Package 'isotracer'

March 21, 2023

Type Package

Title Isotopic Tracer Analysis Using MCMC

Version 1.1.4

Description Implements Bayesian models to analyze data from tracer addition experiments. The implemented method was originally described in the article ``A New Method to Reconstruct Quantitative Food Webs and Nutrient Flows from Isotope Tracer Addition Experiments" by López-Sepulcre et al. (2020) <doi:10.1086/708546>.

License GPL-3

URL https://gitlab.com/matthieu-bruneaux/isotracer

BugReports https://gitlab.com/matthieu-bruneaux/isotracer/-/issues

Depends R (>= 3.6.0)

Imports coda (>= 0.19-3), data.table, dplyr (>= 0.8.5), latex2exp (>= 0.4.0), magrittr, methods (>= 3.6.0), pillar (>= 1.4.3), purrr (>= 0.3.3), Rcpp (>= 1.0.4), rlang (>= 0.4.5), rstan (>= 2.19.3), rstantools, tibble (>= 3.0.0), tidyr (>= 1.0.2), tidyselect (>= 1.0.0)

Suggests bayesplot, covr, cowplot, ggdist, ggplot2, ggraph, gridBase, gridExtra, here, igraph, knitr, lattice, readxl, rmarkdown, testthat, tidygraph, tidyverse, viridisLite

LinkingTo BH (>= 1.72.0), Rcpp (>= 1.0.4), RcppEigen (>= 0.3.3.7.0), StanHeaders (>= 2.19.2), rstan (>= 2.19.3), RcppParallel

SystemRequirements GNU make

NeedsCompilation yes

Encoding UTF-8

LazyData true

Biarch true

RoxygenNote 7.2.3

VignetteBuilder knitr

Author Andrés López-Sepulcre [aut] (<https://orcid.org/0000-0001-9708-0788>),
Matthieu Bruneaux [aut, cre] (<https://orcid.org/0000-0001-6997-192X>)

Maintainer Matthieu Bruneaux <matthieu.bruneaux@gmail.com>

Repository CRAN

Date/Publication 2023-03-21 14:10:02 UTC

R topics documented:

`isotracer-package` *The 'isotracer' package*

Description

The isotracer package allows modelling of fluxes across a network of compartments. Parameters are estimated using a Bayesian MCMC approach.

References

López-Sepulcre, A., M. Bruneaux, S. M. Collins, R. El-Sabaawi, A. S. Flecker, and S. A. Thomas. The American Naturalist (2020). "A New Method to Reconstruct Quantitative Food Webs and Nutrient Flows from Isotope Tracer Addition Experiments." https://doi.org/10.1086/708546.

Stan Development Team (2018). RStan: the R interface to Stan. R package version 2.18.2. https://mc-stan.org

---

`add_covariates` *Add fixed effects of one or several covariates to some parameters.*

Description

Note that new global parameters are not given any default prior.

Usage

```
add_covariates(nm, ..., use_regexpr = TRUE)
```

Arguments

`nm`	A `networkModel` object.
`...`	One or several formulas defining the covariates.
`use_regexpr`	Boolean, use regular expression to match the parameters affected by the formulas?

Value

A `networkModel` object.

Examples

```
# Using a subset of the topology from the Trinidad case study
m <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi, FBOM", "epi -> petro, pseph")

# Taking initial condtions from the 'lalaja' dataset at t=0
# Grouping by transect id
inits <- lalaja[lalaja[["time.days"]] == 0, ]
inits
m <- set_init(m, inits, comp = "compartment", size = "mgN.per.m2",
              prop = "prop15N", group_by = "transect")
m

# Default model
params(m, simplify = TRUE)

# Adding an effect of the "transect" covariate on some parameters
m <- add_covariates(m, upsilon_epi_to_pseph ~ transect)
params(m, simplify = TRUE)
```

---

add_pulse_event *Register a pulse event on one of the compartment of a topology*

Description

When applied to a steady-state compartment, this is equivalent to changing the steady state. Negative values are allowed, so one can add a "pulse" to a steady-state compartment and then later add a similar but negative "pulse" to simulate a drip in a stream for example.

