Package 'paleobuddy'

February 5, 2025

Title Simulating Diversification Dynamics

Version 1.1.0

Description Simulation of species diversification, fossil records, and phylogenies. While the literature on species birth-death simulators is extensive, including important software like 'paleotree' and 'APE', we concluded there were interesting gaps to be filled regarding possible diversification scenarios. Here we strove for flexibility over focus, implementing a large array of regimens for users to experiment with and combine. In this way, 'paleobuddy' can be used in complement to other simulators as a flexible jack of all trades, or, in the case of scenarios implemented only here, can allow for robust and easy simulations for novel situations. Environmental data modified from that in 'RPANDA': Morlon H. et al (2016) <doi:10.1111/2041-210X.12526>.

URL https://github.com/brpetrucci/paleobuddy

Suggests ape, fitdistrplus, knitr, rmarkdown

Imports methods

License GPL-3

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 7.3.2

BugReports https://github.com/brpetrucci/paleobuddy/issues

NeedsCompilation no

Author Bruno do Rosario Petrucci [aut, cre]
(<https://orcid.org/0000-0001-6334-8483>),
Matheus Januario [aut] (<https://orcid.org/0000-0002-6480-7095>),
Tiago Quental [aut] (<https://orcid.org/0000-0002-4832-9468>)

Maintainer Bruno do Rosario Petrucci <petrucci@iastate.edu>

Repository CRAN

Date/Publication 2025-02-05 18:00:07 UTC

Contents

`bd.sim` *General rate Birth-Death simulation*

Description

Simulates a species birth-death process with general rates for any number of starting species. Allows for the speciation/extinction rate to be (1) a constant, (2) a function of time, (3) a function of time and;or an environmental variable, or (4) a vector of numbers representing a step function. Allows for constraining results on the number of species at the end of the simulation, either total or extant. The function can also take an optional shape argument to generate age-dependence on speciation and/or extinction, assuming a Weibull distribution as a model of age-dependence. Returns a `sim` object (see `?sim`). It may return true extinction times or simply information on whether species lived after the maximum simulation time, depending on simulation settings.

Usage

```
bd.sim(
  n0,
  lambda,
  mu,
  tMax = Inf,
  N = Inf,
  lShape = NULL,
  mShape = NULL,
  envL = NULL,
```

```
  envM = NULL,
  lShifts = NULL,
  mShifts = NULL,
  nFinal = c(0, Inf),
  nExtant = c(0, Inf),
  trueExt = FALSE
)
```

Arguments

n0	Initial number of species. Usually 1, in which case the simulation will describe the full diversification of a monophyletic lineage. Note that when lambda is less than or equal to mu, many simulations will go extinct before speciating even once. One way of generating large sample sizes in this case is to increase n0, which will simulate the diversification of a paraphyletic group.
lambda	Speciation rate (events per species per million years) over time. It can be a numeric describing a constant rate, a function(t) describing the variation in speciation over time t, a function(t, env) describing the variation in speciation over time following both time AND an environmental variable (please see envL for details) or a vector containing rates that correspond to each rate between speciation rate shift times times (please see lShifts). Note that lambda should always be greater than or equal to zero.
mu	Similar to lambda, but for the extinction rate. Note: rates should be considered as running from 0 to tMax, as the simulation runs in that direction even though the function inverts speciation and extinction times before returning.
tMax	Ending time of simulation, in million years after the clade origin. Any species still living after tMax is considered extant, and any species that would be generated after tMax is not present in the return.
N	Number of species at the end of the simulation. End of the simulation will be set for one of the times where N species are alive, chosen from all the times there were N species alive weighted by how long the simulation was in that situation. Exactly one of tMax and N must be non-Inf. Note that if N is the chosen condition, mu cannot be 0, since paelobuddy's current algorithm would mean only species 1 would have children. Future features will hopefully remove this limitation.
lShape	Shape parameter defining the degree of age-dependency in speciation rate. This will be equal to the shape parameter in a Weibull distribution: as a species' longevity increases (negative age-dependency). When larger than one, speciation rate will increase as a species' longevity increases (positive age-dependency). It may be a function of time, but see note below for caveats therein. Default is NULL, equivalent to an age-independent process. For lShape != NULL (including when equal to one), lambda will be considered a scale (= 1/rate), and rexp.var will draw a Weibull distribution instead of an exponential. This means Weibull(rate, 1) = Exponential(1/rate). Note that even when lShape != NULL, lambda may still be time-dependent.
mShape	Similar to lShape, but for the extinction rate.

Note: Simulations with time-varying shape behave within theoretical expectations for most cases, but if shape is lower than 1 and varies too much (e.g. `0.5 + 0.5*t`), it can be biased for higher waiting times due to computational error. A degree of time dependence of the order of 0.01 events/my^2 are advisable. It might, although rarely, exhibit a small bias when using shape functions with abrupt time variations. In both cases, error is still quite low for the purposes of the package.

Note: Shape must be greater than 0. We arbitrarily chose 0.01 as the minimum accepted value, so if shape is under 0.01 for any reasonable time in the simulation, it returns an error.

`envL` A `data.frame` describing a time series that represents the variation of an environmental variable (e.g. CO2, temperature, available niches, etc) with time. The first column of this `data.frame` must be time, and the second column must be the values of the variable. This will be internally passed to the `make.rate` function, to create a speciation rate variation in time following the interaction between the environmental variable and the function. Note `paleobuddy` has two environmental data frames, `temp` and `co2`. One can check `RPANDA` for more examples, or use their own time series of a variable of interest

`envM` Similar to `envL`, but for the extinction rate.

`lShifts` Vector of rate shifts. First element must be the starting time for the simulation (`0` or `tMax`). It must have the same length as `lambda`. `c(0, x, tMax)` is equivalent to `c(tMax, tMax - x, 0)` for the purposes of `make.rate`.

`mShifts` Similar to `mShifts`, but for the extinction rate.

`nFinal` A `vector` of length `2`, indicating an interval of acceptable number of species at the end of the simulation. Default value is `c(0, Inf)`, so that any number of species (including zero, the extinction of the whole clade) is accepted. If different from default value, simulation will restart until the number of total species at `tMax` is in the `nFinal` interval. Note that `nFinal` must be a sensible vector. The function will error if its maximum is lower than `1`, or if its length is not `2`.

`nExtant` A `vector` of length `2`, indicating an interval of acceptable number of extant species at the end of the simulation. Equal to `nFinal` in every respect except for that.

Note: The function returns `NA` if it runs for more than `100000` iterations without fulfilling the requirements of `nFinal` and `nExtant`.

Note: Using values other than the default for `nFinal` and `nExtant` will condition simulation results.

`trueExt` A `logical` indicating whether the function should return true or truncated extinction times. When `TRUE`, time of extinction of extant species will be the true time, otherwise it will be `NA` if a species is alive at the end of the simulation.

Note: This is interesting to use to test age-dependent extinction. Age-dependent speciation would require all speciation times (including the ones after extinction) to be recorded, so we do not attempt to add an option to account for that. Since age-dependent extinction and speciation use the same underlying process, however, if one is tested to satisfaction the other should also be in expectations.

Details

Please note while time runs from `0` to `tMax` in the simulation, it returns speciation/extinction times as `tMax` (origin of the group) to `0` (the "present" and end of simulation), so as to conform to other packages in the literature.

Value

A `sim` object, containing extinction times, speciation times, parent, and status information for each species in the simulation. See `?sim`.

Author(s)

Bruno do Rosario Petrucci.

References

Stadler T. 2011. Simulating Trees with a Fixed Number of Extant Species. Systematic Biology. 60(5):676-684.

Examples

```
# we will showcase here some of the possible scenarios for diversification,
# touching on all the kinds of rates

###
# consider first the simplest regimen, constant speciation and extinction

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation
lambda <- 0.11

# extinction
mu <- 0.08

# set a seed
set.seed(1)

# run the simulation, making sure we have more than 1 species in the end
sim <- bd.sim(n0, lambda, mu, tMax, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}
```

```
###
# if we want, we can simulate up to a number of species instead

# initial number of species
n0 <- 1

# maximum simulation time
N <- 10

# speciation
lambda <- 0.11

# extinction
mu <- 0.08

# set a seed
set.seed(1)

# run the simulation, making sure we have more than 1 species in the end
sim <- bd.sim(n0, lambda, mu, N = N)

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# now let us complicate speciation more, maybe a linear function

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# make a vector for time
time <- seq(0, tMax, 0.1)

# speciation rate
lambda <- function(t) {
  return(0.05 + 0.005*t)
}

# extinction rate
mu <- 0.1

# set a seed
set.seed(4)

# run the simulation, making sure we have more than 1 species in the end
sim <- bd.sim(n0, lambda, mu, tMax, nFinal = c(2, Inf))
```

```
# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  # full phylogeny
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# what if we want mu to be a step function?

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation
lambda <- function(t) {
  return(0.02 + 0.005*t)
}

# vector of extinction rates
mList <- c(0.09, 0.08, 0.1)

# vector of shift times. Note mShifts could be c(40, 20, 5) for identical
# results
mShifts <- c(0, 20, 35)

# let us take a look at how make.rate will make it a step function
mu <- make.rate(mList, tMax = tMax, rateShifts = mShifts)

# and plot it
plot(seq(0, tMax, 0.1), rev(mu(seq(0, tMax, 0.1))), type = 'l',
     main = "Extintion rate as a step function", xlab = "Time (Mya)",
     ylab = "Rate (events/species/My)", xlim = c(tMax, 0))

# looking good, we will keep everything else the same

# a different way to define the same extinction function
mu <- function(t) {
  ifelse(t < 20, 0.09,
         ifelse(t < 35, 0.08, 0.1))
}

# set seed
set.seed(2)

# run the simulation
sim <- bd.sim(n0, lambda, mu, tMax, nFinal = c(2, Inf))
# we could instead have used mList and mShifts

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
```

```
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# we can also supply a shape parameter to try age-dependent rates

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation - note that since this is a Weibull scale,
# the unites are my/events/lineage, not events/lineage/my
lambda <- 10

# speciation shape
lShape <- 2

# extinction
mu <- 0.08

# set seed
set.seed(4)

# run the simulation - note the message saying lambda is a scale
sim <- bd.sim(n0, lambda, mu, tMax, lShape = lShape, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# scale can be a time-varying function

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation - note that since this is a Weibull scale,
# the unites are my/events/lineage, not events/lineage/my
lambda <- function(t) {
  return(2 + 0.25*t)
}

# speciation shape
lShape <- 2
```

```
# extinction
mu <- 0.2

# set seed
set.seed(1)

# run the simulation - note the message saying lambda is a scale
sim <- bd.sim(n0, lambda, mu, tMax, lShape = lShape, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# and shape can also vary with time

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation - note that since this is a Weibull scale,
# the unites are my/events/lineage, not events/lineage/my
lambda <- function(t) {
  return(2 + 0.25*t)
}

# speciation shape
lShape <- function(t) {
  return(1 + 0.02*t)
}

# extinction
mu <- 0.2

# set seed
set.seed(4)

# run the simulation - note the message saying lambda is a scale
sim <- bd.sim(n0, lambda, mu, tMax, lShape = lShape, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# finally, we can also have a rate dependent on an environmental variable,
# like temperature data
```

```
# get temperature data (see ?temp)
data(temp)

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation - a scale
lambda <- 10
# note the scale for the age-dependency could be a time-varying function

# speciation shape
lShape <- 2

# extinction, dependent on temperature exponentially
mu <- function(t, env) {
  return(0.1*exp(0.025*env))
}

# need a data frame describing the temperature at different times
envM <- temp

# by passing mu and envM to bd.sim, internally bd.sim will make mu into a
# function dependent only on time, using make.rate
mFunc <- make.rate(mu, tMax = tMax, envRate = envM)

# take a look at how the rate itself will be
plot(seq(0, tMax, 0.1), rev(mFunc(seq(0, tMax, 0.1))),
     main = "Extinction rate varying with temperature", xlab = "Time (Mya)",
     ylab = "Rate (events/species/My)", type = 'l', xlim = c(tMax, 0))

# set seed
set.seed(2)

# run the simulation
sim <- bd.sim(n0, lambda, mu, tMax, lShape = lShape, envM = envM,
              nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# one can mix and match all of these scenarios as they wish - age-dependency
# and constant rates, age-dependent and temperature-dependent rates, etc.
# the only combination that is not allowed is a step function rate and
# environmental data, but one can get around that as follows
```

```
# get the temperature data - see ?temp for information on the data set
data(temp)

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation - a step function of temperature built using ifelse()
# note that this creates two shifts for lambda, for a total of 3 values
# throughout the simulation
lambda <- function(t, env) {
  ifelse(t < 20, env,
         ifelse(t < 30, env / 4, env / 3))
}

# speciation shape
lShape <- 2

# environment variable to use - temperature
envL <- temp

# this is kind of a complicated scale, let us take a look

# make it a function of time
lFunc <- make.rate(lambda, tMax = tMax, envRate = envL)

# plot it
plot(seq(0, tMax, 0.1), rev(lFunc(seq(0, tMax, 0.1))),
     main = "Speciation scale varying with temperature", xlab = "Time (Mya)",
     ylab = "Scale (1/(events/species/My))", type = 'l', xlim = c(tMax, 0))

# extinction
mu <- 0.1

# maximum simulation time
tMax <- 40

# set seed
set.seed(1)


# run the simulation
sim <- bd.sim(n0, lambda, mu, tMax, lShape = lShape, envL = envL,
              nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}
time2 <- Sys.time()
```

```
# after presenting the possible models, we can consider how to
# create mixed models, where the dependency changes over time

###
# consider speciation that becomes environment dependent
# in the middle of the simulation

# get the temperature data - see ?temp for information on the data set
data(temp)

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# time and temperature-dependent speciation
lambda <- function(t, temp) {
  return(
    ifelse(t < 20, 0.1 - 0.005*t,
           0.05 + 0.1*exp(0.02*temp))
  )
}

# extinction
mu <- 0.11

# set seed
set.seed(4)

# run simulation
sim <- bd.sim(n0, lambda, mu, tMax, envL = temp, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# we can also change the environmental variable
# halfway into the simulation

# note below that for this scenario we need make.rate, which
# in general can aid users looking for more complex scenarios
# than those available directly with bd.sim arguments

# get the temperature data - see ?temp for information on the data set
data(temp)

# same for co2 data (and ?co2)
```

```
data(co2)

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation
lambda <- 0.1

# temperature-dependent extinction
m_t1 <- function(t, temp) {
  return(0.05 + 0.1*exp(0.02*temp))
}

# make first function
mu1 <- make.rate(m_t1, tMax = tMax, envRate = temp)

# co2-dependent extinction
m_t2 <- function(t, co2) {
  return(0.02 + 0.14*exp(0.01*co2))
}

# make second function
mu2 <- make.rate(m_t2, tMax = tMax, envRate = co2)

# final extinction function
mu <- function(t) {
  ifelse(t < 20, mu1(t), mu2(t))
}

# set seed
set.seed(3)

# run simulation
sim <- bd.sim(n0, lambda, mu, tMax, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

# note one can also use this mu1 mu2 workflow to create a rate
# dependent on more than one environmental variable, by decoupling
# the dependence of each in a different function and putting those
# together

###
# finally, one could create an extinction rate that turns age-dependent
# in the middle, by making shape time-dependent
```

```
# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation
lambda <- 0.15

# extinction - note that since this is a Weibull scale,
# the unites are my/events/lineage, not events/lineage/my
mu <- function(t) {
  return(8 + 0.05*t)
}

# extinction shape
mShape <- function(t) {
  return(
    ifelse(t < 30, 1, 2)
  )
}

# set seed
set.seed(3)

# run simulation
sim <- bd.sim(n0, lambda, mu, tMax, mShape = mShape,
               nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim)
  ape::plot.phylo(phy)
}

###
# note nFinal has to be sensible
## Not run:
# this would return an error, since it is virtually impossible to get 100
# species at a process with diversification rate -0.09 starting at n0 = 1
sim <- bd.sim(1, lambda = 0.01, mu = 1, tMax = 100, nFinal = c(100, Inf))

## End(Not run)
```

`bd.sim.traits` *MuSSE simulation*

Description

Simulates a species birth-death process following the Multiple State-dependent Speciation and Extinction (MuSSE) or the Hidden State-dependent Speciation and Extinction (HiSSE) model for any

number of starting species. Allows for the speciation/extinction rate to be (1) a constant, or (2) a list of values for each trait state. Traits are simulated to evolve under a simple Mk model (see references). Results can be conditioned on either total simulation time, or total number of extant species at the end of the simulation. Also allows for constraining results on a range of number of species at the end of the simulation, either total or extant, using rejection sampling. Returns a `sim` object (see `?sim`), and a list of data frames describing trait values for each interval. It may return true extinction times or simply information on whether species lived after the maximum simulation time, depending on input. Can simulate any number of traits, but rates need to depend on only one (each, so speciation and extinction can depend on different traits).

