This is a repository copy of *treats: a modular R package for simulating trees and traits*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/210154/

Version: Published Version

Article:
Guillerme, T. orcid.org/0000-0003-4325-1275 (2024) treats: a modular R package for simulating trees and traits. Methods in Ecology and Evolution. ISSN 2041-210X

https://doi.org/10.1111/2041-210x.14306

Reuse
This article is distributed under the terms of the Creative Commons Attribution-NonCommercial (CC BY-NC) licence. This licence allows you to remix, tweak, and build upon this work non-commercially, and any new works must also acknowledge the authors and be non-commercial. You don't have to license any derivative works on the same terms. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Received: 9 June 2023 | Accepted: 1 February 2024

DOI: 10.1111/2041-210X.14306

Methods in Ecology and Evolution

APPLICATION

`treats`: A modular R package for simulating trees and traits

Thomas Guillerme

School of Biosciences, University of Sheffield, Sheffield, UK

Correspondence
Thomas Guillerme
Email: guillert@tcd.ie

Funding information
UKRI-NERC Grant, Grant/Award Number: NE/T000139/1

Handling Editor: Tiago Quental

Abstract

1. Simulating biological realistic data is an important step to understand and investigate biodiversity. Simulated data can be used to generate null, base line or neutral models. These can be used either in comparison to observed data to estimate the mechanisms that generated the data. Or they can be used to explore, understand and develop theoretical advances by proposing toy models.
2. In evolutionary biology, simulations often involve the need of an evolutionary process where descent with modification is at the core of how the simulated data are generated. These evolutionary processes can then be nearly infinitely modified to include complex processes that affect the simulations such as traits co-evolution, competition mechanisms or mass extinction events.
3. Here I present the `treats` package, a modular `R` package for trees and traits simulations. This package is based on a simple birth death algorithm from which all steps can easily be modified by users.
4. `treats` also provides a tidy interface through the `treats` object, allowing users to easily run reproducible simulations. It also comes with an extend manual regularly updated following users' questions or suggestions.

KEYWORDS
birth-death, disparity, ecology, evolution, null-models, simulations, traits, trees

1 | INTRODUCTION

Comparing biological patterns is one of the key ways to understand mechanisms in evolutionary biology. This leads to the development of phylogenetic comparative methods as key methodologically driven topic in ecology, evolution and palaeontology (Felsenstein, 1985; Pennell & Harmon, 2013). These methods rely on comparing patterns in a phylogenetic context to understand biological mechanisms or concepts (Harmon, 2019). These comparisons can be done between observed patterns under different conditions or against null, neutral or baseline models (see Bausman, 2018 for distinctions) suggesting different processes or mechanisms. For example different traits distribution for species with different diets (Deepak et al., 2023) or habitats (Pinto-Ledezma et al., 2017). Or by comparing some observed pattern to one simulated under null or base conditions (Miller et al., 2022). In theory, workers can use the following research pipeline: (1) thinking of a specific mechanism (e.g. mass extinction allowing the surviving species to acquire new morphologies), (2) collecting some data to test this mechanism (e.g. some traits of species across and extinction event) and then (3) comparing these patterns to some simulated under no specific conditions (e.g. a null model where the traits evolve randomly regardless of an extinction event, Puttick et al., 2020). Workers might thus need to simulate a great diversity of evolutionary scenarios to test their specific

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.
© 2024 The Authors. *Methods in Ecology and Evolution* published by John Wiley & Sons Ltd on behalf of British Ecological Society.

question. To do so, we need statistical and software solutions to simulate trees and data to generate many specific null models.

In practice, these evolutionary simulations can be done relatively easily on computers using a birth-death process (Feller, 1939; FitzJohn, 2012; Stadler, 2010). A birth-death process is a continuous time Markov process that had been routinely implemented in `R` (R Core Team, 2023) to simulate realistic phylogenies (e.g. FitzJohn, 2012; Paradis & Schliep, 2019). This general algorithm to generate phylogenetic trees can be coupled with other Markov processes to also generate traits, for example using a Brownian Motion process (BM; Cavalli-Sforza & Edwards, 1967) or an Ornstein Uhlenbeck (OU; Lande, 1976; see Cooper, Thomas, Venditti, et al., 2016 for a distinction between both). In `R`, this can be done with several already well used and well documented packages. For example if you want to simulate diversity through time, you can use `TreeSim` (Stadler, 2011) to simulate diversity under a set of specific parameters (e.g. speciation and extinction) with some events disrupting the simulations (e.g. mass extinctions). You can even improve on generating these patterns using `FossilSim` (Barido-Sottani et al., 2019) to take into account fossilisation processes. You can also use `paleobuddy` (do Rosario Petrucci et al., 2022) or `paleotree` (Bapst, 2012) to generate palaeontology specific data. On the other hand, if you need to simulate both diversity and traits through time, this can be done with specific parameters in `RPANDA` (Morlon et al., 2016), `diversitree` (FitzJohn, 2012) or `PETER` (Puttick et al., 2020) where the traits are generated stochasticaly through time (given some process) during the birth-death process.

Although the packages mentioned above are excellent and routinely used with fast and reliable algorithms and associated documentation, they are all designed for specific tasks and don't allow much modification beyond the input parameters designed by the authors. For example, `TreeSim` can simulate a birth-death tree with some extinction event but is not designed to simulated one with an extinction event that leads to the birth-death process to be not diversity dependent anymore, simulating a release in selection pressure after the extinction event that leads to a different process dominating speciation. Or `PETER` is not designed to simulate a complex set of traits (say three correlated BM traits and two independent OU ones). This absence of modularity has hampered the use of complex and question-driven simulations, although I acknowledge this was not the primary aim of the authors of the excellent packages mentioned above. This has led workers to often develop their own tools to answer specific questions (e.g. Puttick et al., 2020). Therefore, I propose `treats` a modular `R` package to simulate both trees and traits through time. Note that although `treats` is modular and thus allows to be used as go to tool for simulating and trees and traits, it lacks the ready-to-use implemented methods featured in other packages such as fossilisation and sampling (Barido-Sottani et al., 2019; do Rosario Petrucci et al., 2022; Stadler, 2011) or specific macroevolutionary simulations (Morlon et al., 2016; Puttick et al., 2020).