Usage

```
add_pulse_event(nm, time, comp = NULL, unmarked, marked, which = NULL, pulses)
```

Arguments

nm	A networkModel object.
time	Numeric, time at which the pulse is happening.
comp	One compartment name only.
unmarked	Numeric, quantity of unmarked marker added.
marked	Numeric, quantity of marked marker added.
which	Vector of integers giving the nm rows to update. Default is to update all rows.
pulses	Optionally, a tibble containing the pulse information in columns. If provided, ‘comp‘, ‘time‘, ‘unmarked‘ and ‘marked‘ must be strings giving the corresponding column names.

Value

A networkModel object.

Examples

```
m <- trini_mod
m$events <- NULL
pulses <- tibble::tribble(
   ~ stream,    ~ transect, ~ comp, ~ time, ~ qty_14N, ~ qty_15N,
       "UL",  "transect.1",  "NH4",     11,         0,  -0.00569,
       "UL",  "transect.2",  "NH4",     11,         0,  -0.00264,
       "UL",  "transect.3",  "NH4",     11,         0, -0.000726,
       "UL",  "transect.1",  "NO3",     11,         0,  -0.00851,
       "UL",  "transect.2",  "NO3",     11,         0,  -0.01118,
       "UL",  "transect.3",  "NO3",     11,         0,  -0.01244,
   )
m <- add_pulse_event(m, pulses = pulses, comp = "comp", time = "time",
                     unmarked = "qty_14N", marked = "qty_15N")
m
```

---

aquarium_mod *A simple aquarium network model, ready to run*

Description

This network model is the model used in the Quick Start tutorial vignette. It is ready to be run at once with run_mcmc.

Usage

```
aquarium_mod
```

Format

An object of class networkModel (inherits from tbl_df, tbl, data.frame) with 1 rows and 4 columns.

Details

The code used to built the model is given in the example section below.

The aquarium_run dataset is a corresponding MCMC run.

Examples

```
library(tidyverse)
exp <- tibble::tribble(
  ~time.day,  ~species, ~biomass, ~prop15N,
          0,   "algae",     1.02,  0.00384,
          1,   "algae",       NA,   0.0534,
        1.5,   "algae",    0.951,       NA,
          2,   "algae",    0.889,   0.0849,
        2.5,   "algae",       NA,   0.0869,
          3,   "algae",    0.837,   0.0816,
          0, "daphnia",     1.74,  0.00464,
          1, "daphnia",       NA,  0.00493,
        1.5, "daphnia",     2.48,       NA,
          2, "daphnia",       NA,  0.00831,
        2.5, "daphnia",     2.25,       NA,
          3, "daphnia",     2.15,   0.0101,
          0,     "NH4",    0.208,     0.79,
          1,     "NH4",    0.227,       NA,
        1.5,     "NH4",       NA,    0.482,
          2,     "NH4",    0.256,    0.351,
        2.5,     "NH4",       NA,    0.295,
          3,     "NH4",     0.27,        NA
  )
inits <- exp %>% filter(time.day == 0)
obs <- exp %>% filter(time.day > 0)

aquarium_mod <- new_networkModel() %>%
    set_topo("NH4 -> algae -> daphnia -> NH4") %>%
    set_init(inits, comp = "species", size = "biomass",
             prop = "prop15N") %>%
    set_obs(obs, comp = "species", size = "biomass",
                 prop = "prop15N", time = "time.day")
```

`aquarium_run` *An MCMC run from a simple aquarium network model*

Description

This is an MCMC run on `aquarium_mod`. The code used to run the MCMC is: `aquarium_run <- run_mcmc(aquarium_mod, thin = 4)` (note that `thin = 4` was only used here to reduce the size of the data file shipped with the package, but for a real-life analysis keeping the default `thin = 1` is usually recommended). The code used to build the model itself is given in the help page for `aquarium_mod`.

Usage

```
aquarium_run
```

Format

An object of class `networkModelStanfit` (inherits from `mcmc.list`) of length 4.

Examples

```
## Not run:
plot(aquarium_run)
summary(aquarium_run)

## End(Not run)
```

`as.mcmc.list.tidy_flows` *Convert a* `tidy_flows` *object to an* `mcmc.list`

Description

Convert a `tidy_flows` object to an `mcmc.list`

Usage

```
## S3 method for class 'tidy_flows'
as.mcmc.list(x, ...)
```

Arguments

`x`	A tidy flow object, as returned by `tidy_flows`. Note that all chains must have the same iterations extracted (i.e. you must use `n_per_chain` when calling `tidy_flows`).
`...`	Not used for now.

