(1:3), col = heat.colors,
     edges = NULL, tips.nodes = "black", use.3D = TRUE)
#rglwidget() # to display the plot with non-default OpenRGL
```

`print.treats` *Prints a* `treats` *object.*

Description

Summarises the content of a `treats` object.

Usage

```
## S3 method for class 'treats'
print(x, all = FALSE, ...)
```

Arguments

x	A treats object.
all	logical; whether to display the entire object (TRUE) or just summarise its contents (FALSE - default).
...	further arguments to be passed to print or to print.treats.

Value

No return value, summarises x's content.

Author(s)

Thomas Guillerme

See Also

treats

Examples

```
## A treats birth-death parameters object
make.bd.params()
## A treats traits object
make.traits()
## A treats modifiers object
make.modifiers()
## A treats object
treats(stop.rule = list(max.taxa = 10), traits = make.traits())
```

---

trait.process *Trait processes*

Description

Different trait processes implemented in treats.

Usage

```
trait.process(x0, edge.length, ...)
```

Arguments

x0	The previous state. This can be a single value (unidimensional process) or more (multidimensional processes).
edge.length	The branch length (default must be 1). This is always a single value.
...	Any optional argument for the specific process (see details).

Details

The different trait processes implemented in treats are:

- BM.process A Brownian motion process (uni or multidimensional). This function is based on `mvrnorm`. This process can take following optional arguments:
  - `Sigma` a positive-definite symmetric matrix specifying the covariance matrix of the variables (default is `diag(length(x0))`).
  - `...` any named additional argument to be passed to `mvrnorm`.
- discrete.process This process can take following optional arguments:
  - `transitions` a positive-definite squared transition matrix. If left missing, a 2 states equal rates matrix is used.

  Note that for this process, 0 corresponds to state 1, 1 corresponds to state 2, etc... The current version of this process does not allow other discrete traits notation (but future versions will!).
- OU.process A Ornstein-Uhlenbeck process (uni or multidimensional). This function is based on `mvrnorm`. This process can take following optional arguments:
  - `Sigma` the traits variance/covariance (default is `diag(length(x0))`).
  - `alpha` the alpha parameter (default = is `1`).
  - `optimum` the theta parameter (default = is `0`).
  - `...` any named additional argument to be passed to `mvrnorm`.
- no.process An non-process unidimensional function. This function generates a trait value not depending on the branch length nor the previous state This process can take following optional arguments:
  - `fun` a random number function (default is `rnorm`).
  - `...` any named additional argument to be passed to `fun`.
- multi.peak.process A Ornstein-Uhlenbeck process (uni or multidimensional) with multiple optimal values. This function is based on `mvrnorm`. This process can take following optional arguments:
  - `Sigma` the traits variance/covariance (default is `diag(length(x0))`).
  - `alpha` the alpha parameter (default = is `1`).
  - `peaks` the multiple optimal values to be attracted to (default = is `0`). This can be a `numeric` vector to be applied to all the values of `x0` or a `list` of the same length as `x0` for different multiple optimums for each `x0`.
  - `...` any named additional argument to be passed to `mvrnorm`.
- repulsion.process An unidimensional Brownian Motion process that generates a trait value not overlapping with the other living taxa ancestral values. This function is based on `rnorm`. This process can take following optional arguments:
  - `sd` the normal distribution standard deviation.
  - `repulsion` the minimal distance requested between trait values.
  - `max.try` the maximum number of values to draw (if the repulsion value is to hard to achieve).
  - `trait.values` LEAVE AS `NULL` (it designates the trait value table from the birth death process and is handled internally by `treats`).

- lineage LEAVE AS NULL (it designates the lineage object from the birth death process and is handled internally by treats).
- trait LEAVE AS NULL (it which trait to use and is analysed an is handled internally by treats).

More details about the trait.process functions is explained in the treats manual: http://tguillerme.github.io/treats.

Value

Returns one or more "numeric" value(s).

Author(s)

Thomas Guillerme

See Also

treats make.traits

Examples

```
## NOTE: You can visualise most process by making them
## into a "treats" "traits" object using make.traits():

## The Brownian motion process
BM.process(x0 = 0)
plot(make.traits(process = BM.process))
## A covariance matrix between 3 traits
varcovar_matrix <- matrix(c(1/3,1/3,1/3,1/3,2/3,0,1/3,0,2/3), ncol = 3)
BM.process(x0 = c(0,0,0), Sigma = varcovar_matrix)

## The Ornstein-Uhlenbeck process
OU.process(x0 = 0)
plot(make.traits(process = OU.process))

## No process
no.process()
plot(make.traits(process = no.process))

## Multi peaks with peaks at the values 1, 5 and 10
multi.peak.process(peaks = c(1, 5, 10))
plot(make.traits(multi.peak.process, process.args = list(peaks = c(1, 5, 10))))

## Repulsion process
repulsion.process(x0 = 0, repulsion = 1)
plot(make.traits(repulsion.process, process.args = list(repulsion = 5)))

## Discrete trait process
## Generating a stepwise transition matrix for 3 states (with an overal random transition rate)
stepwise_matrix <- transition.matrix(type = "stepwise", states = 3)
## Generatin and plotting the the trait
```

```
plot(make.traits(discrete.process, process.args = list(transitions = stepwise_matrix)))

##
```

`transition.matrix` *Makes a transition matrix*

Description

Utility function for generating discrete characters evolution transition matrices.

Usage