2 | DESCRIPTION

treats is based on the eponymous treats function that allows to simulate a phylogeny and some trait(s) simultaneously. The base birth-death algorithm "grows" a phylogenetic tree and generates traits for each node and tips in the following manner:

1. Generating branch length;
2. Selecting a lineage among the currently living ones;
3. Choosing whether that lineage goes extinct (becomes a tip) or speciates (becomes a node).

These three steps are repeated until the tree reaches the desired age or number of species (the algorithm's implementation is heavily inspired and based on FitzJohn, 2012). If traits are simulated during the process, a fourth step is added:

4. Generating some trait(s) value(s) for the selected lineage (either a tip or a node—but see the "Simulating traits section" below to generate trait values along edges).

In `treats`, these three or four steps are implemented as modular functions that the user can easily change using an internal class of objects called `"modifiers"` or `"traits"` (Figure 1). The simulation then outputs a tree (of class `"phylo"` and a associated table of traits—`"matrix"`) that can be visualised using the `plot.treats` function. A third class of object called `"events"` can be added to the simulations to modify the entire simulation under certain conditions (e.g. simulating a mass extinction). Each element in the algorithm can be modified by the user using the implemented functions `make.bd.params` to set the birth-death parameters, `make.traits` to set the trait(s), `make.modifiers` to set the birth-death algorithm and `make.events` to eventually add one or more events. Users can replace any step in the algorithm by their own specific functions suiting their needs or, for less advanced users, by using already implemented functions.

2.1 | Major functionalities

The following sections provides an overview of the three main functions displayed in Figure 1 (`make.traits, make.modifiers and make.events`).

2.1.1 | Simulating traits

Traits are simulated via the `make.traits` function given one or more processes, the number of dimensions per process and some starting value(s). Essentially, the generation of new trait values is based on a process (a `function`) modifying a trait value (`x0`) relative to some branch length (`edge.length`). For example, a simple BM process could be generated by the function `rnorm` where it modifies

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

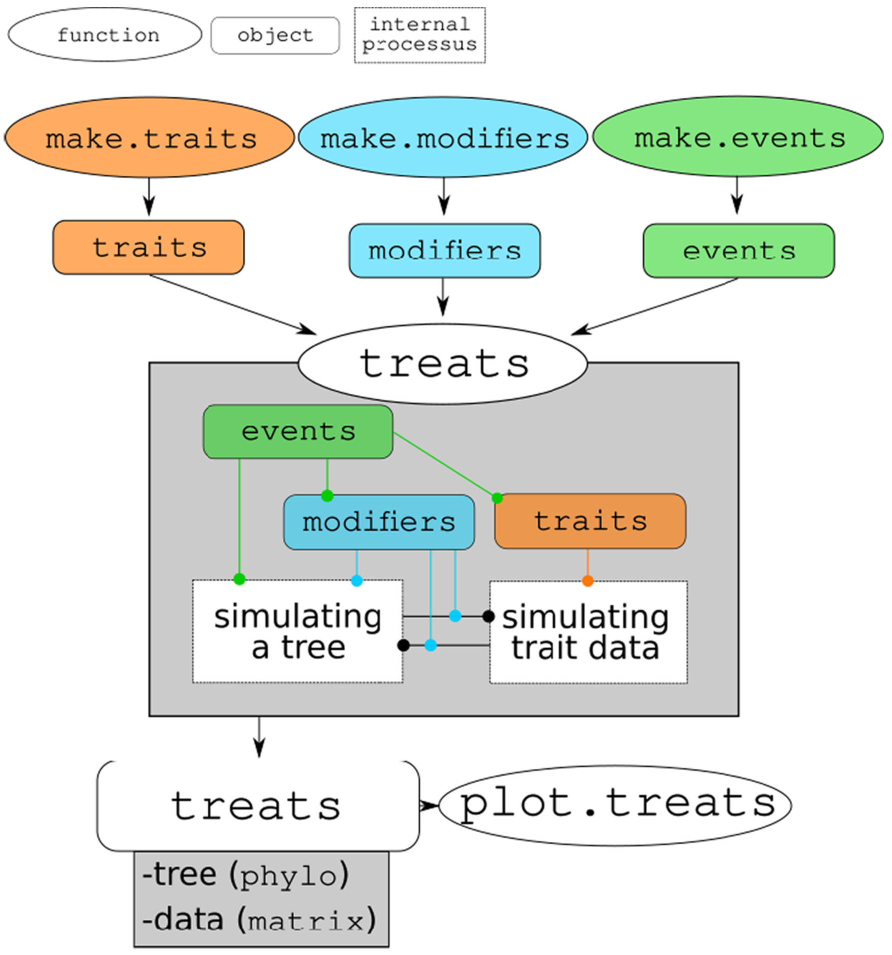


FIGURE 1 treats package workflow: the treats algorithm generates a tree and traits using inbuilt "traits" objects that contain the instructions on how to generate the trait data (e.g. which process? how many dimensions?); "modifiers" objects that contains instructions on how to "grow" the tree (e.g. by linking speciation to trait values or to the current number of species); and "events" objects that can modify the tree structure, "modifiers" or "traits" depending on specific conditions (e.g. 80% of species with positive trait values go extinct after reaching a specific time).

the trait value `x0` relative to the branch length (`edge.length`). The longer the branch length, the more likely the new trait value will be different from `x0`:

```
## Brownian Motion process
my.bm.process <- function(x0, edge.length) {
    rnorm(n = 1, mean = x0, sd = edge.length)
}
## Creating the traits object
my.trait <- make.traits(process = my.bm.process)
```

Of course biological data can be much more complex and often multivariate (Adams & Collyer, 2019) requiring the users to develop more complex function to cater their specific needs. The `treats` package contains a list of pre-built processes that are "ready-to-use":

- `BM.process` and `OU.process`: generalised BM and Ornstein-Uhlenbeck processes in any number of dimensions (including possible correlation);
- `multi.peak.process`: a generalised Ornstein-Uhlenbeck process (uni- or multi-dimensional) which allows for multiple peaks;
- `repulsion.process`: a unidimensional process generating trait values that don't overlap with previously generated trait values;
- `discrete.process`: a process generating discrete trait values;
- `no.process`: ignores the branch length (i.e. a time independent process).