Value

A `mcmc.list` object, with ordered iterations.

as.mcmc.list.tidy_steady_states *Convert a* tidy_steady_states *object to an* mcmc.list

Description

Convert a tidy_steady_states object to an mcmc.list

Usage

```
## S3 method for class 'tidy_steady_states'
as.mcmc.list(x, ...)
```

Arguments

x	A tidy steady states object, as returned by tidy_steady_states. Note that all chains must have the same iterations extracted (i.e. you must use n_per_chain when calling tidy_flows).
...	Not used for now.

Value

A mcmc.list object, with ordered iterations.

as_tbl_graph *Generic for as_tbl_graph()*

Description

Convert a compatible object to a tbl_graph object (from the tidygraph package)

Usage

```
as_tbl_graph(x, ...)
```

Arguments

x	Object to convert to a tbl_graph.
...	Passed to the appropriate method.

Value

A tbl_graph object.

as_tbl_graph.topology *Convert a network topology to a tbl_graph*

Description

Convert a network topology to a tbl_graph

Usage

```
## S3 method for class 'topology'
as_tbl_graph(x, ...)
```

Arguments

x	A network topology.
...	Not used.

Value

A tbl_graph object.

available_priors *List the available priors for model parameters*

Description

List the available priors for model parameters

Usage

```
available_priors()
```

Value

A tibble containing information about the available priors.

Examples

```
available_priors()
```

c.mcmc.list *Combine mcmc.list objects*

Description

Combine mcmc.list objects

Usage

```
## S3 method for class 'mcmc.list'
c(...)
```

Arguments

`...`	`mcmc.list` objects.

Value

A `mcmc.list` object.

calculate_steady_state *Calculate steady-state compartment sizes for a network*

Description

This is an experimental function. It attempts to calculate steady-state compartment sizes using the set parameter values and the initial compartment sizes. Use it with caution!

Usage

```
calculate_steady_state(nm)
```

Arguments

`nm`	A network model, with set parameter values.

Value

A tibble containing steady-state compartment sizes.

Examples

```
m <- aquarium_mod
m <- set_prior(m, constant_p(0), "lambda")
m <- set_params(m, sample_params(m))
proj <- project(m, end = 40)
plot(proj)

z <- calculate_steady_state(m)
z
z$stable_sizes
```

`comps` *Return the compartments of a network model*

Description

Return the compartments of a network model

Usage

```
comps(nm)
```

Arguments

`nm`	A `networkModel` object.

Value

A list of character vectors, with one list element per row of the input network model (list elements are in the same order as the input network model rows). Each list element containing the names of the compartments in the topology defined in the corresponding row of the input network model.

Examples

```
aquarium_mod
comps(aquarium_mod)

trini_mod
comps(trini_mod)
```

constant_p *Define a fixed-value prior*

Description

This is equivalent to having a fixed parameter.

Usage

```
constant_p(value)
```

Arguments

value	The constant value of the parameter.

Value

A list defining the prior.

Examples

```
constant_p(2)
```

delta2prop *Convert delta notation to proportion of heavy isotope*

Description

For details and references about quantities used in expressing isotopic ratios, see:

Usage

```
delta2prop(x = NULL, Rstandard = NULL)
```

Arguments

x	Vector of delta values
Rstandard	String describing the isotopic measurement, e.g. "d15N", "d13C" and used to set automatically Rstandards (see the Section "Ratios for reference standards" for more details). Alternatively, a numeric value to use for Rstandard, e.g. 0.0036765.

Details

- Figure 1 in Coplen, Tyler B. "Guidelines and Recommended Terms for Expression of Stable-Isotope-Ratio and Gas-Ratio Measurement Results." Rapid Communications in Mass Spectrometry 25, no. 17 (September 15, 2011): 2538–60. https://doi.org/10.1002/rcm.5129.

- Table 2.1 in Fry, Brian. Stable Isotope Ecology. New York: Springer-Verlag, 2006. //www.springer.com/gp/book/97803873

Value

A vector of same length of x, containing the proportion (numeric between 0 and 1) of heavy isotope based on the delta values and the Rstandard provided.

Ratios for reference standards

The ratios for reference standards are taken from the Table 2.1 from Fry 2006. Note that the values used for oxygen isotopes are from the standard mean ocean water (SMOW).