Usage

```
bd.sim.traits(
  n0,
  lambda,
  mu,
  tMax = Inf,
  N = Inf,
  nTraits = 1,
  nFocus = 1,
  nStates = 2,
  nHidden = 1,
  X0 = 0,
  Q = list(matrix(c(0, 0.1, 0.1, 0), ncol = 2, nrow = 2)),
  nFinal = c(0, Inf),
  nExtant = c(0, Inf)
)
```

Arguments

`n0`	Initial number of species. Usually 1, in which case the simulation will describe the full diversification of a monophyletic lineage. Note that when `lambda` is less than or equal to `mu`, many simulations will go extinct before speciating even once. One way of generating large sample sizes in this case is to increase `n0`, which will simulate the diversification of a paraphyletic group.
`lambda`	Vector to hold the speciation rate over time. It should either be a constant, or a list of size `nStates`. For each species a trait evolution simulation will be run, and then used to calculate the final speciation rate. Note that `lambda` should always be greater than or equal to zero.
`mu`	Similar to above, but for the extinction rate.
`tMax`	Ending time of simulation, in million years after the clade origin. Any species still living after `tMax` is considered extant, and any species that would be generated after `tMax` is not present in the return.
`N`	Number of species at the end of the simulation. End of the simulation will be set for one of the times where `N` species are alive, chosen from all the times there were `N` species alive weighted by how long the simulation was in that situation. Exactly one of `tMax` and `N` must be non-`Inf`. Note that if `N` is the chosen condition, `mu` cannot be `0`, since paelobuddy's current algorithm would

mean only species 1 would have children. Future features will hopefully remove this limitation.

`nTraits` The number of traits to be considered. `lambda` and `mu` need not reference every trait simulated.

`nFocus` Trait of focus, i.e. the one that rates depend on. If it is one number, that will be the trait of focus for both speciation and extinction rates. If it is of length 2, the first will be the focus for the former, the second for the latter.

`nStates` Number of possible states for categorical trait. The range of values will be assumed to be `(0, nStates - 1)`. Can be a constant or a vector of length `nTraits`, if traits are intended to have different numbers of states.

`nHidden` Number of hidden states for categorical trait. Default is `1`, in which case there are no added hidden traits. Total number of states is then `nStates * nHidden`. States will then be set to a value in the range of `(0, nStates - 1)` to simulate that hidden states are hidden. This is done by setting the value of a state to the remainder of `state / nStates`. E.g. if `nStates = 2` and `nHidden = 3`, possible states during simulation will be in the range `(0, 5)`, but states `(2, 4)` (corresponding to `(0B, 0C)` in the nomenclature of the original HiSSE reference) will be set to `0`, and states `(3, 5)` (corresponding to `(1B, 1C)`) to `1`.

`X0` Initial trait value for original species. Must be within `(0, nStates - 1)`. Can be a constant or a vector of length `nTraits`.

`Q` Transition rate matrix for continuous-time trait evolution. For different states `i` and `j`, the rate at which a species at `i` transitions to `j` is `Q[i + 1, j + 1]`. Must be within a list, so as to allow for different `Q` matrices when `nTraits > 1`.

Note that for all of `nStates`, `nHidden`, `X0` and `Q`, if `nTraits > 1` and any of those is of length `1`, they will be considered to apply to all traits equally. This might lead to problems if, e.g., two traits have different states but the same `Q`, so double check that you are providing all parameters for the required traits.

`nFinal` A `vector` of length `2`, indicating an interval of acceptable number of species at the end of the simulation. Default value is `c(0, Inf)`, so that any number of species (including zero, the extinction of the whole clade) is accepted. If different from default value, simulation will restart until the number of total species at `tMax` is in the `nFinal` interval. Note that `nFinal` must be a sensible vector. The function will error if its maximum is lower than `1`, or if its length is not `2`.

`nExtant` A `vector` of length `2`, indicating an interval of acceptable number of extant species at the end of the simulation. Equal to `nFinal` in every respect except for that.

Note: The function returns `NA` if it runs for more than `100000` iterations without fulfilling the requirements of `nFinal` and `nExtant`.

Note: Using values other than the default for `nFinal` and `nExtant` will condition simulation results.

Details

Please note while time runs from `0` to `tMax` in the simulation, it returns speciation/extinction times as `tMax` (origin of the group) to `0` (the "present" and end of simulation), so as to conform to other packages in the literature.

Value

A `sim` object, containing extinction times, speciation times, parent, and status information for each species in the simulation, and a list object with the trait data frames describing the trait value for each species at each specified interval.

Author(s)

Bruno do Rosario Petrucci.

References

Maddison W.P., Midford P.E., Otto S.P. 2007. Estimating a binary character's effect on speciation and extinction. Systematic Biology. 56(5):701.

FitzJohn R.G. 2012. Diversitree: Comparative Phylogenetic Analyses of Diversification in R. Methods in Ecology and Evolution. 3:1084–1092.

Beaulieu J.M., O'Meara, B.C. 2016. Detecting Hidden Diversification Shifts in Models of Trait-Dependent Speciation and Extinction. Systematic Biology. 65(4):583-601.

Examples

```
###
# first, it's good to check that it can work with constant rates

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation
lambda <- 0.1

# extinction
mu <- 0.03

# set seed
set.seed(1)

# run the simulation, making sure we have more than one species in the end
sim <- bd.sim.traits(n0, lambda, mu, tMax, nFinal = c(2, Inf))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)
  ape::plot.phylo(phy)
}

###
# now let's actually make it trait-dependent, a simple BiSSE model
```

```
# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, trait-independent
mu <- 0.03

# number of traits and states (1 binary trait)
nTraits <- 1
nStates <- 2

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = c("red", "blue")[traits + 1])
}

###
# extinction can be trait-dependent too, of course

# initial number of species
n0 <- 1

# number of species at the end of the simulation
N <- 20

# speciation, higher for state 1
lambda <- c(0.1, 0.2)
```

```
# extinction, highe for state 0
mu <- c(0.06, 0.03)

# number of traits and states (1 binary trait)
nTraits <- 1
nStates <- 2

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, N = N, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = c("red", "blue")[traits + 1])
}

###
# we can complicate the model further by making transition rates asymmetric

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, lower for state 1
mu <- c(0.03, 0.01)

# number of traits and states (1 binary trait)
nTraits <- 1
nStates <- 2

# initial value of the trait
X0 <- 0
```

```
# transition matrix, with q01 higher than q10
Q <- list(matrix(c(0, 0.1,
                   0.25, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = c("red", "blue")[traits + 1])
}

###
# MuSSE is BiSSE but with higher numbers of states

# initial number of species
n0 <- 1

# number of species at the end of the simulation
N <- 20

# speciation, higher for state 1, highest for state 2
lambda <- c(0.1, 0.2, 0.3)

# extinction, higher for state 2
mu <- c(0.03, 0.03, 0.06)

# number of traits and states (1 trinary trait)
nTraits <- 1
nStates <- 3

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical, fully reversible transition rates
Q <- list(matrix(c(0, 0.1, 0.1,
                   0.1, 0, 0.1,
                   0.1, 0.1, 0), ncol = 3, nrow = 3))

# set seed
set.seed(1)
```

```
# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, N = N, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # 0 tips = red, 1 tips = blue, 2 tips = green
  ape::plot.phylo(phy, tip.color = c("red", "blue", "green")[traits + 1])
}

###
# HiSSE is like BiSSE, but with the possibility of hidden traits
# here we have 4 states, representing two states for the observed trait
# (0 and 1) and two for the hidden trait (A and B), i.e. 0A, 1A, 0B, 1B

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1A, highest for 1B
lambda <- c(0.1, 0.2, 0.1, 0.3)

# extinction, lowest for 0B
mu <- c(0.03, 0.03, 0.01, 0.03)

# number of traits and states (1 binary observed trait,
# 1 binary hidden trait)
nTraits <- 1
nStates <- 2
nHidden <- 2

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates. Only one transition
# is allowed at a time, i.e. 0A can go to 0B and 1A,
# but not to 1B, and similarly for others
Q <- list(matrix(c(0, 0.1, 0.1, 0,
                   0.1, 0, 0, 0.1,
                   0.1, 0, 0, 0.1,
                   0, 0.1, 0.1, 0), ncol = 4, nrow = 4))

# set seed
set.seed(1)

# run the simulation
```

```
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, nHidden = nHidden,
                     X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = c("red", "blue")[traits + 1])
}

###
# we can also increase the number of traits, e.g. to have a neutral trait
# evolving with the real one to compare the estimates of the model for each

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, lowest for state 0
mu <- c(0.01, 0.03)

# number of traits and states (2 binary traits)
nTraits <- 2
nStates <- 2

# initial value of both traits
X0 <- 0

# transition matrix, with symmetrical transition rates for trait 1,
# and asymmetrical (and higher) for trait 2
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2),
          matrix(c(0, 1,
                   0.5, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
```

```
traits1 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))
traits2 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[2]]$value, 1)))

# make index for coloring tips
index <- ifelse(!(traits1 | traits2), "red",
                ifelse(traits1 & !traits2, "purple",
                       ifelse(!traits1 & traits2, "magenta", "blue")))
# 00 = red, 10 = purple, 01 = magenta, 11 = blue

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = index)
}

###
# we can then do the same thing, but with the
# second trait controlling extinction

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 10 and 11
lambda <- c(0.1, 0.2)

# extinction, lowest for state 00 and 01
mu <- c(0.01, 0.03)

# number of traits and states (2 binary traits)
nTraits <- 2
nStates <- 2
nFocus <- c(1, 2)

# initial value of both traits
X0 <- 0

# transition matrix, with symmetrical transition rates for trait 1,
# and asymmetrical (and higher) for trait 2
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2),
          matrix(c(0, 1,
                   0.5, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
```

```
                  nStates = nStates, nFocus = nFocus,
                  X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits1 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))
traits2 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[2]]$value, 1)))

# make index for coloring tips
index <- ifelse(!(traits1 | traits2), "red",
                ifelse(traits1 & !traits2, "purple",
                       ifelse(!traits1 & traits2, "magenta", "blue")))
# 00 = red, 10 = purple, 01 = magenta, 11 = blue

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = index)
}

###
# as a final level of complexity, let us change the X0
# and number of states of the trait controlling extinction

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 10 and 11
lambda <- c(0.1, 0.2)

# extinction, lowest for state 00, 01, and 02
mu <- c(0.01, 0.03, 0.03)

# number of traits and states (2 binary traits)
nTraits <- 2
nStates <- c(2, 3)
nFocus <- c(1, 2)

# initial value of both traits
X0 <- c(0, 2)

# transition matrix, with symmetrical transition rates for trait 1,
# and asymmetrical, directed, and higher rates for trait 2
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2),
          matrix(c(0, 1, 0,
                   0.5, 0, 0.75,
                   0, 1, 0), ncol = 3, nrow = 3))
```

```
# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, nFocus = nFocus,
                     X0 = X0, Q = Q, nFinal = c(2, Inf))

# get trait values for all tips
traits1 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[1]]$value, 1)))
traits2 <- unlist(lapply(sim$TRAITS, function(x) tail(x[[2]]$value, 1)))

# make index for coloring tips
index <- ifelse(!(traits1 | (traits2 != 0)), "red",
                ifelse(traits1 & (traits2 == 0), "purple",
                       ifelse(!traits1 & (traits2 == 1), "magenta",
                              ifelse(traits1 & (traits2 == 1), "blue",
                                     ifelse(!traits1 & (traits2 == 2),
                                            "orange", "green")))))

# 00 = red, 10 = purple, 01 = magenta, 11 = blue, 02 = orange, 12 = green

# we can plot the phylogeny to take a look
if (requireNamespace("ape", quietly = TRUE)) {
  phy <- make.phylo(sim$SIM)

  # color 0 valued tips red and 1 valued tips blue
  ape::plot.phylo(phy, tip.color = index)
}
# one could further complicate the model by adding hidden states
# to each trait, each with its own number etc, but these examples
# include all the tools necessary to make these or further extensions
```

`bin.occurrences` *Bin true occurrences into geologic intervals*

Description

Given the output of `sample.clade(..., returnTrue = FALSE)`, returns the occurrence counts in each bin (i.e., the same as `sample.clade(..., returnTrue = TRUE)`). This helps to trace perfect parallels between both output formats of `sample.clade`.

Usage

```
bin.occurrences(fossils, bins)
```

Arguments

fossils	A data.frame exactly as returned by sample.clade(..., returnTrue = FALSE). See ?sample.clade) for details.
bins	A vector of time intervals corresponding to geological time ranges.

Details

This function helps a user bin "true occurrences" directly into binned occurrences, allowing for comparisons among "perfectly known" fossil records and records that have a certain resolution (given by the bins parameter).

Value

A data.frame exactly as returned by sample.clade(..., returnTrue = TRUE). See ?sample.clade) for details.

Author(s)

Matheus Januario.

Examples

```
###
# set seed
set.seed(1)

# run a birth-death simulation
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.05, tMax = 50)

# choose bins
bins <- seq(0, 50, by = 1)

# generate "true" fossil occurrences
fossils_true <- sample.clade(sim, rho = 1, tMax = 50, returnTrue = TRUE)

# bin the true occurrences
fossils_binned <- bin.occurrences(fossils_true, bins)

# compare
fossils_true
fossils_binned
```

binner *Bin occurrences in geologic intervals*

Description

Given a vector of fossil occurrences and time bins to represent geological ranges, returns the occurrence counts in each bin.