Traits are always generated (and stored) only for tips or nodes and not along edges. However, it is possible so generate trait values at specific or regular time intervals by using the option `save.steps` in the `treats` function. This will generate singleton nodes (i.e. nodes with only one descendant) with associated trait values.

Note that the `treats` package primary aim is to generate both a tree and some traits at the same time. However, it is possible to also just generate traits with a given topology. This is done through the function `map.traits` that intakes one or more trees and a "`traits`" object.

2.1.2 | Modifying the birth-death process

Modifying the birth-death process can be done in several ways. Most easily it is done through changing the stopping rules through the `stop.rule` argument (number of total taxa or living ones, or time of the simulation). Equally straightforward, one can modify the parameters of the birth-death process through the `make.bd.params` function: the speciation (λ) and the extinction (μ) ones. These can be

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

either fixed values (for constant speciation and extinction) or values drawn from distributions.

It is also possible to directly modify how the birth-death algorithm works through `make.modifiers` by changing the three main components of the birth-death algorithm as described above. By default, the algorithm uses the following algorithms:

1. **Generating branch length** by drawing a value from an exponential distribution with the rate being function of the current number of lineages scaled by the speciation and extinction parameters

```
## The default branch length generation
rexp(1, rate = number_of_lineages * (speciation + extinction))
```

2. **Selecting a lineage** among the currently living ones by simply sampling across the available (living) lineages:

```
## The default lineage selection
sample(number_of_lineages, 1)
```

3. **Choosing whether that lineage goes extinct (becomes a tip) or speciates (becomes a node)** by drawing a random number between 0 and 1 and comparing it to the ratio of speciation and turnover (speciation + extinction). If the random number is smaller than the ratio of speciation and turnover, the lineage speciates, else it goes extinct:

```
## The default speciation/extinction decider
runif(1) <= (speciation / (speciation + extinction))
```

The function `make.modifiers` allows to specifically change any of these components by providing a different function for each part of the algorithm. For example, one can modify the three functions above so that the branch length is not dependent on the number of lineages, the sampling is always the first species available and the extinction is drawn from a normal distribution. Note that these function need a specific syntax that is detailed in the treats manual.

```
## Lineage independent waiting:
lineage.independent <- function(bd.params,
                                lineage = NULL,
                                trait.values = NULL,
                                modify.fun = NULL) {
   my_rate <- bd.params$speciation + bd.params$extinction
   return(rexp(1, rate = my_rate))
}
## Selecting always the first species
select.first <- function(bd.params,
                         lineage = NULL,
                         trait.values = NULL,
                         modify.fun = NULL) {
    return(as.integer(1))
}
## Random normal speciation
normal.speciation <- function(bd.params,
                              lineage = NULL,
                              trait.values = NULL,
                              modify.fun = NULL) {
   my_turnover <- bd.params$speciation/
                  (bd.params$speciation + bd.params$extinction)
   return(rnorm(1) <= my_turnover)
}
## Creating the modifier object
modified.birth.death <- make.modifiers(branch.length = lineage.
independent,
                                       selection = select.first,
                                       speciation = normal.speciation)
```

2.1.3 | Creating events

The final major argument to be passed to `treats` are the "`events`" objects generated through make. events where the following information needs to be specified:

- The `target` designating what the event should be applied to (e.g. "`taxa`" for modifying the number of species, "`traits`" for modifying the traits, etc.).
- The `condition` which is a function returning a logical value of when to trigger the event (e.g. when reaching a certain number of taxa, after some specific time has ellapsed or if some specific trait value is reached, etc.).
- The `modification` which is a function that specifically modifies an internal object in the treats algorithm.

For a more exhaustive list of events so you can refer to the `treats` manual with many different detailed examples. Briefly though, this is how the mass extinction events are designed in the example above.

```
## Creating an extinction that removes species with positive trait values
positive_extinction <- make.events(
     target = "taxa",
     condition = age.condition(15),
     modification = trait.extinction(x = 0, condition = `>=`))
```

For this event, the target is the number of taxa in the simulations. This is indicated using the `target = "taxa"` argument. Then the event is triggered using the argument `age.condition(15)` and modifies the internal `lineage` object using the `trait.extinction(x=0, condition=`>=`)` function. `age.condition` and trait.extinction are both function factories that are not going to be detailed here (see Wickham, 2019). Effectively these arguments can be passed directly as standard functions. For example, to trigger the event when reaching time 15, we can use the following function:

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
## Returns TRUE when reaching time 15
reaching.time15 <- function(bd.params, lineage, trait.values, time) {
    return(time > 15)
}
```

This will trigger the modification which is equivalent to the following function modifying the lineage internal list:

```
removing.75.taxa <- function(bd.params, lineage, trait.values) {
    ## Select a portion of the living species to go extinct
    extinct <- sample(lineage$n, round(lineage$n * 0.75))
    ## Update the lineage object
    lineage$livings <- lineage$livings[-extinct]
    lineage$n <- lineage$n - length(extinct)
    return(lineage)
}
```

Hence, the extinction event described above is equivalent to the following one:

```
## Creating an extinction that removes species with positive trait values
positive_extinction <- make.events(
    target = "taxa",
    condition = reaching.time15,
    modification = removing.75.taxa)
```

2.1.4 | Visualising results

The `treats` package also comes with tools to visualise trees and traits together or separately. This can be done through the generic S3 plot function (calling `plot.treats`) and allows to display up to three traits or two traits and time (in 3D) as displayed in Figures 4 and 5. These functions can also be used to visualise trees and traits together from non "`treats`" objects by using the `make.treats` function to transform them into "`treats`" objects.

2.2 | Brief applied example

To illustrate this we will look whether it is possible to detect changes in disparity (i.e. diversity of traits) in a subset of the data published from Beck and Lee (2014) implemented in Guillerme (2018). This dataset contains the ordinated traits for 50 mammalian species across the Cretaceous-Palaeogene extinction event (K-Pg, 66 Mya; Figure 2).