Standards recognized by this function are: `c("d15N", "d2H", "d13C","d17O.SMOW", "d18O.SMOW", "d33S", "d34S", "d36S")`

Examples

```
deltas <- c(78, 5180, 263, 1065, NA, 153, 345)

# Rstandard can be specified with a string for some preset references
prop15N <- delta2prop(deltas, "d15N")
prop13C <- delta2prop(deltas, "d13C")

# Rstandard can also be specified manually for non-preset references
prop15N_manual <- delta2prop(deltas, 0.0036765)
prop13C_manual <- delta2prop(deltas, 0.011180)

# Call delta2prop() to get the detail of available references
delta2prop()
```

`dic` *Calculate DIC from a model output*

Description

Note that DIC might not be indicated for network models, as the posteriors are often not multinormal distributions.

Usage

```
dic(..., weight = TRUE)
```

Arguments

`...`	One or several `mcmc.list` objects, output(s) from `run_mcmc`.
`weight`	Boolean, if TRUE calculate DIC weights based on Link and Barker 2010 (Link, W. A., and R. J. Barker. 2010. Bayesian Inference With Ecological Applications. Amsterdam Boston Heidelberg London: Elsevier/Academic Press).

Details

LOO is probably not a good choice either since the data is akin to a time series (so data points are not independent). Maybe WAIC could be an option? (TODO: read about this.)

DIC is calculated as:

DIC = Dbar + pD

where D are deviance values calculated as -2 * loglik for each MCMC iteration, Dbar is the mean deviance value and pD is the effective number of parameters in the model and can be calculated as var(D)/2 (Gelman 2003).

Value

A tibble with one row per `mcmc.list` object provided in `....` This tibble is sorted by DIC, so the row order might be different from the `mcmc.list` objects order.

Examples

```
# Define two different models
m1 <- aquarium_mod
m2 <- set_topo(m1, c("NH4 -> algae -> daphnia -> NH4", "algae -> NH4"))
m2 <- set_priors(m2, priors(m1))
m2 <- set_priors(m2, normal_p(0, 0.5), "upsilon_algae_to_NH4")
# Run the models
r1 <- run_mcmc(m1, chains = 2)
r2 <- run_mcmc(m2, chains = 2)
# Model comparison with DIC
dic(r1, r2)
```

`eelgrass` *Eelgrass phosphate incorporation data (McRoy & Barsdate 1970)*

Description

Dataset built from the article "Phosphate absorption in eelgrass" by McRoy and Barsdate (1970)

Usage

```
eelgrass
```

Format

Tibble with columns

light_treatment Light treatment: "light" or "dark".

addition_site The location where 32P phosphate was added: in the "upper" water compartment or in the "lower" water compartment.

compartment Obsered compartment, one of "leaves_stem", "roots_rhizome", "upper_water", or "lower_water".

time_min Elapsed time in minutes since the 32P addition.

n_32P_per_mg Number of 32P atoms per mg (estimated from Figure 2 of the original paper).

mass_mg Compartment mass in mg (taken from Table 1 of the original paper). Assumed constant during the experiment.

n_32P Number of 32P atoms in the compartment. Calculated from the two previous columns.

Details

In brief, the experimental setup consists in individual eelgrass plants placed in 250 ml containers. Each container is partitioned by a layer of paraffin into an upper water compartment (containing the leaves and stems) and a lower water compartment (containing the roots and rhizomes).

Radioactive phosphorus (32P) is added as phosphate either in the upper or lower water compartment in each container. Containers were incubated either in light or dark conditions.

Tissue samples were collected and dried at various time points and 32P activity was measured (Figure 2 in the original paper). Biomass estimates in initial conditions were given in Table 1 of the original paper.

Data preparation

The data for 32P abundance per mg is extracted from Figure 2 of the original article. Atom counts per mg were derived from cpm per mg using a half-life value of 14.268 days for 32P.

For simplicity and in order to be able to match the 32P data with the biomass data (see below), only four compartments are considerd in the package dataset. Upper and lower water compartments match the compartments from the original article. "Leaf and stem" pools the original compartments "leaf tip", "leaf middle", "leaf base", and "stem". "Roots and rhizome" pools the original compartments "root" and "rhizome". Pooling is done by averaging the cpm per mg data, thereby making the rough approximation that each component of the pool contributes the same biomass as the other components.