```
transition.matrix(type, states, rates = runif, self = TRUE, ...)
```

Arguments

`type`	the type of transition matrix, either "equal rates", "stepwise", "symmetric", or "all rates different". See details.
`states`	the number of states.
`rates`	either a fixed value for a rate to attribute to each possible transitions or a `function` to generate the rates (default is `runif`). See details.
`self`	logical, whether to allow reverting states (i.e. transition rates from state A to the same state A; `TRUE`; default) or not (`FALSE`).
`...`	if `rates` is a function, any optional arguments to be passed to it.

Details

The following transition rate matrices are currently implemented:

- "equal rates" (or "ER") where all transitions are equal (including no transition if `self = TRUE`).
- "stepwise" (or "Dollo") transitions are allowed only in a step wise way (e.g. state 1 to 2 and 2 to 3 are allowed but not 1 to 3).
- "symmetric" ("SYM") where transitions between states are all different but not directional (e.g. the change of state 1 to 2 is equal to 2 to 1). If `self = TRUE`, the non transitions (e.g. from state 1 to 1) are equal.
- "all rates different" (or "ARD") where all transitions are different. Note that if rates is a give value (rather than a function), then all rates are actually equal.

If `rates` is a function that generates negative values or a negative value, the output transition matrix always returns absolute values.

Value

Returns a squared `"matrix"`.

Author(s)

Thomas Guillerme

See Also

make.traits discrete.process

Examples

```
## A two states equal rates matrix with a rate of 1
## and no stationary rates (no probability of staying in the same state)
transition.matrix(type = "equal rates", states = 2, rates = 1, self = FALSE)

## Two different 6 states stepwise matrix with a random absolute normal rate
transition.matrix(type = "stepwise", states = 6, rates = rnorm)
transition.matrix(type = "stepwise", states = 6, rates = rnorm)
```

---

treats *Diversity and disparity simulator*

Description

Simulating phylogenetic trees and traits. See full manual here: https://github.com/TGuillerme/treats

Usage

```
treats(
  stop.rule,
  bd.params,
  traits = NULL,
  modifiers = NULL,
  events = NULL,
  save.steps = NULL,
  null.error = FALSE,
  replicates,
  verbose = TRUE
)
```

Arguments

stop.rule	The rules on when to stop the simulation (see details).
bd.params	A "bd.params" object or a named list of parameters for the birth-death process (see details or make.bd.params).
traits	A "traits" object (see make.traits).
modifiers	A "modifiers" object (see make.modifiers).

events	A "events" object (see make.events).
save.steps	Optional, "numeric" value to save the simulations at specific internal points (this can slow down the algorithm significantly for large trees).
null.error	Logical, whether to return an error when the birth-death parameters fails to build a tree (FALSE; default and highly recommended) or whether to return NULL (TRUE). Can also be set to an integer value for the numbers of trials (see details).
replicates	Optional, the number of replicates for the simulation.
verbose	Logical, whether to be verbose (TRUE; default) or not (FALSE).

Details

stop.rule: The rule(s) for when to stop the simulation. When multiple rules are given, the simulation stops when any rule is broken. The allowed rules are:

- max.taxa The maximum number of taxa (including extinct ones).
- max.living The maximum number of living taxa (i.e. non extinct).
- max.time The maximum amount of phylogenetic (in arbitrary units).

bd.params: This can be either a "treats" "bd.params" object (see make.bd.params) or a list of named parameters. The allowed parameters are:

- speciation The speciation parameter value.
- extinction The extinction parameter value.

By default, this parameter is set to bd.params = list(speciation = 1)

If null.error is set to a numeric value, the function will run multiple times until a correct tree is generated. Using this option can greatly increase computational time!

Value

This function outputs either a "phylo" object if no traits where generated or a treats object that is a list of at least two elements: $tree, a "phylo" object and $data, a "matrix" of the trait values.

Author(s)

Thomas Guillerme

See Also

plot.treats make.traits make.modifiers make.events

Examples

```
## Setting pure birth tree (no extinction) parameters
my_bd_params <- list(speciation = 1)
## Setting a stopping rule: stop when reaching 10 taxa.
my_stop_rule <- list(max.taxa = 10)

## Run a birth tree without traits
```

```
a_tree <- treats(bd.params = my_bd_params,
                 stop.rule = my_stop_rule)
## Plot the results
plot(a_tree)

## Add an extinction parameter
my_bd_params$extinction <- 1/3

## Add a simple trait simulation (default Brownian motion)
my_trait <- make.traits()

## Run a birth-death tree with traits simulation
treats(bd.params = my_bd_params,
     stop.rule = my_stop_rule,
     traits    = my_trait)

## Simulating a tree using modifiers
## Making a modifier to make speciation trait dependent
my_modifiers <- make.modifiers(branch.length = branch.length.trait,
                               selection     = selection,
                               speciation    = speciation.trait)

## Simulating the tree
treats(stop.rule = list(max.taxa = 20),
     traits = make.traits(),
     modifiers = my_modifiers)

## Run a birth death tree with an event
## 80% mass extinction at time 4
mass_extinction <- make.events(
                      target       = "taxa",
                      condition    = age.condition(4),
                      modification = random.extinction(0.8))

## Set the simulation parameters
stop.rule <- list(max.time = 5)
bd.params <- list(extinction = 0, speciation = 1)

## Run the simulations
set.seed(123)
results <- treats(bd.params = bd.params,
                  stop.rule = stop.rule,
                  events    = mass_extinction)
## Plot the results
plot(results, show.tip.label = FALSE)
axisPhylo()
```

Index