```
## Loading the package and data
library(treats)
data(BeckLee_tree)
data(BeckLee_mat99)

## Creating the time slices
time_slices <- chrono.subsets(BeckLee_mat99, BeckLee_tree,
                              method = "continuous",
                              model = "proximity",
                              time = seq(from = 96, to = 36, by = -10))

## Calculating disparity on the two first dimensions only
observed_disparity <- dispRity(boot.matrix(time_slices),
                               metric = c(sum, variances),
                               dimensions = c(1,2))

## Plotting the tree and the disparity through time
par(mfrow = c(1,2), bty = "null")
```

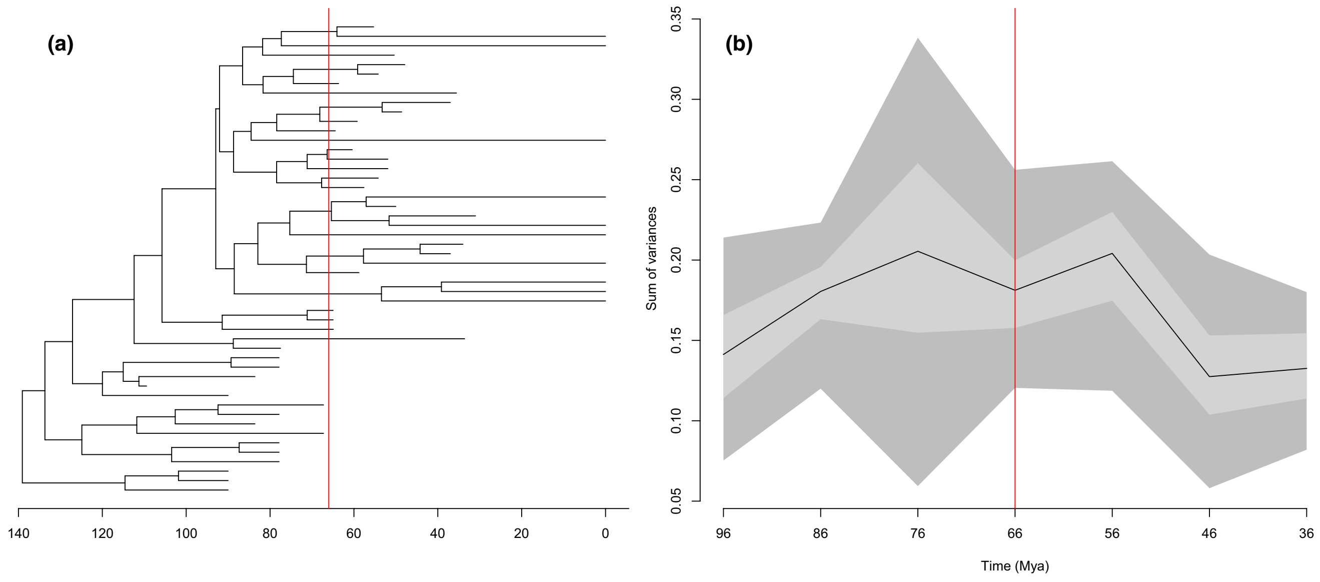


FIGURE 2 Observed phylogeny (a) and disparity through time (b) in the subset of Beck and Lee's (2014) dataset. The red line represents the K-Pg boundary. Is this change in disparity related to the K-Pg mass extinction? Or, more generally, is it even possible to detect potential changes of disparity due to a mass extinction event?

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
## Thre tree
plot(ladderize(BeckLee_tree), show.tip.label = FALSE)
axisPhylo()
abline(v = BeckLee_tree$root.time - 66, col = "red")
legend("topleft", pch = NULL, legend = "A", bty = "n", cex = 2)

## The disparity
plot(observed_disparity, ylab = "Sum of variances")
abline(v = 4, col = "red")
legend("topleft", pch = NULL, legend = "B", bty = "n", cex = 2)
```

Using this example dataset, one might be interested in testing whether the K-Pg extinction had an effect on disparity through time. But can such effect be detected in the first place? We can test this by simulating some datasets with similar properties as the observed data and measure changes in disparity in these simulated datasets.

2.2.1 | Simulating trees

The first and simplest way is to simulate tree topologies that have similar properties than the observed one. To do so, we need to use some **speciation** parameter indicating the rate at which lineages speciate (*aka* "birth" or "λ") and an **extinction** parameter indicating the rate at which they go extinct (*aka* "death" or "μ"). Here, we are using two relatively arbitrary (speciation = 0.035 and extinction = 0.02) to get trees roughly matching the observed tree. Note that you might want to consider more appropriate ways to calculate these rates for research projects (e.g. Magallon & Sanderson, 2001). We also need a stopping rule for when to stop the simulations (in our case when reaching 140 time units). This will produce a single random tree using the input parameters (Figure 3). The number of time units in `treats` is arbitrary and is not equivalent to millions of years. Using 140 time units here allows to simulate number of tips in a similar order of magnitude as the ones in the observed data. Note that I will not discuss the options here in great details. Much more information can be found in the `treats` manual.

```
## Using the birth-death parameters from the observed tree
my_bd_params <- make.bd.params(speciation = 0.035, extinction =
0.02)
## Setting the stopping rule (stop after 140 time units)
stop_rule <- list(max.time = 140)

## Simulate the tree
set.seed(2)
sim_tree <- treats(bd.params = my_bd_params,
                   stop.rule = stop_rule,
                   null.error = 100,
                   verbose = FALSE)
```