Usage

```
binner(x, bins)
```

Arguments

x	The vector containing occurrence times for one given species.
bins	A vector of time intervals corresponding to geological time ranges.

Details

The convention for couting occurrences inside a bin is to count all occurrences exactly in the boundary furthest from zero and exclude bins exactly in the boundary closest to zero. Then, in the bin closest to zero (i.e., the "last", or "most recent" bin), include all occurrence on each of the two boundaries. So occurrences that fall on a boundary are placed on the most recent bin possible

Value

A vector of occurrence counts for each interval, sorted from furthest to closest to zero.

Author(s)

Matheus Januario and Bruno do Rosario Petrucci

Examples

```
###
# first let us create some artificial occurrence data and check

# occurrence vector
x <- c(5.2, 4.9, 4.1, 3.2, 1, 0.2)

# bins vector
bins <- c(6, 5, 4, 3, 2, 1, 0)

# result
binnedSamp <- binner(x, bins)
binnedSamp

###
# it should work with any type of number in bins
```

```
# occurrence vector
x <- c(6.7, 5.03, 4.2, 3.4, 1.2, 0.4)

# bins vector
bins <- c(7.2, 6.7, 5.6, 4.3, 3.2, sqrt(2), 1, 0)

# result
binnedSamp <- binner(x, bins)
binnedSamp

###
# let us try with a real simulated species fossil record

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim(1, lambda = 0.1, mu = 0.05, tMax = 15)

# sample it
sampled <- sample.clade(sim = sim, rho = 1, tMax = 15, S = 1)$SampT

# bins vector
bins <- c(15.1, 12.3, 10, 7.1, 5.8, 3.4, 2.2, 0)

# result
binnedsample <- binner(sampled, bins)
binnedsample
```

---

co2 *Jurassic CO2 data*

Description

CO2 data during the Jurassic. Modified from the co2 set in RPANDA, originally taken from Mayhew et al (2008, 2012). Inverted so lower times represent time since first measurement, to be in line with the past-to-present convention of most time-dependent functions in paleobuddy.

Usage

```
data(co2)
```

Format

A data frame with 53 rows and 2 variables:

t A numeric vector representing time since the beginning of the data frame age, 520 million years ago, in million years. We set this from past to present as opposed to present to past since

birth-death functions in paleobuddy consider time going in the former direction. Hence t = 0 represents the time point at 520mya, while t = 520 represents the present.

co2 A numeric vector representing CO2 concentration as the ratio of CO2 mass at t over the present.

Source

https://github.com/hmorlon/PANDA

References

Morlon H. et al (2016) RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution* 7: 589-597.

Mayhew, P.J. et al (2008) A long-term association between global temperature and biodiversity, origination and extinction in the fossil record *Proc. of the Royal Soc. B* 275:47-53.

Mayhew, P.J. et al (2012) Biodiversity tracks temperature over time *Proc. of the Nat. Ac. of Sci. of the USA* 109:15141-15145.

Berner R.A. & Kothavala, Z. (2001) GEOCARB III: A revised model of atmospheric CO2 over Phanerozoic time *Am. J. Sci.* 301:182–204.

---

draw.sim *Draw a sim object*

Description

Draws species longevities for a paleobuddy simulation (a sim object - see ?sim) in the graphics window. Allows for the assignment of speciation and sampling events, and further customization.

Usage

```
draw.sim(
  sim,
  traits = NULL,
  fossils = NULL,
  lineageColors = NULL,
  sortBy = "TS",
  lwdLin = 4,
  tipLabels = NULL,
  showLabel = TRUE,
  traitID = 1,
 traitColors = c("#a40000", "#16317d", "#007e2f", "#ffcd12", "#b86092", "#721b3e",
    "#00b7a7"),
  traitLegendPlacement = "topleft",
  fossilsFormat = "exact",
  fossilRangeAlpha = 100,
  restoreOldPar = TRUE,
  ...
)
```

Arguments

sim	A sim object, containing extinction times, speciation times, parent, and status information for each species in the simulation. See ?sim.
traits	A list of data frames enconding the value of one or more traits during the lifetime of each species, usually coming from the TRAITS member of the output of bd.sim.traits. It should have length equal to the number of species in sim, and the traitIDth trait (see below) (i.e. the data frame of number traitID for each species) will be used to draw trait values.
fossils	A data.frame containing the fossil occurrences of each lineage, e.g. as returned by the sample.clade function. The format of this argument will define the way fossils are drawn (see below).
lineageColors	Character vector giving the colors of all lineages, sorted by the original lineage order (the one in the sim object). Must have same length as the number of lineages in the sim object. If NULL (default value) all lineages are plotted as black. this parameter has no effect if traits is also provided.
sortBy	A single character or integer vector indicating how lineages should be sorted in the plot. If it is a string (see example 3), it indicates which element in the sim object that should be used to sort lineages in the plot. If it is a vector of integers, it directly specifies the order in which lineages should be drawn, from the bottom (i.e. the first integer) to the upper side (#th integer, with # = number of lineages in sim) of the figure. Default value of this parameter is "TS", so by default species will be sorted by order of origination in the simulation.
lwdLin	The relative thickness/size of all elements (i.e., lines and points in the plot. Default value is 4 (i.e. equal to lwd = 4 for the black horizontal lines).
tipLabels	Character vector manually assigning the tip labels of all lineages, sorted by the original lineage order (the one in the sim object). Must have same length as the number of lineages in the sim object. If NULL (default value) all lineages are plotted as "t#", with "#" being the position of that lineage in the sim object.
showLabel	A logical on whether to draw species labels (i.e. species 1 being t1, species 2 t2 etc.). Default is TRUE.
traitID	Numerical giving the trait which will be plotted. this parameter is only useful when multiple traits were simulated in the same sim object, i.e. when traits has more than one data frame per species.
traitColors	Character vector providing colors for the states of a given trait, so its length must equal or exceed the number of states. Default values provide 7 colors (and so they can plot up to 7 states).
traitLegendPlacement	Placement of state legend. Accepted values are "topleft" (default value), "bottomleft", "bottomright", "topright", and "none".
fossilsFormat	Character assigning if fossils will be represented by exact time placements ("exact", default value), by horizontal bars giving range information ("ranges"), or by both forms ("all").
fossilRangeAlpha	Numerical giving color transparency for fossil range representation. Integers between 0 and 255 are preferred, but any float between 0 and 1 is also accepted. Default value is 100.

restoreOldPar	Logical assigning if plot default values show be restored after function finalizes plotting. Deafult is TRUE, but users interesting in using plot additions (e.g. abline() to highlight a certain age) should assign this as FALSE to use the x and y values in the plot. If false, x-axis follows time, and y-axis follows the number of species plotted, with 1 being the bottom lineage, and the upper y-limit being the Nth lineage in the sim.
...	Further arguments to be passed to plot

Value

A plot of the simulation in the graphics window. If the fossils data.frame is supplied, its format will dictate how fossil occurrences will be plotted. If fossils has a SampT column (i.e. the occurrence times are exact), fossil occurrences are assigned as dots. If fossils has columns MaxT and MinT (i.e. the early and late stage bounds associated with each occurrence), fossil occurrences are represented as slightly jittered, semitransparent bars indicating the early and late bounds of each fossil occurrence.

Author(s)

Matheus Januario

Examples

```
# we start drawing a simple simulation

# maximum simulation time
tMax <- 10

# set seed
set.seed(1)

# run a simulation
sim <- bd.sim(n0 = 1, lambda = 0.6, mu = 0.55, tMax = tMax,
              nFinal = c(10,20))

# draw it
draw.sim(sim)

###
# we can add fossils to the drawing

# maximum simulation time
tMax <- 10

# set seed
set.seed(1)

# run a simulation
sim <- bd.sim(n0 = 1, lambda = 0.6, mu = 0.55, tMax = tMax,
              nFinal = c(10,20))
```

```
# set seed
set.seed(1)

# simulate data resulting from a fossilization process
# with exact occurrence times
fossils <- sample.clade(sim = sim, rho = 4, tMax = tMax, returnTrue = TRUE)

# draw it
draw.sim(sim, fossils = fossils)

# we can order the vertical drawing of species based on
# any element of sim
draw.sim(sim, fossils = fossils, sortBy = "PAR")
# here we cluster lineages with their daughters by
# sorting them by the "PAR" list of the sim object

draw.sim(sim, fossils = fossils, sortBy = "TE")
# here we sort lineages by their extinction times

###
# fossils can also be represented by ranges

# maximum simulation time
tMax <- 10

# set seed
set.seed(1)

# run birth-death simulation
sim <- bd.sim(n0 = 1, lambda = 0.6, mu = 0.55, tMax = tMax,
              nFinal = c(10,20))

# simulate data resulting from a fossilization process
# with fossil occurrence time ranges

# set seed
set.seed(20)

# create time bins randomly
bins <- c(tMax, 0, runif(n = rpois(1, lambda = 6), min = 0, max = tMax))

# set seed
set.seed(1)

# simulate fossil sampling
fossils <- sample.clade(sim = sim, rho = 2, tMax = tMax,
                        returnTrue = FALSE, bins = bins)

# get old par
oldPar <- par(no.readonly = TRUE)

# draw it, sorting lineages by their parent
draw.sim(sim, fossils = fossils, sortBy = "PAR",
```

```
         fossilsFormat = "ranges", restoreOldPar = FALSE)

# adding the bounds of the simulated bins
abline(v = bins, lty = 2, col = "blue", lwd = 0.5)

# alternatively, we can draw lineages varying colors and tip labels
# (note how they are sorted)
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges",
         tipLabels = paste0("spp_", 1:length(sim$TS)),
         lineageColors = rep(c("red", "green", "blue"), times = 5))

# restore old par
par(oldPar)

###
# we can control how to sort displayed species exactly

# maximum simulation time
tMax <- 10

# set seed
set.seed(1)

# run birth-death simulations
sim <- bd.sim(n0 = 1, lambda = 0.6, mu = 0.55, tMax = tMax,
              nFinal = c(10,20))

# set seed
set.seed(1)

# simulate fossil sampling
fossils <- sample.clade(sim = sim, rho = 4, tMax = tMax, returnTrue = TRUE)

# draw it with random sorting (in pratice this could be a trait
# value, for instance)
draw.sim(sim, fossils = fossils, sortBy = sample(1:length(sim$TS)))

###
# we can display trait values as well

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, lowest for state 0
mu <- c(0.01, 0.03)

# number of traits and states (2 binary traits)
```

```
nTraits <- 2
nStates <- 2

# initial value of both traits
X0 <- 0

# transition matrix, with symmetrical transition rates for trait 1,
# and asymmetrical (and higher) for trait 2
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2),
          matrix(c(0, 1,
                   0.5, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, 10))

# maybe we want to take a look at the traits of fossil records too
fossils <- sample.clade(sim$SIM, rho = 0.5, tMax = max(sim$SIM$TS),
                        returnAll = TRUE, bins = seq(0, 20, by = 1))

draw.sim(sim$SIM, traits = sim$TRAITS, sortBy = "PAR",
         fossils = fossils, fossilsFormat = "all",
         traitLegendPlacement = "bottomleft")
# note how fossil ranges are displayed above and below the true
# occurrence times, but we could also draw only one or the other

# just ranges
draw.sim(sim$SIM, traits = sim$TRAITS, sortBy = "PAR",
         fossils = fossils, fossilsFormat = "ranges",
         traitLegendPlacement = "bottomleft")

# just true occurrence times
draw.sim(sim$SIM, traits = sim$TRAITS, sortBy = "PAR", traitID = 2,
         fossils = fossils, fossilsFormat = "exact",
         traitLegendPlacement = "bottomleft")
# note the different traitID, so that segments are colored
# following the value of the second trait
```

`find.lineages` *Separate a paleobuddy simulation into monophyletic clades*

Description

Separates a `sim` object into `sim` objects each with a mother species and its descendants. If argument `S` is not used, it returns by default the list of `sim` objects descended from each species with an `NA`

parent in the original input (meaning species alive at the beginning of the simulation). If a vector of numbers is supplied for S, the list of sim objects return will instead be descended from each species in S. Returns for each clade a vector with the original identity of member species as well.

Usage

```
find.lineages(sim, S = NULL)
```

Arguments

sim	A sim object, containing extinction times, speciation times, parent, and status information for each species in the simulation. See ?sim.
S	A vector of species in sim. If not supplied, S will be the starting species in the simulation, i.e. those for which the parent is NA. If only one species has NA as parent, there is only one clade in the sim object, and therefore the function will return the input.

Value

A list object with (named) sim objects corresponding to the clades descended from species in S. For each clade, an extra vector LIN is included so the user can identify the order of species in the returned sim objects with the order of species in the original simulation.

Author(s)

Bruno do Rosario Petrucci and Matheus Januario.