The biomass data is taken from Table 1 in the original paper. Experimental containers had 160 cc of seawater in the upper compartment and 80 cc of seawater in the lower compartment. Based on comparison with data from Risgaard-Petersen 1998, I assumed that the biomasses for tissues were given in dry weight. I assumed that this was also the case for the cpm/mg data (i.e. cpm/mg of dry weight).

Source

Data was taken from the figures and tables of the original paper. The original paper is: McRoy, C. Peter, and Robert J. Barsdate. “Phosphate Absorption in Eelgrass1.” Limnology and Oceanography 15, no. 1 (January 1, 1970): 6–13. https://doi.org/10.4319/lo.1970.15.1.0006.

exponential_p *Define an exponential prior*

Description

Define an exponential prior

Usage

```
exponential_p(lambda)
```

Arguments

lambda	Lambda parameter (rate) of the exponential distribution. The mean of the exponential distribution is 1/lambda.

Value

A list defining the prior.

Examples

```
exponential_p(0.5)
```

filter *Filter (alias for filter function from dplyr)*

Description

Filter (alias for filter function from dplyr)

Arguments

.data	Data to filter.
...	Passed to dplyr::filter.
preserve	Ignored.

Value

See the returned value for dplyr::filter.

`filter.ppcNetworkModel` *Filter method for output of tidy_data_and_posterior_predict()*

Description

Filter method for output of tidy_data_and_posterior_predict()

Usage

```
## S3 method for class 'ppcNetworkModel'
filter(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	A ppcNetworkModel object.
`...`	Passed to dplyr::filter.
`.preserve`	Ignored.

Value

A pccNetworkModel object filtered appropriately based on the [["vars"]] tibble.

`filter_by_group` *Filter a tibble based on the "group" column*

Description

This function can be used to filter any tibble (e.g. network model object) that has a "group" column. See the Examples for more details and syntax.

Usage

```
filter_by_group(.data, ...)
```

Arguments

`.data`	A tibble that has a ‘group‘ column, such as a ‘networkModel‘ object.
`...`	Conditional expressions for filtering (see the Examples).

Value

A tibble similar to the input object, but with rows filtered based on ....

Examples

```
trini_mod
trini_mod$group
groups(trini_mod)
filter_by_group(trini_mod, stream == "LL", transect == "transect.1")
filter_by_group(trini_mod, transect == "transect.1")
## Not run:
# The code below would raise an error because there is no "color" grouping variable.
filter_by_group(trini_mod, color == "red")

## End(Not run)
```

---

format.prior *Pretty formatting of a* prior *object*

Description

Pretty formatting of a prior object

Usage

```
## S3 method for class 'prior'
format(x, ...)
```

Arguments

x	An object of class prior.
...	Not used.

Value

A character string for pretty printing of a prior.

---

format.prior_tibble *Pretty formatting of a* prior_tibble *object*

Description

Pretty formatting of a prior_tibble object

Usage

```
## S3 method for class 'prior_tibble'
format(x, ...)
```

Arguments

x	An object of class prior_tibble.
...	Not used.

Value

A character string for pretty printing of a prior tibble.

---

gamma_p *Define a gamma prior*

Description

Note the name of the function to define a prior (gamma_p), in order to avoid confusion with the R mathematical function gamma.

Usage

```
gamma_p(alpha, beta)
```

Arguments

alpha	Shape parameter (equivalent to the shape parameter of R's rgamma).
beta	Rate parameter (equivalent to the rate parameter of R's rgamma).

Value

A list defining the prior.

Examples

```
gamma_p(9, 2)
hist(sample_from_prior(gamma_p(9, 2), 1e3))
```

ggflows *A quick-and-dirty way of visualizing relative flows in a network*

Description

A quick-and-dirty way of visualizing relative flows in a network

Usage

```
ggflows(x, layout = "auto", edge = "fan", max_width, legend = TRUE, ...)
```

Arguments

x	A tibble with the flow estimates, with columns "from", "to", and "flow".
layout	Optional, layout to use (e.g. "sugiyama", "kk", "stress")
edge	"curve" (the default), "line" or "fan".
max_width	Optional, numeric giving the maximum edge width (minimum width is always 1).
legend	Boolean, display edge width legend?
...	Not used.

Value

A ggplot2 plot.

Examples