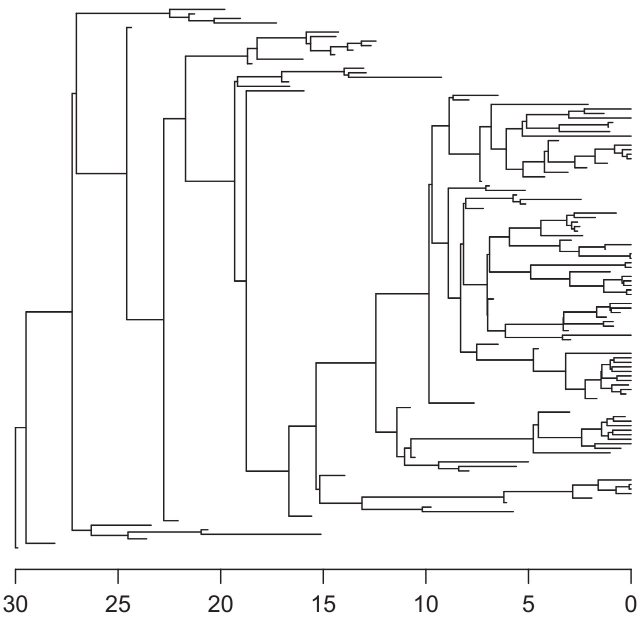


FIGURE 3 A randomly simulated tree with similar properties as the observed one. Note that the time here is expressed in arbitrary units.

2.2.2 | Simulating trees and traits

For our specific question, we will also need to simulate some traits associated with each node and tip. For simplicity we will simulate a two dimensional BM trait. To do so, we can create a "`traits`" object with the function `make.traits`. This results in a two-dimensional trait space for all the simulated species and their nodes (Figure 4).

```
## Creating a trait in 2 dimensions.
my_traits <- make.traits(process = BM.process, n = 2)

## Simulate the tree and traits
set.seed(123)
sim_data <- treats(traits = my_traits,
                   bd.params = my_bd_params,
                   stop.rule = stop_rule,
                   null.error = 100,
                   verbose = FALSE)

par(mfrow = c(1,2))
## Plotting one trait through time
plot(sim_data, ylab = "Trait 1", las = 1, main = "Trait 1 through time")

## Plotting the two dimensions against each other
plot(sim_data, trait = c(2,1), ylab = "Trait 1", xlab = "Trait 2", las = 1,
main = "Traits 1 and 2")
```

2.2.3 | Simulating a trees and traits with events

To answer our question, we also want to simulate an extinction event. To do so, we can create two different "`events`" object with the function

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

`make.events`. The first one will simulate a random extinction after reaching 66 time units and then making three quarter (0.75) of the taxa go extinct. The second one will simulate a random extinction but based on trait values: after reaching time 66, all the species with positive trait values will go extinct. Both scenarios illustrate two different types of mass extinctions but they are not equivalent: because of the ancestral trait value starting at 0, we expect the second scenario to remove on average only 50% of the species (i.e. half the species are expected to evolve a trait value above 0). For more details on simulating the effect of mass extinction and the difficulties to simulate an unambiguous effect of a mass extinction, see Puttick et al. (2020). Because of this stochasticity of the simulations, we will repeat them 50 times (using `replicates=50`) to generate a distribution of possible simulated scenarios as opposed to a random single one that could be idiosyncratic (Figure 5).

```
## Creating a random mass extinction
random_extinction <- make.events(
    target = "taxa",
    condition = age.condition(140-66),
    modification = random.extinction(0.75))
## Creating an extinction that removes species with positive trait values
positive_extinction <- make.events(
    target = "taxa",
    condition = age.condition(140-66),
    modification = trait.extinction(x = 0, condition = `>=`))

set.seed(123)
## Simulate the tree and traits with a random extinction event
sim_rand_extinction <- treats(
                  traits = my_traits,
                  bd.params = my_bd_params,
                  stop.rule = stop_rule,
                  events = random_extinction,
                  null.error = 100,
                  replicates = 50)
## Simulate the tree and traits with a selective extinction event
sim_trait_extinction <- treats(
                  traits = my_traits,
                  bd.params = my_bd_params,
                  stop.rule = stop_rule,
                  events = positive_extinction,
                  null.error = 100,
                  replicates = 50)
```

Once we have simulated a distribution of trees and traits with the two extinction scenarios, we can measure disparity as in Figure 2 for all the simulated data and compare it to the observed disparity.

```
## Remove single nodes simulated at the extinction
sim_rand_extinction <- drop.singles(sim_rand_extinction)
sim_trait_extinction <- drop.singles(sim_trait_extinction)

## Calculate the dispRity for all the simulations
random_extinction_disparity <- dispRitreats(sim_rand_extinction,
                              method = "continuous",
                              model = "proximity",
                              time = seq(from = 96, to = 36, by = -10),
                              metric = c(sum, variances),
                              scale.trees = FALSE)
selective_extinction_disparity <- dispRitreats(sim_trait_extinction,
                              method = "continuous",
                              model = "proximity",
```