Examples

```
###
# first, we run a simple simulation with one starting species

# set seed
set.seed(1)

# run simulation with a minimum of 20 species
sim <- bd.sim(n0 = 3, lambda = 0.1, mu = 0.1, tMax = 10,
              nFinal = c(20, Inf))

# get a simulation object with the clade originating from species 2
clades <- find.lineages(sim, S = 2)

# now we can check to make sure the subclade was correctly separated

# change NA to 0 on the clade's TE
clades[[1]]$sim$TE[clades[[1]]$sim$EXTANT] <- 0

# plot the phylogeny
if (requireNamespace("ape", quietly = TRUE)) {
  plot <- ape::plot.phylo(
    make.phylo(clades[[1]]$sim),
```

```
    main = "red: extinction events \n blue: speciation events");
  ape::axisPhylo()
}

# check speciation times
for (j in 2:length(clades[[1]]$sim$TS)) {
  # the subtraction is just to adjust the wt with the plot scale
  lines(x = c(
    sort(clades[[1]]$sim$TS, decreasing = TRUE)[2] -
       clades[[1]]$sim$TS[j],
    sort(clades[[1]]$sim$TS, decreasing = TRUE)[2] -
       clades[[1]]$sim$TS[j]),
    y = c(plot$y.lim[1], plot$y.lim[2]), lwd = 2, col = "blue")
}

# check extinction times:
for (j in 1:length(sim$TE)) {
  # the subtraction is just to adjust the wt with the plot scale
  lines(x = c(
    sort(clades[[1]]$sim$TS, decreasing = TRUE)[2] -
       clades[[1]]$sim$TE[j],
    sort(clades[[1]]$sim$TS, decreasing = TRUE)[2] -
       clades[[1]]$sim$TE[j]),
    y = c(plot$y.lim[1], plot$y.lim[2]), lwd = 2, col = "red")
}

###
# now we try a simulation with 3 clades

# set seed
set.seed(4)

# run simulation
sim <- bd.sim(n0 = 3, lambda = 0.1, mu = 0.1, tMax = 10,
              nFinal = c(20, Inf))

# get subclades descended from original species
clades <- find.lineages(sim)

# get current par options so we can reset later
oldPar <- par(no.readonly = TRUE)

# set up for plotting side by side
par(mfrow = c(1, length(clades)))

# for each clade
for (i in 1:length(clades)) {
  # change NA to 0 on the clade's TE
  clades[[i]]$sim$TE[clades[[i]]$sim$EXTANT] <- 0

  # if there is only one lineage in the clade, nothing happens
  if (length(clades[[i]]$sim$TE) < 2) {
    # placeholder plot
```

```
    plot(NA, xlim = c(-1, 1), ylim = c(-1, 1))
    text("simulation with \n just one lineage", x = 0, y = 0.5, cex = 2)
  }

  # else, plot phylogeny
  else {
    if (requireNamespace("ape", quietly = TRUE)) {
      plot <- ape::plot.phylo(
        make.phylo(clades[[i]]$sim),
        main = "red: extinction events \n blue: speciation events");
      ape::axisPhylo()
    }

    # check speciation times
    for (j in 2:length(clades[[i]]$sim$TS)) {
      # the subtraction is just to adjust the wt with the plot scale
      lines(x = c(
        sort(clades[[i]]$sim$TS, decreasing = TRUE)[2] -
           clades[[i]]$sim$TS[j],
        sort(clades[[i]]$sim$TS, decreasing = TRUE)[2] -
           clades[[i]]$sim$TS[j]),
        y = c(plot$y.lim[1], plot$y.lim[2]), lwd = 2, col = "blue")
    }

    # check extinction times:
    for (j in 1:length(sim$TE)) {
      # the subtraction is just to adjust the wt with the plot scale
      lines(x = c(
        sort(clades[[i]]$sim$TS, decreasing = TRUE)[2] -
           clades[[i]]$sim$TE[j],
        sort(clades[[i]]$sim$TS, decreasing = TRUE)[2] -
           clades[[i]]$sim$TE[j]),
        y = c(plot$y.lim[1], plot$y.lim[2]), lwd = 2, col = "red")
    }
  }
}

# reset par
par(oldPar)

###
# we can also have an example with more non-starting species in S

# set seed
set.seed(3)

# run simulation
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10,
              nFinal = c(10, Inf))

# get current par options so we can reset later
oldPar <- par(no.readonly = TRUE)
```

```
# set up for plotting side by side
par(mfrow = c(1, 2))

if (requireNamespace("ape", quietly = TRUE)) {

  # first we plot the clade started by 1
  ape::plot.phylo(make.phylo(sim), main = "original")
  ape::axisPhylo()

  # this should look the same
  ape::plot.phylo(make.phylo(find.lineages(sim)[[1]]$sim),
                  main="after find.lineages()")
  ape::axisPhylo()

  # get sublcades descended from the second and third species
  clades <- find.lineages(sim, c(2,3))

  # and these should be part of the previous phylogenies
  ape::plot.phylo(make.phylo(clades$clade_2$sim),
                  main = "Daughters of sp 2")
  ape::axisPhylo()

  ape::plot.phylo(make.phylo(clades$clade_3$sim),
                  main = "Daughters of sp 3")
  ape::axisPhylo()
}

# reset par
par(oldPar)

###
# if there is only one clade and we use the default for
# S, we get back the original simulation object

# set seed
set.seed(1)

# run simulation
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.08, tMax = 10,
              nFinal = c(5, Inf))

# get current par options so we can reset later
oldPar <- par(no.readonly = TRUE)

# set up for plotting side by side
par(mfrow = c(1, 2))

# plotting sim and find.lineages(sim) - should be equal
if (requireNamespace("ape", quietly = TRUE)) {

  ape::plot.phylo(make.phylo(sim), main="original")
  ape::axisPhylo()
  ape::plot.phylo(make.phylo(find.lineages(sim)[[1]]$sim),
```

```
                  main="after find.lineages()")
  ape::axisPhylo()
}

# reset par
par(oldPar)
```

---

make.phylo *Phylogeny generating*

Description

Generates a phylogeny from a sim object containing speciation and extinction times, parent and status information (see ?sim). Returns a phylo object containing information on the phylogeny, following an "evolutionary Hennigian" (sensu Ezard et al 2011) format (i.e., a bifurcating tree). Takes an optional argument encoding fossil occurrences to return a sampled ancestor tree (see references). This tree consists of the original tree, plus the fossil occurrences added as branches of length 0 branching off of the corresponding species at the time of occurrence. Such trees can be used, as is or with small modifications, as starting trees in phylogenetic inference software that make use of the fossilized birth-death model. Returns NA and sends a warning if the simulation has only one lineage or if more than one species has NA as parent (i.e. there is no single common ancestor in the simulation). In the latter case, please use find.lineages first.

Usage

```
make.phylo(
  sim,
  fossils = NULL,
  saFormat = "branch",
  returnTrueExt = TRUE,
  returnRootTime = NULL
)
```

Arguments

sim	A sim object, containing extinction times, speciation times, parent, and status information for each species in the simulation. See ?sim.
fossils	A data frame with a "Species" column and a SampT column, usually an output of the sample.clade function. Species names must contain only one number each, corresponding to the order of the sim vectors. Note that we require it to have a SampT column, i.e. fossils must have an exact age. This assumption might be relaxed in the future.
saFormat	A string indicating which form sampled ancestors should take in the tree. If set to "branch" (default), SAs are returned as 0-length branches. If set to "node", they are returned as degree-2 nodes. Note that some software prefer the former (e.g. Beast) and some the latter (e.g. RevBayes). The code for making 0-length branches become nodes was written by Joshua A. Justison.

`returnTrueExt` A logical indicating whether to include in tree the tips representing the true extinction time of extinct species. If set to `FALSE`, the returned tree will include those tips. If `TRUE` (default), they will be dropped and instead the last sampled fossil of a given species will be the last sampled tip of that species. Note that if a species was not sampled it will then not appear in the tree. If no fossils have been added to the tree with the parameter `fossils`, this will have the same effect as the ape function `drop.fossil`, returning an ultrametric tree. Note that if this is set to `FALSE`, the `root.time` and `root.edge` arguments will not be accurate, depending on which species are dropped. The user is encouraged to use the `ape` package to correct these problems, as shown in an example below.

`returnRootTime` Logical indicating if phylo should have information regarding `root.time`. If set to `NULL` (default), returned phylogenies will not have `root.time` if there is at least one extant lineage in the sim object. If there are only extinct lineages in the `sim` object and it is set to `NULL`, `root.time` will be returned. If set to `FALSE` or `TRUE`, `root.time` will be removed or forced into the phylo object, respectively. In this case, we highly recommend users to read about the behavior of some functions (such as APE's `axisPhylo`) when this argument is forced.

Details

When `root.time` is added to a phylogeny, packages such as APE can change their interpretation of the information in the `phylo` object. For instance, a completely extinct phylogeny might be interpreted as extant if there is no info about `root.time`. This might create misleading interpretations even with simple functions such as `ape::axisPhylo`. `make.phylo` tries to accommodate different evo/paleo practices in its default value for `returnRootTime` by automatically attributing `root.time` when the `sim` object is extinct. We encourage careful inspection of output if users force `make.phylo` to use a specific behavior, especially when using phylogenies generated by this function as input in functions from other packages. For extinct phylogenies, it might usually be important to explicitly provide information that the edge is indeed a relevant part of the phylogeny (for instance adding `root.edge = TRUE` when plotting a phylogeny with `root.time` information with `ape::plot.phylo`. An example below provides a visualization of this issue.

Value

A `phylo` object from the APE package. Tip labels are numbered following the order of species in the `sim` object. If fossil occurrence data was supplied, the tree will include fossil occurrences as tips with branch length `0`, bifurcating at its sampling time from the corresponding species' edge (i.e. a sampled ancestor tree). Note that to obtain a true sampled ancestor (SA) tree, one must perform the last step of deleting tips that are not either extant or fossil occurrences (i.e. the tips at true time of extinction).

Note this package does not depend on APE (Paradis et al, 2004) since it is never used inside its functions, but it is suggested since one might want to manipulate the phylogenies generated by this function. Furthermore, a limited version of the `drop.tip` function from APE has been copied for use in this function (namely, due to the parameter `returnTrueExt`). Likewise, a limited version of `collapse.singles` and `node.depth.edgelength` were also copied to support those features. One does not need to have APE installed for the function to use that code, but the authors wished to do their due diligence by crediting the package and its maintainers.

Author(s)

Matheus Januario and Bruno do Rosario Petrucci

References

Ezard, T. H., Pearson, P. N., Aze, T., & Purvis, A. (2012). The meaning of birth and death (in macroevolutionary birth-death models). Biology letters, 8(1), 139-142.

Paradis, E., Claude, J., Strimmer, & K. (2004). APE: Analyses of Phylogenetics and Evolution in R language. Bioinformatics, 20(2), 289-290.

Heath, T. A., Huelsenbeck, J. P., & Stadler, T. (2014). The fossilized birth–death process for coherent calibration of divergence-time estimates. Proceedings of the National Academy of Sciences, 111(29), E2957-E2966.

Examples

```
###
# we can start with a simple phylogeny

# set a simulation seed
set.seed(1)

# simulate a BD process with constant rates
sim <- bd.sim(n0 = 1, lambda = 0.3, mu = 0.1, tMax = 10,
              nExtant = c(2, Inf))

# make the phylogeny
phy <- make.phylo(sim)

# plot it
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  ape::plot.phylo(phy)

  # we can also plot only the molecular phylogeny
  ape::plot.phylo(ape::drop.fossil(phy))

  # reset par
  par(oldPar)
}

###
# this works for sim generated with any of the scenarios in bd.sim

# set seed
set.seed(1)
```

```
# simulate
sim <- bd.sim(n0 = 1, lambda = function(t) 0.2 + 0.01*t,
              mu = function(t) 0.03 + 0.015*t, tMax = 10,
              nExtant = c(2, Inf))

# make the phylogeny
phy <- make.phylo(sim)

# plot it
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  # plot phylogeny
  ape::plot.phylo(phy)
  ape::axisPhylo()

  # we can also plot only the molecular phylogeny
  ape::plot.phylo(ape::drop.fossil(phy))
  ape::axisPhylo()

  # reset par
  par(oldPar)
}

###
# we can use the fossils argument to generate a sample ancestors tree

# set seed
set.seed(1)

# simulate a simple birth-death process
sim <- bd.sim(n0 = 1, lambda = 0.2, mu = 0.05, tMax = 10,
              nExtant = c(2, Inf))

# make the traditional phylogeny
phy <- make.phylo(sim)

# sample fossils
fossils <- sample.clade(sim, 0.1, 10)

# make the sampled ancestor tree
fbdPhy <- make.phylo(sim, fossils)

# plot them
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # visualize longevities and fossil occurrences
```

```
  draw.sim(sim, fossils = fossils)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  # phylogeny
  ape::plot.phylo(phy, main = "Phylogenetic tree")
  ape::axisPhylo()

  # sampled ancestor tree
  ape::plot.phylo(fbdPhy, main = "Sampled Ancestor tree")
  ape::axisPhylo()

  # reset par
  par(oldPar)
}

###
# we can instead have the sampled ancestors as degree-2 nodes

# set seed
set.seed(1)

# simulate a simple birth-death process
sim <- bd.sim(n0 = 1, lambda = 0.2, mu = 0.05, tMax = 10,
              nExtant = c(2, Inf))

# make the traditional phylogeny
phy <- make.phylo(sim)

# sample fossils
fossils <- sample.clade(sim, 0.1, 10)

# make the sampled ancestor tree
fbdPhy <- make.phylo(sim, fossils, saFormat = "node")

# plot them
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # visualize longevities and fossil occurrences
  draw.sim(sim, fossils = fossils)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  # phylogeny
  ape::plot.phylo(phy, main = "Phylogenetic tree")
  ape::axisPhylo()

  # sampled ancestor tree, need show.node.label parameter to see SAs
  ape::plot.phylo(fbdPhy, main = "Sampled Ancestor tree",
```

```
                  show.node.label = TRUE)
  ape::axisPhylo()

  # reset par
  par(oldPar)
}

###
# we can use the returnTrueExt argument to delete the extinct tips and
# have the last sampled fossil of a species as the fossil tip instead

# set seed
set.seed(5)

# simulate a simple birth-death process
sim <- bd.sim(n0 = 1, lambda = 0.2, mu = 0.05, tMax = 10,
              nExtant = c(2, Inf))

# make the traditional phylogeny
phy <- make.phylo(sim)

# sample fossils
fossils <- sample.clade(sim, 0.5, 10)

# make the sampled ancestor tree
fbdPhy <- make.phylo(sim, fossils, saFormat = "node", returnTrueExt = FALSE)
# returnTrueExt = FALSE means the extinct tips will be removed,
# so we will only see the last sampled fossil (see tree below)

# plot them
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # visualize longevities and fossil occurrences
  draw.sim(sim, fossils = fossils)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  # phylogeny
  ape::plot.phylo(phy, main = "Phylogenetic tree")
  ape::axisPhylo()

  # sampled ancestor tree, need show.node.label parameter to see SAs
  ape::plot.phylo(fbdPhy, main = "Sampled Ancestor tree",
                  show.node.label = TRUE)
  ape::axisPhylo()
  # note how t1.3 is an extinct tip now, as opposed to t1, since
  # we would not know the exact extinction time for t1,
  # rather just see the last sampled fossil

  # reset par
```

```
  par(oldPar)
}

###
# suppose in the last example, t2 went extinct and left no fossils
# this might lead to problems with the root.time object

# set seed
set.seed(5)

# simulate a simple birth-death process
sim <- bd.sim(n0 = 1, lambda = 0.2, mu = 0.05, tMax = 10,
              nExtant = c(2, Inf))

# make the traditional phylogeny
phy <- make.phylo(sim)

# sample fossils
fossils <- sample.clade(sim, 0.5, 10)

# make it so t2 is extinct
sim$TE[2] <- 9
sim$EXTANT[2] <- FALSE

# take out fossils of t2
fossils <- fossils[-which(fossils$Species == "t2"), ]

# make the sampled ancestor tree
fbdPhy <- make.phylo(sim, fossils, saFormat = "node", returnTrueExt = FALSE)
# returnTrueExt = FALSE means the extinct tips will be removed,
# so we will only see the last sampled fossil (see tree below)

# plot them
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # visualize longevities and fossil occurrences
  draw.sim(sim, fossils = fossils)

  # change par to show phylogenies
  par(mfrow = c(1, 2))

  # phylogeny
  ape::plot.phylo(phy, main = "Phylogenetic tree")
  ape::axisPhylo()

  # sampled ancestor tree, need show.node.label parameter to see SAs
  ape::plot.phylo(fbdPhy, main = "Sampled Ancestor tree",
                  show.node.label = TRUE)
  ape::axisPhylo()
  # note how t2 is gone, since it went extinct and left no fossils
```

```
  # this made it so the length of the tree + the root edge
  # does not equal the origin time of the simulation anymore
  max(ape::node.depth.edgelength(fbdPhy)) + fbdPhy$root.edge
  # it should equal 10

  # to correct it, we need to set the root edge again
  fbdPhy$root.edge <- 10 - max(ape::node.depth.edgelength(fbdPhy))
  # this is necessary because ape does not automatically fix the root.edge
  # when species are dropped, and analyes using phylogenies + fossils
  # frequently condition on the origin of the process

  # reset par
  par(oldPar)
}

###
# finally, we can test the usage of returnRootTime

# set seed
set.seed(1)

# simulate a simple birth-death process with more than one
# species and completely extinct:
sim <- bd.sim(n0 = 1, lambda = 0.5, mu = 0.5, tMax = 10, nExtant = c(0, 0))

# make a phylogeny using default values
phy <- make.phylo(sim)

# force phylo to not have root.time info
phy_rootless <- make.phylo(sim, returnRootTime = FALSE)

# plot them
if (requireNamespace("ape", quietly = TRUE)) {
  # store old par settings
  oldPar <- par(no.readonly = TRUE)

  # change par to show phylogenies
  par(mfrow = c(1, 3))

  # if we use the default value, axisPhylo works as intended
  ape::plot.phylo(phy, root.edge = TRUE, main = "root.time default value")
  ape::axisPhylo()

  # note that without root.edge, we have incorrect times,
  # as APE assumes tMax was the time of first speciation
  ape::plot.phylo(phy, main = "root.edge not passed to plot.phylo")
  ape::axisPhylo()

  # if we force root.time to be FALSE, APE assumes the tree is
  # ultrametric, which leads to an incorrect time axis
  ape::plot.phylo(phy_rootless, main = "root.time forced as FALSE")
  ape::axisPhylo()
  # note time scale in axis
```

```
  # reset par
  par(oldPar)
}
```

`make.rate` *Create a flexible rate for birth-death or sampling simulations*

Description

Generates a function determining the variation of a rate (speciation, extinction, sampling) with respect to time. To be used on birth-death or sampling functions, it takes as the base rate (1) a constant, (2) a function of time, (3) a function of time and a time-series (usually an environmental variable), or (4) a vector of numbers describing rates as a step function. Requires information regarding the maximum simulation time, and allows for optional extra parameters to tweak the baseline rate.

Usage

```
make.rate(rate, tMax = NULL, envRate = NULL, rateShifts = NULL)
```

Arguments

`rate` The baseline function with which to make the rate. It can be a

A number For constant birth-death rates.

A function of time For rates that vary with time. Note that this can be any function of time.

A function of time and an environmental variable For rates varying with time and an environmental variable, such as temperature. Note that supplying a function on more than one variable without an accompanying `envRate` will result in an error.

A numeric vector To create step function rates. Note this must be accompanied by a corresponding vector of rate shift times, `rateShifts`.

`tMax` Ending time of simulation, in million years after the clade's origin. Needed to ensure `rateShifts` runs the correct way.

`envRate` A `data.frame` representing a time-series, usually an environmental variable (e.g. CO2, temperature, etc) varying with time. The first column of this `data.frame` must be time, and the second column must be the values of the variable. The function will return an error if the user supplies `envRate` without `rate` being a function of two variables. `paleobuddy` has two environmental data frames, `temp` and `co2`. One can check `RPANDA` for more examples.

Note that, since simulation functions are run in forward-time (i.e. with `0` being the origin time of the simulation), the time column of `envRate` is assumed to do so as well, so that the row corresponding to `t = 0` is assumed to be the value

of the time-series when the simulation starts, and `t = tMax` is assumed to be its value when the simulation ends (the present).

Acknowledgements: The strategy to transform a function of `t` and `envRate` into a function of `t` only using `envRate` was adapted from `RPANDA`.

`rateShifts` A vector indicating the time of rate shifts in a step function. The first element must be the first or last time point for the simulation, i.e. `0` or `tMax`. Since functions in paleobuddy run from `0` to `tMax`, if `rateShifts` runs from past to present (meaning `rateShifts[2] < rateShifts[1]`), we take `tMax - rateShifts` as the shifts vector. Note that supplying `rateShifts` when `rate` is not a numeric vector of the same length will result in an error.

Value

A constant or time-varying function (depending on input) that can then be used as a rate in the other `paleobuddy` functions.

Author(s)

Bruno do Rosario Petrucci

References

Morlon H. et al (2016) RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution* 7: 589-597.

Examples

```
# first we need a time vector to use on plots
time <- seq(0, 50, 0.1)

###
# we can have a step function rate

# vector of rates
rate <- c(0.1, 0.2, 0.3, 0.2)

# vector of rate shifts
rateShifts <- c(0, 10, 20, 35)
# this could be c(50, 40, 30, 15) for equivalent results

# make the rate
r <- make.rate(rate, tMax = 50, rateShifts = rateShifts)

# plot it
plot(time, rev(r(time)),type = 'l', xlim = c(max(time), min(time)))

# note that this method of generating a step function rate is slower to
# numerically integrate

# it is also not possible a rate and a shifts vector and a time-series
# dependency, so in cases where one looks to run many simulations, or have a
```

```
# step function modified by an environmental variable, consider
# using ifelse() (see below)

###
# we can have an environmental variable (or any time-series)

# temperature data
data(temp)

# function
rate <- function(t, env) {
  return(0.05*env)
}

# make the rate
r <- make.rate(rate, envRate = temp)

# plot it
plot(time, rev(r(time)), type = 'l', xlim = c(max(time), min(time)))

###
# we can have a rate that depends on time AND temperature

# temperature data
data(temp)

# function
rate <- function(t, env) {
  return(0.001*exp(0.1*t) + 0.05*env)
}

# make a rate
r <- make.rate(rate, envRate = temp)

# plot it
plot(time, rev(r(time)), type = 'l', xlim = c(max(time), min(time)))

###
# as mentioned above, we could also use ifelse() to
# construct a step function that is modulated by temperature

# temperature data
data(temp)

# function
rate <- function(t, env) {
  return(ifelse(t < 10, 0.1 + 0.01*env,
                ifelse(t < 30, 0.2 - 0.005*env,
                       ifelse(t <= 50, 0.1 + 0.005*env, 0))))
}

# rate
r <- make.rate(rate, envRate = temp)
```

```
# plot it
plot(time, rev(r(time)), type = 'l', xlim = c(max(time), min(time)))

# while using ifelse() to construct a step function is more
# cumbersome, it leads to much faster numerical integration,
# so in cases where the method above is proving too slow,
# consider using ifelse() even if there is no time-series dependence

###
# make.rate will leave some types of functions unaltered

# constant rates
r <- make.rate(0.5)

# plot it
plot(time, rep(r, length(time)), type = 'l',
     xlim = c(max(time), min(time)))

###
# linear rates

# function
rate <- function(t) {
  return(0.01*t)
}

# create rate
r <- make.rate(rate)

# plot it
plot(time, rev(r(time)), type = 'l', xlim = c(max(time), min(time)))

###
# any time-varying function, really

# function
rate <- function(t) {
  return(abs(sin(t))*0.1 + 0.05)
}

# create rate
r <- make.rate(rate)

# plot it
plot(time, r(time), type = 'l')
```

---

paleobuddy *paleobuddy: Simulating diversification dynamics*

Description

paleobuddy provides users with flexible scenarios for species birth-death simulations. It also provides the possibility of generating phylogenetic trees (with extinct and extant species) and fossil records (with a number of preservation scenarios) from the same underlying process.

Birth-death simulation

Users have access to a large array of scenarios to use and combine for species birth-death simulation. The function bd.sim allows for constant rates, rates varying as a function of time, or time and/or an environmental variable, as well as age-dependent rates by using a shape parameter from a Weibull distribution (which can itself also be time-dependent). Extinction and speciation rates can be supplied independently, so that one can combine multiple types of scenarios for birth and death rates. The function find.lineages separates birth-death simulations into monophyletic clades so one can generate fossil records and phylogenies (see below) for clades with a specific mother species. This is particularly useful for simulations with multiple starting species. See ?bd.sim and ?find.lineages for more information.

All birth-death simulation functions return a sim object, which is a list of vectors containing speciation times, extinction times, status (extant or extinct) and parent identity for each species of the simulation. We supply methods for summarizing and printing sim objects in a more informative manner (see Visualization below). See ?sim for more information.

Fossil record simulation

The package provides users with a similarly diverse array of scenarios for preservation rates in generating fossil records from birth-death simulations. The function sample.clade accepts constant, time-varying, and environmentally dependent rates. Users might also supply a model describing the distribution of fossil occurrences over a species duration to simulate age-dependent sampling. See ?sample.clade for more information.

Phylogeny generation

We believe it is imperative to be able to generate fossil records and phylogenetic trees from the same underlying process, so the package provides make.phylo, a function that takes a simulation object of the form returned by bd.sim and generates a phylo object from the APE package. One can then use functions such as ape::plot.phylo and ape::drop.fossil to plot the phylogeny or analyze the phylogeny of extant species. Since APE is not required for any function in the package, it is a suggested but not imported package. Note that, as above, the function find.lineages allows users to separate clades with mother species of choice, the results of which can be passed to make.phylo to generate separate phylogenies for each clade. See ?make.phylo and ?find.lineages for more information.

Note: If a user wishes to perform the opposite operation - transform a phylo object into a sim object, perhaps to use paleobuddy for sampling on phylogenies generated by other packages, see ?phylo.to.sim.

Visualization

paleobuddy provides the user with a number of options for visualizing a sim object besides phylogenies. The sim object returned by birth-death simulation functions (see above) has summary

and plot methods. `summary(sim)` gives quantitative details of a `sim` objective, namely the total and extant number of species, and summaries of species durations and speciation times. `plot(sim)` plots births, deaths, and diversity through time for that realization. The function `draw.sim` draws longevities of species in the simulation, allowing for customization through the addition of fossil occurrences (which can be time points or ranges), and vertical order of the drawn longevities.

Utility functions

The package makes use of a few helper functions for simulation and testing that we make available for the user. `rexp.var` aims to emulate the behavior of `rexp`, the native R function for drawing an exponentially distributed variate, with the possibility of supplying a time-varying rate. The function also allows for a shape parameter, in which case the times drawn will be distributed as a Weibull, possibly with time-varying parameters, for age-dependent rates. `var.rate.div` calculates the expected diversity of a birth-death process with varying rates for any time period, which is useful when testing the birth-death simulation functions. Finally, `binner` takes a vector of fossil occurrence times and a vector of time boundaries and returns the number of occurrences within each time period. This is mostly for use in the `sample.clade` function. See `?rexp.var`, `?var.rate.div` and `?binner` for more information.

Author(s)

Bruno do Rosario Petrucci, Matheus Januario and Tiago B. Quental

Maintainer: Bruno do Rosario Petrucci <petrucci@iastate.edu>

See Also

Useful links:

- https://github.com/brpetrucci/paleobuddy
- Report bugs at https://github.com/brpetrucci/paleobuddy/issues

Examples

```
# here we present a quick example of paleobuddy usage
# for a more involved introduction, see the \code{overview} vignette

# make a vector for time
time <- seq(0, 10, 0.1)

# speciation rate
lambda <- function(t) {
  0.15 + 0.03*t
}

# extinction rate
mu <- 0.08

# these are pretty simple scenarios, of course
# check the examples in ?bd.sim for a more comprehensive review

# diversification
```

```
d <- function(t) {
  lambda(t) - mu
}

# calculate how many species we expect over 10 million years
div <- var.rate.div(rate = d, n0 = 1, t = time)
# note we are starting with 3 species (n0 = 3), but the user
# can provide any value - the most common scenario is n0 = 1

# plot it
plot(time, rev(div), type = 'l', main = "Expected diversity",
     xlab = "Time (My)", ylab = "Species",
     xlim = c(max(time), min(time)))

# we then expect around 9 species
# alive by the present, seems pretty good

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim(n0 = 1, lambda = lambda, mu = mu,
              tMax = 10, nFinal = c(20, Inf))
# nFinal controls the final number of species
# here we throw away simulations with less than 20 species generated

# draw longevities
draw.sim(sim)

# from sim, we can create fossil records for each species
# rho is the fossil sampling rate, see ?sample.clade
samp <- sample.clade(sim = sim, rho = 0.75, tMax = 10,
                     bins = seq(10, 0, -1))
# note 7 out of the 31 species did not leave a fossil - we can in this way
# simulate the incompleteness of the fossil record

# we can draw fossil occurrences as well, and order by extinction time
draw.sim(sim, fossils = samp, sortBy = "TE")

# take a look at the phylogeny
if (requireNamespace("ape", quietly = TRUE)) {
  ape::plot.phylo(make.phylo(sim), root.edge = TRUE)
  ape::axisPhylo()
}
```

---

phylo.to.sim *Converting a phylogeny in a paleobuddy object*

Description

Generates a sim object using a phylo object and some additional information (depending on other arguments). It is the inverse of the make.phylo function. Input is (1) a phylogeny, following an evolutionary Hennigian (sensu Ezard et al 2011) format (i.e., a fully bifurcating phylogeny), (2) information on the "mother lineage" of each tip in the phylogeny, (3) the status ("extant" or "extinct") of each lineage, (4) the stem age (or age of origination of the clade), and (5) the stem length (or time interval between the stem age and the first speciation event). The user can also choose if the event dating should be done from root to tips or from tips-to-root. The function returns a sim object (see ?sim). The function does not accept more than one species having NA as parent (which is interpreted as if there were no single common ancestor in the phylogeny). In that case, use find.lineages first.

Usage

```
phylo.to.sim(
  phy,
  mothers,
  extant,
  dateFromPresent = TRUE,
  stemAge = NULL,
  stemLength = NULL
)
```

Arguments

phy	A phylo object, which may contain only extant or extant and extinct lineages.
mothers	Vector containing the mother of each tip in the phylogeny. First species' mother should be NA. See details below.
extant	Logical vector indicating which lineages are extant and extinct.
dateFromPresent	Logical vector indicating if speciation/extinction events should be dated from present-to-root (TRUE, default value) or from root-to-present. As it is impossible to date "from present" without a living lineage, it is internally set to FALSE and prints a message in the prompt if there are no extant species.
stemAge	Numeric vetor indicating the age, in absolute geological time (million years ago), when the first lineage of the clade originated. It is not needed when dateFromPresent is TRUE and stemLength is provided, or when phy has a root.edge. This argument is required if dateFromPresent is FALSE.
stemLength	Numeric vector indicating the time difference between the stemAge and the first speciation event of the group. This argument is required if dateFromPresent is FALSE, but users have no need to assign values in this parameter if phy has a $root.edge, which is taken by the function as the stemLength value.

Details

See Details below for more information on each argument.

Mothers:

The function needs the indication of a mother lineage for every tip in the phylogeny but one (which is interpreted as the first known lineage in the clade, and should have NA as the mother). This assignment might be straightforward for simulations (as in the examples section below), but is a non-trivial task for empirical phylogenies. As there are many ways to assign impossible combinations of motherthood, the function does not return any specific error message if the provided motherhood does not map to possible lineages given the phylogeny. Instead, the function tends to crash when an "impossible" motherhood is assigned, but this is not guaranteed to happen because the set of "impossible" ways to assign motherhood is vast, and therefore has not allowed for a test of every possibility. If the function crashes when all lineages have reasonable motherhood, users should submit an issue report at https://github.com/brpetrucci/paleobuddy/issues.