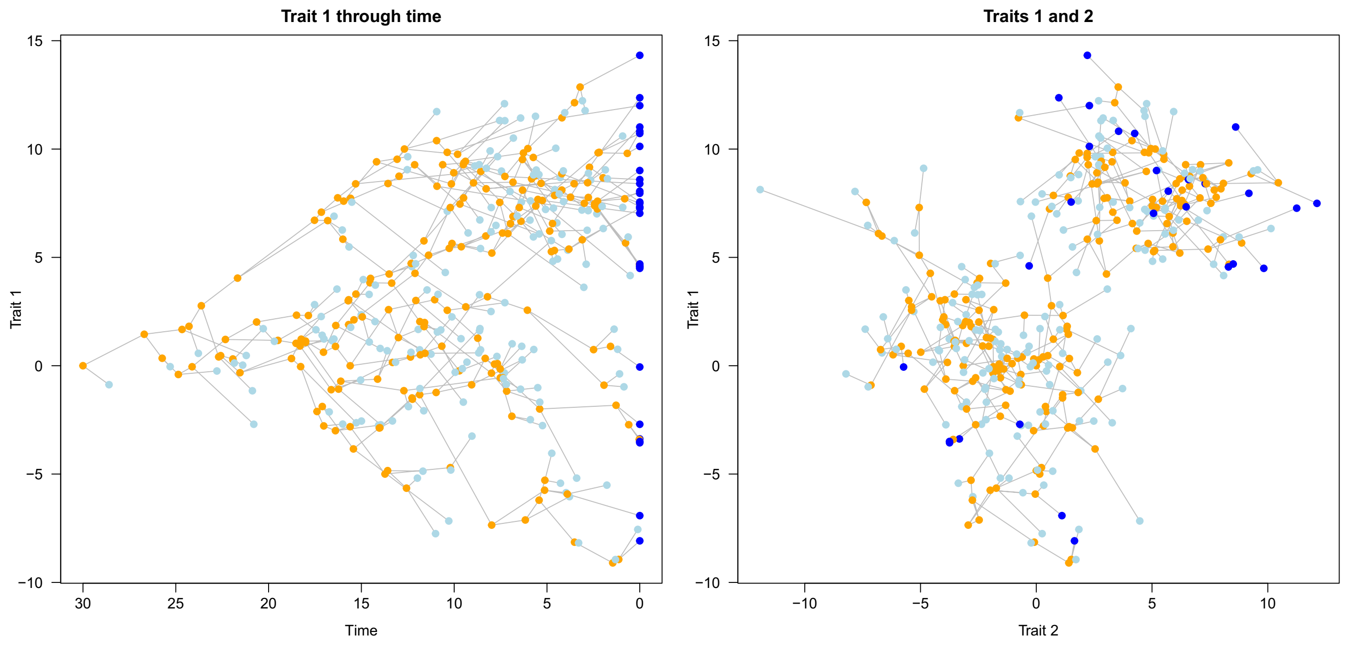
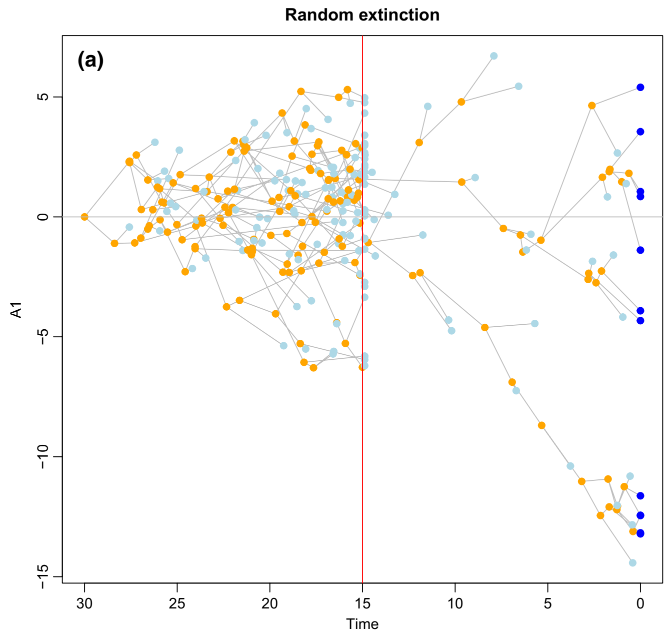


FIGURE 4 A randomly simulated tree with a two-dimensional random Brownian Motion trait. Orange, light blue and dark blue dots respectively represent nodes, fossils and living species.

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License



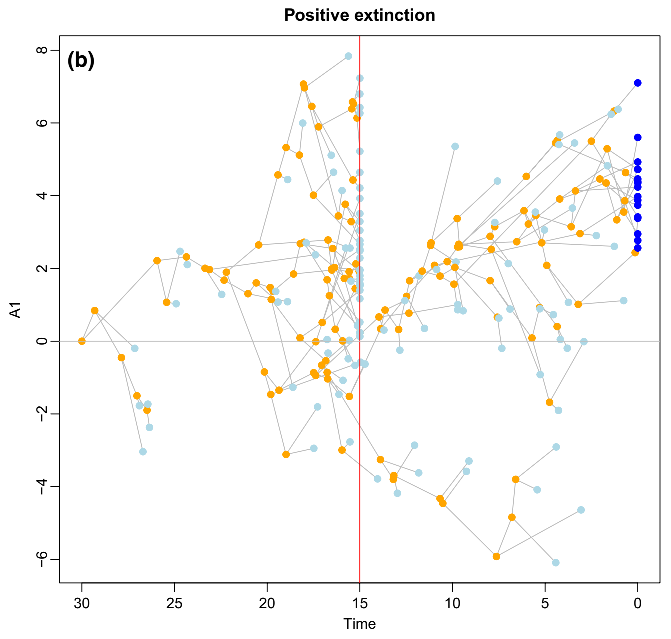
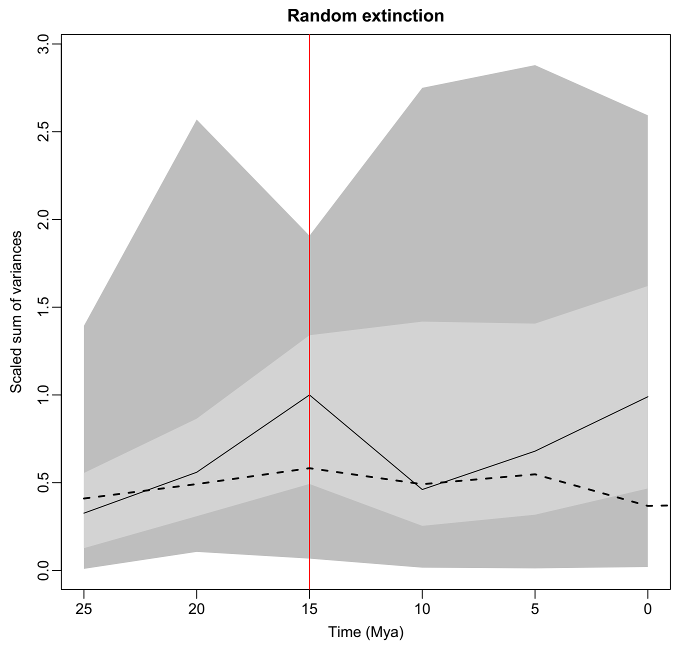


FIGURE 5 Two different mass extinction events: (a) 75% of species go extinct; (b) all species with positive trait values go extinct. The red line marks the extinction event.