Dating:

Phylogenies store the relative distances between speciation (and possibly extinction) times of each lineage. However, to get absolute times for those events (which are required to construct the output of this function), users should provide a moment in absolute geological time to position the phylogeny. This could be (1) the present, which is used as reference in the case at least one lineage in the phylogeny is extant (i.e., default behavior of the function), or (2) some time in the past, which is the stemAge parameter. Those two possible dating methods are used by setting dateFromPresent to TRUE or FALSE. If users have extant lineages in their phylogeny but do not have a reasonable value for stemAge, they are encouraged to use present-to-root dating (dateFromPresent = TRUE), as in that case deviations in the value of stemLength will only affect the speciation time of the first lineage of the clade. In other words, when dateFromPresent is set to FALSE, user error in stemAge or stemLength will bias the absolute (but not the relative) dating of all nodes in the phylogeny.

Value

A sim object. For details, see ?sim. Items in the object follow their tip assignment in the phylogeny.

Author(s)

Matheus Januario.

References

Ezard, T. H., Pearson, P. N., Aze, T., & Purvis, A. (2012). The meaning of birth and death (in macroevolutionary birth-death models). *Biology letters*, 8(1), 139-142.

Examples

```
# to check the usage of the function, let us make sure it transforms a
# phylogeny generated with make.phylo back into the original simulation

###
# birth-death process

# set seed
set.seed(1)

# run simulation
sim <- bd.sim(1, lambda = 0.3, mu = 0.1, tMax = 10, nFinal = c(10, Inf))
```

```
# convert birth-death into phylo
phy <- make.phylo(sim)

# convert phylo into a sim object again
res <- phylo.to.sim(phy = phy, extant = sim$EXTANT, mothers = sim$PAR)

# test if simulation and converted object are the same
all.equal(sim, res)

###
# birth-death process with extinct lineages:
# set seed
set.seed(1)

# run simulation
sim <- bd.sim(1, lambda = 0.1, mu = 0.3, tMax = 10, nFinal = c(2, 4))

# convert birth-death into phylo
phy <- make.phylo(sim)

# convert phylo into a sim object again
res <- phylo.to.sim(phy = phy, extant = sim$EXTANT, mothers = sim$PAR, stemAge = max(sim$TS))

# test if simulation and converted object are the same
all.equal(sim, res)

###
# pure birth process

# set seed
set.seed(1)

# run simulation
sim <- bd.sim(1, lambda = 0.2, mu = 0, tMax = 10, nFinal = c(10, Inf))

# convert birth-death into phylo
phy <- make.phylo(sim)

# convert phylo into birth-death again
# note we can supply optional arguments, see description above
res <- phylo.to.sim(phy = phy, extant = sim$EXTANT, mothers = sim$PAR,
                stemAge = 10, stemLength = (10 - sim$TS[2]))

# testing if simulation and converted object are the same
all.equal(sim, res)
```

---

rexp.var *General rate exponential and Weibull waiting times*

Description

Generates a waiting time following an exponential or Weibull distribution with constant or varying rates. Output can be used as the waiting time to an extinction, speciation, or fossil sampling event. Allows for an optional shape parameter, in which case `rate` will be taken as a Weibull scale. Allows for further customization by restricting possible waiting time outputs with arguments for (1) current time, to consider only the rates starting at that time, (2) maximum time, to bound the output and therefore allow for faster calculations if one only cares about waiting times lower than a given amount, and (3) speciation time, necessary to scale rates in the case where the output is to follow a Weibull distribution, i.e. for age-dependent processes. This function is used in birth-death and sampling functions, but can also be convenient for any user looking to write their own code requiring exponential or Weibull distributions with varying rates.

Usage

```
rexp.var(n, rate, now = 0, tMax = Inf, shape = NULL, TS = 0, fast = FALSE)
```

Arguments

`n`	The number of waiting times to return. Usually 1, but we allow for a higher `n` to be consistent with the `rexp` function.
`rate`	The rate parameter for the exponential distribution. If `shape` is not `NULL`, `rate` is a scale for a Weibull distribution. In both cases we allow for any time-varying function. Note `rate` can be constant.
`now`	The current time. Needed if one wants to consider only the interval between the current time and the maximum time for the time-varying rate. Note this does means the waiting time is `>= now`, so we also subtract `now` from the result before returning. The default is `0`.
`tMax`	The simulation ending time. If the waiting time would be too high, we return `tMax + 0.01` to signify the event never happens, if `fast == TRUE`. Otherwise we return the true waiting time. By default, `tMax` will be `Inf`, but if `FAST == TRUE` one must supply a finite value.
`shape`	Shape of the Weibull distribution. Can be a `numeric` for constant shape or a `function(t)` for time-varying. When smaller than one, rate will decrease along species' age (negative age-dependency). When larger than one, rate will increase along each species' age (positive age-dependency). Default is `NULL`, so the function acts as an exponential distribution. For `shape != NULL` (including when equal to one), `rate` will be considered a scale (= 1/rate), and `rexp.var` will draw a Weibull distribution instead of an exponential. This means Weibull(rate, 1) = Exponential(1/rate). Notice even when `Shape != NULL`, `rate` may still be time-dependent. Note: Time-varying shape is within expectations for most cases, but if it is lower than 1 and varies too much (e.g. `0.5 + 0.5*t`), it can be slightly biased for higher waiting times due to computational error. Slopes (or equivalent, since it can be any function of time) of the order of 0.01 are advisable. It rarely also displays small biases for abrupt variations. In both cases, error is still quite low for the purposes of the package.

Note: We do not test for shape < 0 here since as we allow shape to be a function this would severely slow the rest of the package. It is tested on the birth-death functions, and the user should make sure not to use any functions that become negative eventually.

`TS` Speciation time, used to account for the scaling between simulation and species time. The default is `0`. Supplying a `TS > now` will return an error.

`fast` If set to `FALSE`, waiting times larger than `tMax` will not be thrown away. This argument is needed so one can test the function without bias.

Value

A vector of waiting times following the exponential or Weibull distribution with the given parameters.

Author(s)

Bruno do Rosario Petrucci.

Examples

```
###
# let us start by checking a simple exponential variable

# rate
rate <- 0.1

# set seed
set.seed(1)

# find the waiting time
t <- rexp.var(n = 1, rate)
t

# this is the same as t <- rexp(1, rate)

###
# now let us try a linear function for the rate

# rate
rate <- function(t) {
  return(0.01*t + 0.1)
}

# set seed
set.seed(1)

# find the waiting time
t <- rexp.var(n = 1, rate)
t

###
```

```
# what if rate is exponential?

# rate
rate <- function(t) {
  return(0.01 * exp(0.1*t) + 0.02)
}

# set seed
set.seed(1)

# find the waiting time
t <- rexp.var(n = 1, rate)
t

###
# if we give a shape argument, we have a Weibull distribution

# scale - note that this is equivalent to 1/rate if shape were NULL
rate <- 2

# shape
shape <- 1

# speciation time
TS <- 0

# set seed
set.seed(1)

# find the vector of waiting time
t <- rexp.var(n = 1, rate, shape = shape, TS = TS)
t

###
# when shape = 1, the Weibull is an exponential, we could do better

# scale
rate <- 10

# shape
shape <- 2

# speciation time
TS <- 0

# set seed
set.seed(1)

# find the vector of waiting times - it doesn't need to be just one
t <- rexp.var(n = 5, rate, shape = shape, TS = TS)
t

###
```

```
# shape can be less than one, of course

# scale
rate <- 10

# shape
shape <- 0.5

# note we can supply now (default 0) and tMax (default Inf)

# now matters when we wish to consider only waiting times
# after that time, important when using time-varying rates
now <- 3

# tMax matters when fast = TRUE, so that higher times will be
# thrown out and returned as tMax + 0.01
tMax <- 40

# speciation time - it will be greater than 0 frequently during a
# simulation, as it is used to represent where in the species life we
# currently are and rescale accordingly
TS <- 2.5

# set seed
set.seed(1)

# find the vector of waiting times
t <- rexp.var(n = 10, rate, now, tMax,
              shape = shape, TS = TS, fast = TRUE)
t

# note how some results are tMax + 0.01, since fast = TRUE

###
# both rate and shape can be time varying for a Weibull

# scale
rate <- function(t) {
  return(0.25*t + 5)
}

# shape
shape <- 3

# current time
now <- 0

# maximum time to check
tMax <- 40

# speciation time
TS <- 0
```

```
# set seed
set.seed(1)

# find the vector of waiting times
t <- rexp.var(n = 5, rate, now, tMax,
              shape = shape, TS = TS, fast = TRUE)
t
```

`sample.clade` *General rate fossil sampling*

Description

Generates occurrence times or time ranges (as most empirical fossil occurrences) for each of the desired species using a Poisson process. Allows for the Poisson rate to be (1) a constant, (2) a function of time, (3) a function of time and a time-series (usually environmental) variable, or (4) a vector of numbers (rates in a step function). Allows for age-dependent sampling with a parameter for a distribution representing the expected occurrence number over a species duration. Allows for further flexibility in rates by a shift times vector and environmental matrix parameters. Finally, allows for the simulation of trait-dependent fossil sampling when trait value information is supplied.

Usage

```
sample.clade(
  sim,
  rho,
  tMax,
  S = NULL,
  envR = NULL,
  rShifts = NULL,
  returnTrue = TRUE,
  returnAll = FALSE,
  bins = NULL,
  adFun = NULL,
  ...
)
```

Arguments

`sim`	A `sim` object, containing extinction times, speciation times, parent, and status information (extant or extinct) for each species in the simulation. See `?sim`.
`rho`	Sampling rate (per species per million years) over time. It can be a `numeric` describing a constant rate, a `function(t)` describing the variation in sampling over time `t`, a `function(t, env)` describing the variation in sampling over time following both time AND a time-series, usually an environmental variable (see `envR`), or a `vector` of rates, corresponding to each rate between sampling rate

shift times times (see rShifts), describing an episodic model of fossil sampling. If adFun is supplied, it will be used to find the number of occurrences during the species duration, and a normalized rho*adFun will determine their distribution along the species duration. Note that rho should always be greater than or equal to zero.

tMax — The maximum simulation time, used by rexp.var. A sampling time greater than tMax would mean the occurrence is sampled after the present, so for consistency we require this argument. This is also required to ensure time follows the correct direction both in the Poisson process and in the output.

S — A vector of species numbers to be sampled. The default is all species in sim. Species not included in S will not be sampled by the function.

envR — A data frame containing time points and values of an environmental variable, like temperature, for each time point. This will be used to create a sampling rate, so rho must be a function of time and said variable if envR is not NULL. Note paleobuddy has two environmental data frames, temp and co2. See RPANDA for more examples.

rShifts — Vector of rate shifts. First element must be the starting time for the simulation (0 or tMax). It must have the same length as lambda. c(0, x, tMax) is equivalent to c(tMax, tMax - x, 0) for the purposes of make.rate.

returnTrue — If set to FALSE, it will contain the occurrence times as ranges. In this way, we simulate the granularity presented by empirical fossil records. If returnTrue is TRUE, this is ignored.

returnAll — If set to TRUE, returns both the true sampling time and age ranges. Default is FALSE.

bins — A vector of time intervals corresponding to geological time ranges. It must be supplied if returnTrue or returnAll is TRUE.

adFun — A density function representing the age-dependent preservation model. It must be a density function, and consequently integrate to 1 (though this condition is not verified by the function). If not provided, a uniform distribution will be used by default. The function must also have the following properties:

- Return a vector of preservation densities for each time in a given vector t in geological time.
- Be parameterized in the absolute geological time associated to each moment in age (i.e. age works relative to absolute geological time, in Mya - in other words, the convention is TS > 0). The function *does not* directly use the lineage's age (which would mean that TS = 0 for all species whenever they are born). Because of this, it is assumed to go from tMax to 0, as opposed to most functions in the package.
- Should be limited between s (i.e. the lineage's speciation/birth) and e (i.e. the lineage's extinction/death), with s > e. It is possible to assign parameters in absolute geological time (see third example) and relative to age as long as this follows the convention of age expressed in absolute geological time (see fourth example).
- Include the arguments t, s, e and sp. The argument sp is used to pass species-specific parameters (see examples), allowing for dFun to be species-inhomogeneous.

... Additional parameters used by adFun. See examples.

Details

Optionally takes a vector of time bins representing geologic periods, if the user wishes occurrence times to be represented as a range instead of true points.

The age-dependent preservation function assumes that all extant species at the end of the simulations have TE = 0 (i.e., the function assumes all extant species got extinct exaclty when the simulation ended. This might create distortion for some adFun - especially in the case of bell-shaped functions. As interpretations of what age-dependent preservation mean to species alive at the end of the simulation, we recommend users to implement their own preservation functions for the species that are extant at the end of the simulation.

Value

A data.frame containing species names/numbers, whether each species is extant or extinct, and the true occurrence times of each fossil, a range of occurrence times based on bins, or both.

Author(s)

Matheus Januario and Bruno do Rosario Petrucci.

Examples

```
# vector of times
time <- seq(10, 0, -0.1)

###
# we can start with a constant case

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# sampling rate
rho <- 2

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho, tMax = 10,
                        bins = bins, returnTrue = FALSE)

# draw simulation with fossil occurrences as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

###
# sampling can be any function of time
```

```
# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# sampling rate
rho <- function(t) {
  return(2 - 0.15*t)
}

# plot sampling function
plot(x = time, y = rho(time), type = "l",
     ylab = "Preservation rate",
     xlab = "Time since the start of the simulation (My)")
# note for these examples we do not reverse time in the plot
# see other functions in the package for examples where we do

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho, tMax = 10,
                        bins = bins, returnTrue = FALSE)

# draw simulation with fossil occurrences as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

###
# now we can try a step function rate
# not running because it takes a long time

## Not run:
# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# we will use the less efficient method of creating a step function
# one could instead use ifelse()

# rates vector
rList <- c(2, 5, 0.5)

# rate shifts vector
rShifts <- c(0, 4, 8)

# make it a function so we can plot it
rho <- make.rate(rList, 10, rateShifts = rShifts)

# plot sampling function
```

```
plot(x = time, y = rho(time), type = "l",
     ylab = "Preservation rate",
     xlab = "Time since the start of the simulation (My)")

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho = rList, rShifts = rShifts, tMax = 10,
                        bins = bins, returnTrue = FALSE)

# draw simulation with fossil occurrences as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

## End(Not run)

###
# finally, sample.clade also accepts an environmental variable

# get temperature data
data(temp)

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# rho will be temperature dependent
envR <- temp

# function describing environmental dependence
r_t <- function(t, env) {
  return(((env) / 12) ^ 6)
}

# make it a function so we can plot it
rho <- make.rate(r_t, tMax = tMax, envRate = envR)

# plot sampling function
plot(x = time, y = rho(time), type = "l",
     ylab = "Preservation rate",
     xlab = "Time since the start of the simulation (My)")

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho = r_t, envR = envR, tMax = 10, bins = bins)
# now we record the true time of fossil occurrences

# draw simulation with fossil occurrences as time points
draw.sim(sim, fossils = fossils)

# note that any techniques used in examples for ?bd.sim to create more
# complex mixed scenarios can be used here as well
```

```
###
# sampling can also be age-dependent

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# sampling rate
rho <- 3

# here we will use the PERT function. It is described in:
# Silvestro et al 2014

# age-dependence distribution
# note that a and b define the beta distribution used, and can be modified
dPERT <- function(t, s, e, sp, a = 3, b = 3, log = FALSE) {
  # check if it is a valid PERT
  if (e >= s) {
   message("There is no PERT with e >= s")
   return(rep(NaN, times = length(t)))
  }

  # find the valid and invalid times
  id1 <- which(t <= e | t >= s)
  id2 <- which(!(t <= e | t >= s))
  t <- t[id2]

  # initialize result vector
  res <- vector()

  # if user wants a log function
  if (log) {
   # invalid times get -Inf
   res[id1] <- -Inf

   # valid times calculated with log
   res[id2] <- log(((s - t) ^ 2) * ((-e + t) ^ 2) /
                 ((s - e) ^ 5 * beta(a, b)))
  }

  # otherwise
  else{
   res[id1] <- 0

   res[id2] <- ((s - t) ^ 2) * ((-e + t) ^ 2) / ((s - e) ^ 5 * beta(a, b))
  }

  return(res)
}
```

```
# plot it for an example species who lived from 10 to 5 million years ago
plot(time, rev(dPERT(t = time, s = 10, e = 5, a = 1)),
     main = "Age-dependence distribution",
     xlab = "Species age (My)", ylab = "Density",
     xlim = c(0, 5), type = "l")

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho, tMax = 10, adFun = dPERT, bins = bins,
                        returnAll = TRUE)
# can use returnAll to get occurrences as both time points and ranges

# draw simulation with fossil occurrences as time points
draw.sim(sim, fossils = fossils)
# the warning is to let you know the ranges won't be used

# and also as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

###
# we can have more parameters on adFun

# sampling rate
rho <- function(t) {
 return(1 + 0.5*t)
}
# since here rho is time-dependent, the function finds the
# number of occurrences using rho, and their distribution
# using a normalized rho * adFun

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# here we will use the triangular distribution

# age-dependence distribution
dTRI <- function(t, s, e, sp, md) {
  # make sure it is a valid TRI
  if (e >= s) {
   message("There is no TRI with e >= s")
   return(rep(NaN, times = length(t)))
  }

  # another condition we must check
  if (md < e | md > s) {
   message("There is no TRI with md outside [s, e] interval")
   return(rep(NaN, times = length(t)))
  }
```

```
  # needed to vectorize the function:
  id1 <- which(t >= e & t < md)
  id2 <- which(t == md)
  id3 <- which(t > md & t <= s)
  id4 <- which( !(1:length(t) %in% c(id1,id2,id3)))

  # actually vetorizing function
  res <- vector()

  # (t >= e & t < md)
  res[id1] <- (2*(t[id1] - e)) / ((s - e) * (md - e))

  # (t == md)
  res[id2] <- 2 / (s - e)

  # (md < t & t <= s)
  res[id3] <- (2*(s - t[id3])) / ((s - e) * (s - md))

  # outside function's limits
  res[id4] <- 0

  return(res)
}

# set mode at 8
md <- 8

# plot it for an example species who lived from 10mya to the present
plot(time, rev(dTRI(time, 10, 5, 1, md)),
     main = "Age-dependence distribution",
     xlab = "Species age (My)", ylab = "Density",
     xlim = c(0, 5), type = "l")

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences for the first species
fossils <- sample.clade(sim, rho, tMax = 10, S = 1, adFun = dTRI,
                        bins = bins, returnTrue = FALSE, md = md)
# note we provide the peak for the triangular sampling as an argument
# here that peak is assigned in absolute geological, but
# it usually makes more sense to express this in terms
# of age (a given percentile of the age, for instance) - see below

# draw simulation with fossil occurrences as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

###
# we can also have a hat-shaped increase through the duration of a species
# with more parameters than TS and TE, but with the parameters relating to
# the relative age of each lineage
```

```
# sampling rate
rho <- function(t) {
 return(1 + 0.1*t)
}

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# age-dependence distribution, with the "mde" of the triangle
# being exactly at the last quarter of the duration of EACH lineage
dTRImod1 <- function(t, s, e, sp) {
  # note that now we don't have the "md" parameter here,
  # but it is calculated inside the function

  # check if it is a valid TRI
  if (e >= s) {
   message("There is no TRI with e >= s")
   return(rep(NaN, times = length(t)))
  }

  # calculate md
  md <- ((s - e) / 4) + e
  # md is at the last quarter of the duration of the lineage

  # please note that the same logic can be used to sample parameters
  # internally in the function, running for instance:
  # md <- runif (n = 1, min = e, max = s)

  # check it is a valid md
  if (md < e | md > s) {
   message("There is no TRI with md outside [s, e] interval")
   return(rep(NaN, times = length(t)))
  }

  # needed to vectorize function
  id1 <- which(t >= e & t < md)
  id2 <- which(t == md)
  id3 <- which(t>md & t <= s)
  id4 <- which( !(1:length(t) %in% c(id1,id2,id3)))

  # vectorize the function
  res<-vector()

  res[id1] <- (2 * (t[id1] - e)) / ((s - e) * (md - e))
  res[id2] <- 2 / (s - e)
  res[id3] <- (2 * (s - t[id3])) / ((s - e) * (s - md))
  res[id4] <- 0

  return(res)
}
```

```
# plot for a species living between 10 and 0 mya
plot(time, rev(dTRImod1(time, 10, 0, 1)),
    main = "Age-dependence distribution",
    xlab = "Species age (My)", ylab = "Density",
    xlim = c(0, 10), type = "l")

# sample first two species
fossils <- sample.clade(sim = sim, rho = rho, tMax = 10, adFun = dTRImod1)

# draw simulation with fossil occurrences as time points
draw.sim(sim, fossils = fossils)

# here, we fix md at the last quarter
# of the duration of the lineage

###
# the parameters of adFun can also relate to each specific lineage,
# which is useful when using variable parameters for each species
# to keep track of those parameters after the sampling is over

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# sampling rate
rho <- 3

# get the par and par1 vectors

# get mins vector
minsPar <- ifelse(is.na(sim$TE), 0, sim$TE)

# a random time inside each species' duration
par <- runif(n = length(sim$TE), min = minsPar, max = sim$TS)

# its complement to the middle of the lineage's age.
par1 <- (((sim$TS - minsPar) / 2) + minsPar) - par
# note that the interaction between these two parameters creates a
# deterministic parameter, but inside the function one of them ("par")
# is a random parameter

dTRImod2 <- function(t, s, e, sp) {
  # make sure it is a valid TRI
  if (e >= s) {
   message("There is no TRI with e >= s")
   return(rep(NaN, times = length(t)))
  }

  # md depends on parameters
  md <- par[sp] + par1[sp]
```

```
  # check that md is valid
  if (md < e | md > s) {
   message("There is no TRI with md outside [s, e] interval")
   return(rep(NaN, times = length(t)))
  }

  id1 <- which(t >= e & t < md)
  id2 <- which(t == md)
  id3 <- which(t > md & t <= s)
  id4 <- which(!(1:length(t) %in% c(id1,id2,id3)))

  res <- vector()

  res[id1] <- (2*(t[id1] - e)) / ((s - e) * (md - e))
  res[id2] <- 2 / (s - e)
  res[id3] <- (2*(s - t[id3])) / ((s - e) * (s - md))
  res[id4] <- 0

  return(res)
}

# plot for a species living between 10 and 0 mya
plot(time, rev(dTRImod2(time, 10, 0, 1)),
    main = "Age-dependence distribution",
    xlab = "Species age (My)", ylab = "Density",
    xlim = c(0, 10), type = "l")

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho, tMax = 10, adFun = dTRImod2, bins = bins,
                        returnTrue = FALSE)

# draw simulation with fossil occurrences as time ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

###
# we can also have a mix of age-independent and age-dependent
# sampling in the same simulation

# set seed
set.seed(1)

# simulate a group
sim <- bd.sim(n0 = 1, lambda = 0.1, mu = 0.1, tMax = 10)

# sampling rate
rho <- 7

# define a uniform to represente age-independence

# age-dependence distribution (a uniform density distribution in age)
# in the format that the function needs
custom.uniform <- function(t, s, e, sp) {
```

```
  # make sure it is a valid uniform
  if (e >= s) {
   message("There is no uniform function with e >= s")
   return(rep(NaN, times = length(t)))
  }

  res <- dunif(x = t, min = e, max = s)

  return(res)
}

# PERT as above
dPERT <- function(t, s, e, sp, a = 3, b = 3, log = FALSE) {
  # check if it is a valid PERT
  if (e >= s) {
   message("There is no PERT with e >= s")
   return(rep(NaN, times = length(t)))
  }

  # find the valid and invalid times
  id1 <- which(t <= e | t >= s)
  id2 <- which(!(t <= e | t >= s))
  t <- t[id2]

  # initialize result vector
  res <- vector()

  # if user wants a log function
  if (log) {
   # invalid times get -Inf
   res[id1] <- -Inf

   # valid times calculated with log
   res[id2] <- log(((s - t) ^ 2) * ((-e + t) ^ 2) /
                ((s - e) ^ 5 * beta(a, b)))
  }

  # otherwise
  else{
   res[id1] <- 0

   res[id2] <- ((s - t) ^ 2) * ((-e + t) ^ 2) / ((s - e) ^ 5 * beta(a, b))
  }

  return(res)
}

# actual age-dependency defined by a mix
dPERTAndUniform <- function(t, s, e, sp) {
  return(
    ifelse(t > 5, custom.uniform(t, s, e, sp),
           dPERT(t, s, e, sp))
  )
```

```
}
# starts out uniform, then becomes PERT
# after 5my (in absolute geological time)

# plot it for an example species who lived from 10 to 0 million years ago
plot(time, rev(dPERTAndUniform(time, 10, 0, 1)),
    main = "Age-dependence distribution",
    xlab = "Species age (My)", ylab = "Density",
    xlim = c(0, 10), type = "l")

# bins for fossil ranges
bins <- seq(from = 10, to = 0, by = -1)

# simulate fossil occurrences data frame
fossils <- sample.clade(sim, rho, tMax = 10, adFun = dPERTAndUniform,
                        bins = bins, returnTrue = FALSE)

# draw simulation with fossil occurrences as ranges
draw.sim(sim, fossils = fossils, fossilsFormat = "ranges")

# note how occurrences cluster close to the speciation time of
# species 1, but not its extinction time, since around 5mya
# the PERT becomes the effective age-dependence distribution
```

`sample.clade.traits` *Trait-dependent fossil sampling*

Description

Generates occurrence times or time ranges (as most empirical fossil occurrences) for each of the desired species using a Poisson process. Poisson rate should be dependent on some discrete trait, the value of which for each species will be supplied using the parameter `traits`. Rate can be dependent on observed traits only, or on a combination of observed and hidden traits (in which case the supplied trait data frame `traits` should have all possible states, observed or hidden, see examples for more details).

Usage

```
sample.clade.traits(
  sim,
  rho,
  tMax,
  traits,
  nFocus = 1,
  nStates = 2,
  nHidden = 1,
  S = NULL,
  returnTrue = TRUE,
```

```
  returnAll = FALSE,
  bins = NULL
)
```

Arguments

sim	A sim object, containing extinction times, speciation times, parent, and status information (extant or extinct) for each species in the simulation. See ?sim.
rho	Sampling rate (per species per million years) over time. It is a vector of rates, corpointEstimatesponding to the value of the rate for each value of the traits encoded in the traits parameter. It should therefore be of length nStates * nHidden. Note that rho should always be greater than or equal to zero.
tMax	The maximum simulation time, used by rexp.var. A sampling time greater than tMax would mean the occurrence is sampled after the present, so for consistency we require this argument. This is also required to ensure time follows the correct direction both in the Poisson process and in the output.
traits	List of trait data frames, usually one of the returns of bd.sim. traits[[i]][[j]] should corpointEstimatespond to the jth trait data frame for species i. The data frames contain the following columns value A vector of trait values the species took at specific intervals of time. max A vector of time values corpointEstimatesponding to the upper bound of each interval. min A vector of time values corpointEstimatesponding to the lower bound of each interval
nFocus	Trait of focus, i.e. the one that rho depends on. Note that traits can have multiple trait data frames per species, but only one of the simulated traits can affect fossil sampling rates. E.g. if nFocus = 1, then the first trait data frame per species will be used to simulate fossil occurrences.
nStates	Number of possible states for categorical trait. The range of values will be assumed to be (0, nStates - 1).
nHidden	Number of hidden states for categorical trait. Default is 1, in which case there are no added hidden traits. Total number of states is then nStates * nHidden. States will then be set to a value in the range of (0, nStates - 1) to simulate that hidden states are hidden. This is done by setting the value of a state to the remainder of state / nStates. E.g. if nStates = 2 and nHidden = 3, possible states during simulation will be in the range (0, 5), but states (2, 4) (corpointEstimatesponding to (0B, 0C) in the nomenclature of the original HiSSE reference) will be set to 0, and states (3, 5) (corpointEstimatesponding to (1B, 1C)) to 1. Note that since the traits is supplied as a parameter, the user must ensure that all states from 0 to nStates * nHidden - 1 are reppointEstimatesented in the trait information. See examples for more details on how to properly run hidden-states fossil sampling simulations.
S	A vector of species numbers to be sampled. The default is all species in sim. Species not included in S will not be sampled by the function.

returnTrue	If set to FALSE, it will contain the occurrence times as ranges. In this way, we simulate the granularity presented by empirical fossil records. If returnTrue is TRUE, this is ignored.
returnAll	If set to TRUE, returns both the true sampling time and age ranges. Default is FALSE.
bins	A vector of time intervals corresponding to geological time ranges. It must be supplied if returnTrue or returnAll is TRUE.

Details

Optionally takes a vector of time bins reppointEstimatesenting geologic periods, if the user wishes occurrence times to be reppointEstimatesented as a range instead of true points.

Value

A data.frame containing species names/numbers, whether each species is extant or extinct, and the true occurrence times of each fossil, a range of occurrence times based on bins, or both. Also a list object with the trait data frames describing the trait value for each species at each specified interval. Note that this list will only be different from the supplied traits parameter if nHidden > 1, in which case it will transform hidden traits into observed ones (see details for parameter nHidden).

Author(s)

Bruno do Rosario Petrucci.