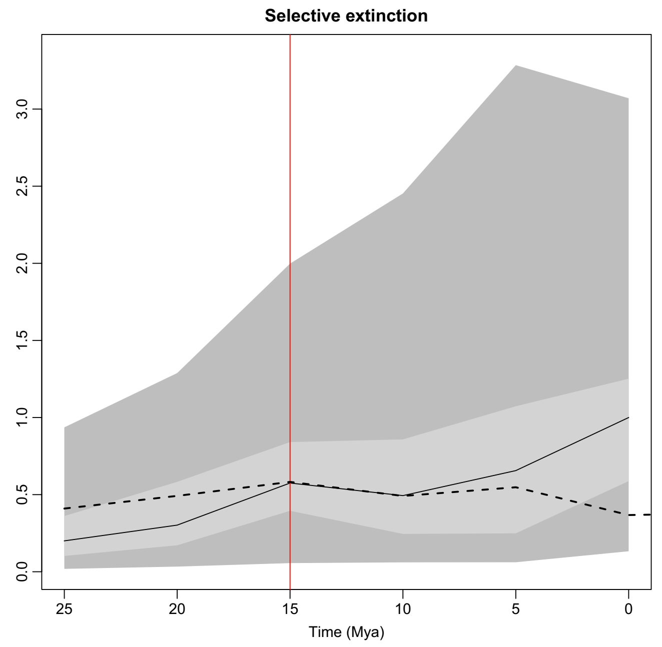


FIGURE 6 Comparison of the observed sum of variances (dashed line) to the simulated one (ligth and dark grey polygons and black line representing respectively the 95% and 50% confidence interval and the median value).

```
                    time   = seq(from = 96, to = 36, by = -10),
                    metric = c(sum, variances),
                    scale.trees = FALSE)

## Scale the disparity results (to compare to the observed ones)
random_extinction_disparity <- scale.dispRity(random_extinction_
disparity)
selective_extinction_disparity <- scale.dispRity(selective_extinc-
tion_disparity)
observed_disparity <- unlist(get.disparity(scale.dispRity(observed_
disparity)))

## Plotting the results with the observed disparity
par(mfrow = c(1,2))
plot(random_extinction_disparity, main = "Random extinction",
     ylab = "Scaled sum of variances", ylim = c(0, 1))
abline(v = 4, col = "red")
lines(x = 1:7, y = observed_disparity, lty = 2, lwd = 2)
```