Examples

```
###
# first a simple BiSSE simulation, with
# binary state-dependent fossil sampling

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, highe for state 0
mu <- c(0.06, 0.03)

# number of traits and states (1 binary trait)
nTraits <- 1
nStates <- 2

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates
```

```
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax = tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# now a fossil sampling rate, with higher rate for state 1
rho <- c(0.5, 1)

# run fossil sampling
fossils <- sample.clade.traits(sim$SIM, rho, tMax, sim$TRAITS)

# draw simulation with fossil occurrences as time points
draw.sim(sim$SIM, traits = sim$TRAITS,
         fossils = fossils$FOSSILS, traitLegendPlacement = "bottomleft")

###
# can also run a HiSSE model, with hidden traits having an effect on rates

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 20

# speciation, higher for state 1A, highest for 1B
lambda <- c(0.1, 0.2, 0.1, 0.3)

# extinction, lowest for 0B
mu <- c(0.03, 0.03, 0.01, 0.03)

# number of traits and states--in this case, we just run with 4 observed
# states, so that our traits data frames will include that info for sampling
nTraits <- 1
nStates <- 4

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates. Only one transition
# is allowed at a time, i.e. 0A can go to 0B and 1A,
# but not to 1B, and similarly for others
Q <- list(matrix(c(0, 0.1, 0.1, 0,
                   0.1, 0, 0, 0.1,
                   0.1, 0, 0, 0.1,
                   0, 0.1, 0.1, 0), ncol = 4, nrow = 4))

# set seed
set.seed(1)
```

```
# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates,
                     X0 = X0, Q = Q, nFinal = c(2, Inf))
# note how sim$TRAITS contains states 2 and 3, even though this
# is a HiSSE model, because we need that information to run hidden states
# on fossil sampling as well (see below)

# now a fossil sampling rate, with higher rate for state 1A,
# and highest yet for state 1B
rho <- c(0.5, 1, 0.5, 2)

# bins for fossil occurrences
bins <- seq(tMax, 0, -1)

# run fossil sampling, with returnAll = TRUE to include ranges
fossils <- sample.clade.traits(sim$SIM, rho, tMax, sim$TRAITS,
                               nStates = 2, nHidden = 2,
                               returnAll = TRUE, bins = bins)
# note how fossils$TRAITS only contains trait values 0 and 1, similar to
# if we had ran bd.sim.traits with nHidden = 2, since sample.clade.traits
# did the job of transforming observed into hidden states

# draw simulation with fossil occurrences as time points AND ranges
draw.sim(sim$SIM, traits = sim$TRAITS, fossils = fossils$FOSSILS,
         fossilsFormat = "all", traitLegendPlacement = "bottomleft")
# note that there are a lot of fossils, so the ranges are difficult to see

# see ?sample.clade for further examples using different return types
# (fossil ranges etc.), and ?bd.sim.traits for more examples using
# higher numbers of states (hidden or observed), though in
# sample.clade.traits we always have only one trait of focus
```

`sim` *Details, generics, and methods for the* `sim` *class*

Description

The `sim` class is a frequent return and input argument for functions in paleobuddy. It contains the following four elements.

`TE` Numeric vector of extinction times, with `NA` as the time of extinction for extant species.

`TS` Numeric vector of speciation times, with `tMax` as the time of speciation for species that started the simulation.

`PAR` Numeric vector of parents. Species that started the simulation have `NA`, while species that were generated during the simulation have their parent's number. Species are numbered as they are born.

EXTANT Vector of logicals representing whether each species is extant.

Here we declare useful generics and methods for sim objects.

Usage

```
is.sim(sim)

## S3 method for class 'sim'
print(x, ...)

## S3 method for class 'sim'
head(x, ...)

## S3 method for class 'sim'
tail(x, ...)

## S3 method for class 'sim'
summary(object, ...)

## S3 method for class 'sim'
plot(x, ...)

sim.counts(sim, t)
```

Arguments

sim, x, object	Object of class "sim"
...	Further arguments inherited from generics.
t	Time t (in Mya). Used for counting and/or plotting births, deaths and species number.

Details

is.sim A sim object must contain 4 members (usually vectors for extinction times, speciation times, species' parents and status), and all of these must have the correct length (i.e. same as all the others) and types. We do not utilize the members' order inside sim for our tests, since they are accessed with the $ operator and therefore the order is irrelevant.

print.sim The printing of a sim object is formatted into a more straightforward and informative sequence manner. We provide details only for the first few species, since otherwise this print could be overwhelming for simulations with 10+ species.

head.sim Selects only a number of species from the beginning of a sim object.

tail.sim Selects only a number of species from the end of a sim object.

summary.sim Quantitative details on the sim object. Prints the number of species, number of extant species, summary of durations and speciation waiting times, in case there are more than one species.

plot.sim Plots births, deaths, and diversity through time for the sim object.

sim.counts Calculates the births, deaths, and diversity for a sim at time t.

`temp` *Cenozoic temperature data*

Description

Temperature data during the Cenozoic. Modified from the `InfTemp` data set in RPANDA, originally inferred from delta O18 measurements. Inverted so lower times represent time since first measurement, to be in line with the past-to-present convention of most time-dependent functions in paleobuddy.

Usage

```
data(temp)
```

Format

A data frame with 17632 rows and 2 variables:

t A numeric vector representing time since the beginning of the data frame age, approximately 67 million years ago, in million years. We set this from past to present as opposed to present to past since birth-death functions in `paleobuddy` consider time going in the former direction. Hence `t = 0` represents the time point at 67.5173mya, while `t = 67.5173` represents the present.

temperature A numeric vector representing temperature in degrees celsius corresponding to time `t`. Note there might be more than one temperature for each time `t` given the resolution of the data set.

Source

https://github.com/hmorlon/PANDA

References

Morlon H. et al (2016) RPANDA: an R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution* 7: 589-597.

Epstein, S. et al (1953) Revised carbonate-water isotopic temperature scale *Geol. Soc. Am. Bull.* 64: 1315-1326.

Zachos, J.C. et al (2008) An early Cenozoic perspective on greenhouse warming and carbon-cycle dynamics *Nature* 451: 279-283.

Condamine, F.L. et al (2013) Macroevolutionary perspectives to environmental change *Eco Lett.* 16: 72-85.

traits.summary *Summarizing trait data*

Description

Summarizes trait data from a sim object, usually the ouput of bd.sim in the case where diversification rates are trait-dependent. Returns a list of trait values at the present or the time of extinction (depending on whether the species is alive at present), and optionally returns values at the time of fossil sampling if provided with a fossil record object fossils, usually the output of sample.clade. Does not make assumptions on the number of traits described in the traits parameter, so that if that list has more than one trait per species, multiple vectors will be returned by the function.

Usage

```
traits.summary(sim, traits, fossils = NULL, selection = "all")
```

Arguments

sim	A sim object, containing extinction times, speciation times, parent, and status information for each species in the simulation. See ?sim.
traits	List of trait data frames, usually one of the returns of bd.sim. traits[[i]][[j]] should correspond to the jth trait data frame for species i. The data frames contain the following columns value A vector of trait values the species took at specific intervals of time. max A vector of time values corresponding to the upper bound of each interval. min A vector of time values corresponding to the lower bound of each interval
fossils	A data frame with a "Species" column and a SampT column, usually an output of the sample.clade function. Species names must contain only one number each, corresponding to the order of the sim vectors. Note that we require it to have a SampT column, i.e. fossils must have an exact age. This assumption might be relaxed in the future.
selection	Which subset of species to collect trait data for. If set to "all", it will return every trait value it has access to, i.e. either all species, living or dead, or all species plus fossils if fossils is supplied. If set to "extant", it will return only trait values for living species. If set to "extinct", it will return only trait values for extinct species, and fossils if fossils is supplied. If set to "fossil", it will return values for only the fossil species (and therefore requires a fossils parameter). If set to "sampled", it will function the same as in the case for "extant", except it will also return values for the fossils if fossils is supplied.

Value

A named list of named vectors of trait values. List element names refer to each trait, so i.e. res$traitN will correspond to the vector of trait values for trait N. Vector element names refer to the species, using the default naming convention of the package (tN is the Nth species in the simulation, and tN.M is the Mth sampled fossil of that species).

Author(s)

Bruno do Rosario Petrucci

Examples

```
###
# need a simple simulation to use as an example

# initial number of species
n0 <- 1

# maximum simulation time
tMax <- 40

# speciation, higher for state 1
lambda <- c(0.1, 0.2)

# extinction, trait-independent
mu <- 0.03

# number of traits and states (1 binary trait)
nTraits <- 1
nStates <- 2

# initial value of the trait
X0 <- 0

# transition matrix, with symmetrical transition rates
Q <- list(matrix(c(0, 0.1,
                   0.1, 0), ncol = 2, nrow = 2))

# set seed
set.seed(1)

# run the simulation
sim <- bd.sim.traits(n0, lambda, mu, tMax, nTraits = nTraits,
                     nStates = nStates, X0 = X0, Q = Q, nFinal = c(2, Inf))

# get all trait values
traitSummary <- traits.summary(sim$SIM, sim$TRAITS)
traitSummary

# could get only the extant values, instead
traitSummary <- traits.summary(sim$SIM, sim$TRAITS, selection = "extant")
traitSummary

# or all the extinct values
traitSummary <- traits.summary(sim$SIM, sim$TRAITS, selection = "extinct")
traitSummary

# set seed
set.seed(1)
```

```
# maybe we want to take a look at the traits of fossil records too
fossils <- sample.clade(sim$SIM, rho = 0.5, tMax = max(sim$SIM$TS))

# get the trait values for all extinct species, including fossil samples
traitSummary <- traits.summary(sim$SIM, sim$TRAITS,
                               fossils = fossils, selection = "extinct")
traitSummary

# can also get the values for all sampled species, i.e. extant or fossils
traitSummary <- traits.summary(sim$SIM, sim$TRAITS,
                               fossils = fossils, selection = "sampled")
traitSummary

# or just the fossil species
traitSummary <- traits.summary(sim$SIM, sim$TRAITS,
                               fossils = fossils, selection = "fossil")
traitSummary
```

`var.rate.div` *Expected diversity for general exponential rates*

Description

Calculates the expected species diversity on an interval given a (possibly time-dependent) exponential rate. Takes as the base rate (1) a constant, (2) a function of time, (3) a function of time interacting with an environmental variable, or (4) a vector of numbers describing rates as a step function. Requires information regarding the maximum simulation time, and allows for optional extra parameters to tweak the baseline rate. For more information on the creation of the final rate, see `make.rate`.

Usage

```
var.rate.div(rate, t, n0 = 1, tMax = NULL, envRate = NULL, rateShifts = NULL)
```

Arguments

`rate` The baseline function with which to make the rate. It can be a

A number For constant birth-death rates.

A function of time For rates that vary with time. Note that this can be any function of time.

A function of time and an environmental variable For rates varying with time and an environmental variable, such as temperature. Note that supplying a function on more than one variable without an accompanying `envRate` will result in an error.

A numeric vector To create step function rates. Note this must be accompanied by a corresponding vector of rate shift times, `rateShifts`.

t	A time vector over which to consider the distribution.
n0	The initial number of species is by default 1, but one can change to any nonnegative number. Note: var.rate.div will find the expected number of species given a rate rate and an initial number of parents n0, so in a biological context rate is diversification rate, not speciation (unless extinction is 0).
tMax	Ending time of simulation, in million years after the clade's origin. Needed to ensure rateShifts runs the correct way.
envRate	A data.frame representing a time-series, usually an environmental variable (e.g. CO2, temperature, etc) varying with time. The first column of this data.frame must be time, and the second column must be the values of the variable. The function will return an error if the user supplies envRate without rate being a function of two variables. paleobuddy has two environmental data frames, temp and co2. One can check RPANDA for more examples. Note that, since simulation functions are run in forward-time (i.e. with 0 being the origin time of the simulation), the time column of envRate is assumed to do so as well, so that the row corresponding to t = 0 is assumed to be the value of the time-series when the simulation starts, and t = tMax is assumed to be its value when the simulation ends (the present). Acknowledgements: The strategy to transform a function of t and envRate into a function of t only using envRate was adapted from RPANDA.
rateShifts	A vector indicating the time of rate shifts in a step function. The first element must be the first or last time point for the simulation, i.e. 0 or tMax. Since functions in paleobuddy run from 0 to tMax, if rateShifts runs from past to present (meaning rateShifts[2] < rateShifts[1]), we take tMax - rateShifts as the shifts vector. Note that supplying rateShifts when rate is not a numeric vector of the same length will result in an error.

Value

A vector of the expected number of species per time point supplied in t, which can then be used to plot vs. t.

Examples

```
# let us first create a vector of times to use in these examples
time <- seq(0, 50, 0.1)

###
# we can start simple: create a constant rate
rate <- 0.1

# make the rate
r <- make.rate(0.5)

# plot it
plot(time, rep(r, length(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')
```

```
# get expected diversity
div <- var.rate.div(rate, time)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')

###
# something a bit more complex: a linear rate
rate <- function(t) {
  return(1 - 0.05*t)
}

# make the rate
r <- make.rate(rate)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')
# negative values are ok since this represents a diversification rate

# get expected diversity
div <- var.rate.div(rate, time)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')

###
# remember: rate is the diversification rate!

# we can create speciation...
lambda <- function(t) {
  return(0.5 - 0.01*t)
}

# ...and extinction...
mu <- function(t) {
  return(0.01*t)
}

# ...and get rate as diversification
rate <- function(t) {
  return(lambda(t) - mu(t))
}

# make the rate
r <- make.rate(rate)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')
```

```
# get expected diversity
div <- var.rate.div(rate, time)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(50, 0), type = 'l')

###
# we can use ifelse() to make a step function like this
rate <- function(t) {
  return(ifelse(t < 2, 0.2,
          ifelse(t < 3, 0.4,
           ifelse(t < 5, -0.2, 0.5))))
}

# change time so things are faster
time <- seq(0, 10, 0.1)

# make the rate
r <- make.rate(rate)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')
# negative rates is ok since this represents a diversification rate

# get expected diversity
div <- var.rate.div(rate, time)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

# this method of creating a step function might be annoying, but when
# running thousands of simulations it will provide a much faster
# integration than when using our method of transforming
# a rates and a shifts vector into a function of time

###
# ...which we can do as follows

# rates vector
rateList <- c(0.2, 0.4, -0.2, 0.5)

# rate shifts vector
rateShifts <- c(0, 2, 3, 5)

# make the rate
r <- make.rate(rateList, tMax = 10, rateShifts = rateShifts)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
```

```
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')
# negative rates is ok since this represents a diversification rate

# get expected diversity
div <- var.rate.div(rateList, time, tMax = 10, rateShifts = rateShifts)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

###
# finally let us see what we can do with environmental variables

# get the temperature data
data(temp)

# diversification
rate <- function(t, env) {
  return(0.2 + 2*exp(-0.25*env))
}

# make the rate
r <- make.rate(rate, tMax = tMax, envRate = temp)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

# get expected diversity
div <- var.rate.div(rate, time, tMax = tMax, envRate = temp)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

###
# we can also have a function that depends on both time AND temperature

# diversification
rate <- function(t, env) {
  return(0.02 * env - 0.01 * t)
}

# make the rate
r <- make.rate(rate, tMax = tMax, envRate = temp)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

# get expected diversity
div <- var.rate.div(rate, time, tMax = tMax, envRate = temp)
```

```
# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')

###
# as mentioned above, we could also use ifelse() to construct a step
# function that is modulated by temperature

# diversification
rate <- function(t, env) {
  return(ifelse(t < 2, 0.1 + 0.01*env,
          ifelse(t < 5, 0.2 - 0.05*env,
           ifelse(t < 8, 0.1 + 0.1*env, 0.2))))
}

# make the rate
r <- make.rate(rate, tMax = tMax, envRate = temp)

# plot it
plot(time, rev(r(time)), ylab = "Diversification rate",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')


# get expected diversity
div <- var.rate.div(rate, time, tMax = tMax, envRate = temp)

# plot it
plot(time, rev(div), ylab = "Expected number of species",
     xlab = "Time (Mya)", xlim = c(10, 0), type = 'l')


# takes a bit long so we set it to not run, but the user
# should feel free to explore this and other scenarios
```

Index