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
plot(selective_extinction_disparity, main = "Selective extinction",
     ylab = "", ylim = c(0, 1))
abline(v = 4, col = "red")
lines(x = 1:7, y = observed_disparity, lty = 2, lwd = 2)
```

From these results we can draw some preliminary conclusions: the observed change in disparity is more likely due to a selective mass extinction rather than a random one (Figure 6). This is of course a very crude way of testing this, a more rigorous approach is needed to answer the question: more and better quality data, and more thorough methods (e.g. using a rank envelope test Murrell, 2018).

3 | ADDITIONAL INFORMATION

3.1 | Manuals, vignette and templates

The `treats` package comes with internal documentation (e.g. `?treats`) but also with a thorough and extended vignette in a gitbook format: the `treats` manual. This manual is designed so that it can be regularly updated and enhanced through the lifetime of the package facilitating the interface between methods development and usage (Cooper, Thomas, & FitzJohn, 2016). Furthermore, a library of simulation templates is maintained on the GitHub page. These templates are written and shared in the form of GitHub issues template and can be submitted and shared by any users. Either for them to have them stored somewhere curated, or better yet, so that other users can reuse, comment and mofidy them for their own projects.

3.2 | Further directions

This paper describes the first version of the `treats` package. However, I intend to continuously develop this package. For example future planned versions will include abiotic events and a better integration with the `dispRity` package. This will be done while keeping track change (through `NEWS` file), continuous integration and unit testing.

3.3 | Repeatability and reproducibility

This paper is entirely reproducible from an Rmarkdown document available on GitHub. The data used for the example above (Beck & Lee, 2014) is available from the `dispRity` package (Guillerme, 2018).

4 | CONCLUSION

The `treats` package modular architecture allows workers to develop their own specific biological simulation scenarios based on their own specific research question. The pipeline of the package through the different `"treats"` objects (`"traits"`, `"modifiers"` and `"events"`) also allows workers to generate publication standard results through plotting but also with easily reproducible and reusable scripts.

ACKNOWLEDGEMENTS

Many thanks to Mark Puttick for inspiring the development the package through previous collaborations. Thanks to Andrew Beckerman, Ian Brennan, Gustavo Burin, Christopher Cooney, Natalie Cooper, Alex Cranston, Jasmine Hardie, Tom Lansley, Clement Prieul, Joe Rees, James Rule, Sophie Ryan and Gavin Thomas, for support and useful comments on late stages of the development of this package. Thanks to William Gearty, one anonymous reviewer and one anonymous associate editor for their very useful suggestions. This work was funded by UKRI-NERC Grant NE/T000139/1 awarded to Gavin Thomas.

CONFLICT OF INTEREST STATEMENT

I declare no conflicts of interest.

PEER REVIEW

The peer review history for this article is available at https://www.webofscience.com/api/gateway/wos/peer-review/10.1111/2041-210X.14306.

DATA AVAILABILITY STATEMENT

The `treats` package is available on the CRAN (https://cran.r-project.org/web/packages/treats/) or on GitHub (https://github.com/TGuillerme/treats) with more associated information. All the versions of the package are archived on ZENODO with associated DOI: 10.5281/zenodo.10207680.

ORCID

Thomas Guillerme https://orcid.org/0000-0003-4325-1275

REFERENCES

Adams, D. C., & Collyer, M. L. (2019). Phylogenetic comparative methods and the evolution of multivariate phenotypes. *Annual Review of Ecology, Evolution, and Systematics*, *50*, 405–425.

Bapst, D. W. (2012). Paleotree: An R package for paleontological and phylogenetic analyses of evolution. *Methods in Ecology and Evolution*, *3*(5), 803–807.

Barido-Sottani, J., Pett, W., O'Reilly, J. E., & Warnock, R. C. M. (2019). FossilSim: An R package for simulating fossil occurrence data under mechanistic models of preservation and recovery. *Methods in Ecology and Evolution*, *10*(6), 835–840.

Bausman, W. C. (2018). Modeling: Neutral, null, and baseline. *Philosophy of Science*, *85*(4), 594–616. https://doi.org/10.1086/699021

Beck, R. M. D., & Lee, M. S. Y. (2014). Ancient dates or accelerated rates? Morphological clocks and the antiquity of placental mammals. *Proceedings of the Royal Society B: Biological Sciences*, *281*(1793), 20141278.

Cavalli-Sforza, L. L., & Edwards, A. W. F. (1967). Phylogenetic analysis. Models and estimation procedures. *American Journal of Human Genetics*, *19*(3 Pt 1), 233–257.

Cooper, N., Thomas, G. H., & FitzJohn, R. G. (2016). Shedding light on the 'dark Side' of phylogenetic comparative methods. *Methods in Ecology and Evolution*, *7*(6), 693–699.

Cooper, N., Thomas, G. H., Venditti, C., Meade, A., & Freckleton, R. P. (2016). A cautionary note on the use of Ornstein Uhlenbeck

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

models in macroevolutionary studies. *Biological Journal of the Linnean Society*, *118*(1), 64–77.

Deepak, V., Gower, D. J., & Cooper, N. (2023). Diet and habit explain head-shape convergences in natricine snakes. *Journal of Evolutionary Biology*, *36*(2), 399–411.

do Rosario Petrucci, B., Januario, M., & Quental, T. (2022). Paleobuddy: An R package for flexible simulations of diversification and fossil sampling. *Methods in Ecology and Evolution*, *13*(12), 2692–2698.

Feller, W. (1939). Die grundlagen der volterraschen theorie des kampfes ums dasein in wahrscheinlichkeitstheoretischer behandlung. *Acta Biotheoretica*, *5*(1), 11–40.

Felsenstein, J. (1985). Phylogenies and the comparative method. *The American Naturalist*, *125*(1), 1–15.

FitzJohn, R. G. (2012). Diversitree: Comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution*, *3*(6), 1084–1092.

Guillerme, T. (2018). dispRity: A modular R package for measuring disparity. *Methods in Ecology and Evolution*, *9*(7), 1755–1763.

Harmon, L. (2019). *Phylogenetic comparative methods: Learning from trees*. https://ecoevorxiv.org/repository/view/4486/

Lande, R. (1976). Natural selection and random genetic drift in phenotypic evolution. *Evolution*, *30*(2), 314–334.

Magallon, S., & Sanderson, M. J. (2001). Absolute diversification rates in angiosperm clades. *Evolution*, *55*(9), 1762–1780.

Miller, E. C., Martinez, C. M., Friedman, S. T., Wainwright, P. C., Price, S. A., & Tornabene, L. (2022). Alternating regimes of shallow and deep-sea diversification explain a species-richness paradox in marine fishes. *Proceedings of the National Academy of Sciences of the United States of America*, *119*(43), e2123544119.

Morlon, H., Lewitus, E., Condamine, F. L., Manceau, M., Clavel, J., & Drury, J. (2016). RPANDA: An R package for macroevolutionary analyses on phylogenetic trees. *Methods in Ecology and Evolution*, *7*(5), 589–597.

Murrell, D. J. (2018). A global envelope test to detect non-random bursts of trait evolution. *Methods in Ecology and Evolution*, *9*(7), 1739–1748.

Paradis, E., & Schliep, K. (2019). Ape 5.0: An environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, *35*, 526–528. https://doi.org/10.1093/bioinformatics/bty633

Pennell, M. W., & Harmon, L. J. (2013). An integrative view of phylogenetic comparative methods: Connections to population genetics, community ecology, and paleobiology. *Annals of the New York Academy of Sciences*, *1289*(1), 90–105.

Pinto-Ledezma, J. N., Simon, L. M., Diniz-Filho, J. A., & Villalobos, F. (2017). The geographical diversification of Furnariides: The role of Forest versus open habitats in driving species richness gradients. *Journal of Biogeography*, *44*(8), 1683–1693.

Puttick, M. N., Guillerme, T., & Wills, M. A. (2020). The complex effects of mass extinctions on morphological disparity. *Evolution*, *74*(10), 2207–2220.

R Core Team. (2023). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Stadler, T. (2010). Sampling-through-time in birth–death trees. *Journal of Theoretical Biology*, *267*(3), 396–404.

Stadler, T. (2011). Simulating trees with a fixed number of extant species. *Systematic Biology*, *60*(5), 676–684.

Wickham, H. (2019). *Advanced R*. CRC Press.

How to cite this article: Guillerme, T. (2024). `treats`: A modular `R` package for simulating trees and traits. *Methods in Ecology and Evolution*, *00*, 1–10. https://doi.org/10.1111/2041-210X.14306

2041210x, 0, Downloaded from https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14306 by University Of Sheffield, Wiley Online Library on [12/03/2024]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License