```
if (requireNamespace("ggraph")) {
  z <- tibble::tribble(
              ~from,               ~to,            ~flow,
    "leavesAndStem", "rootsAndRhizome", 333.929866077124,
       "lowerWater", "rootsAndRhizome", 4425.15780019304,
  "rootsAndRhizome",   "leavesAndStem", 525.208837577916,
       "upperWater",   "leavesAndStem", 11224.0814971855
  )
  ggflows(z)
  ggflows(z, max_width = 15)
}
```

ggtopo *Plot a topology*

Description

A quick plot using ggraph

Usage

```
ggtopo(x, layout = "auto", edge = "fan", ...)
```

Arguments

x	A network model or a topology matrix.
layout	Optional, layout to use (e.g. "sugiyama", "kk", "stress")
edge	"fan" (the default) or "line" or "curve".
...	Passed to the methods.

Value

A ggplot2 plot.

Examples

```
if (requireNamespace("ggraph")) {
  ggtopo(aquarium_mod, edge = "line")
}
```

ggtopo.networkModel *Plot a network topology*

Description

A quick plot using ggraph

Usage

```
## S3 method for class 'networkModel'
ggtopo(x, layout = "auto", edge = "fan", ...)
```

Arguments

x	A topology matrix.
layout	Optional, layout to use (e.g. "sugiyama", "kk", "stress")
edge	"curve" (the default) or "line".
...	Not used for now.

Value

A ggplot2 plot.

Examples

```
if (requireNamespace("ggraph")) {
  ggtopo(aquarium_mod, edge = "line")
  ggtopo(trini_mod)
}
```

`ggtopo.topology` *Plot a topology*

Description

A quick plot using ggraph

Usage

```
## S3 method for class 'topology'
ggtopo(x, layout = "auto", edge = "fan", ...)
```

Arguments

`x`	A topology matrix.
`layout`	Optional, layout to use (e.g. "sugiyama", "kk", "stress")
`edge`	"curve" (the default), "line" or "fan".
`...`	Not used for now.

Value

A ggplot2 plot.

Examples

```
if (requireNamespace("ggraph")) {
  z <- topo(aquarium_mod)
  ggtopo(z)
  ggtopo(z, edge = "line")

  z <- topo(trini_mod)
  ggtopo(z)

  # For finer control, one can build a tbl_graph from the topology and
  # use ggraph directly
  x <- as_tbl_graph(z)
  library(ggraph)
```

```
  ggraph(x) + geom_edge_link()
}
```

`groups.networkModel` *Get the grouping for a* `networkModel` *object*

Description

Get the grouping for a `networkModel` object

Usage

```
## S3 method for class 'networkModel'
groups(x)
```

Arguments

`x`	A `networkModel` object.

Value

A tibble giving the grouping variable(s) for the input network model. This tibble is in the same order as the rows of the input network model. If the input network model did not have any grouping variable, returns `NULL`.

Examples

```
groups(aquarium_mod)
groups(trini_mod)
```

`hcauchy_p` *Define a half-Cauchy prior (on [0;+Inf])*

Description

Define a half-Cauchy prior (on [0;+Inf])

Usage

```
hcauchy_p(scale)
```

Arguments

`scale`	Median of the half-Cauchy distribution.

Value

A list defining the prior.

Examples

```
hcauchy_p(scale = 0.5)
```

`lalaja` *Dataset for nitrogren fluxes in a Trinidadian mountain stream (Collins 2016)*

Description

Dataset built from the article "Fish introductions and light modulate food web fluxes in tropical streams: a whole-ecosystem experimental approach" by Collins et al. (2016).

Usage

```
lalaja
```

Format

Tibble with columns

stream Stream identity. It is always "UL" (for "Upper lalaja") in this dataset. See the model `trini_mod` also shipped with the package for the full dataset from the original Collins et al. study, including data from the Lower Lajaja stream.

transect Transect identity. Three transects were sampled downstream of the drip location: `c("transect.1", "transect.2", "transect.3")`.

compartment Foodweb compartments. Eight compartments are included in this dataset: "NH4", dissolved ammonium; "NH3", dissolved nitrate; "epi", epilithon (primary producers growing on the surface of rocks on the stream bed); "FBOM", fine benthic organic material; "tricor", *Tricorythodes* (invertebrate); "pseph", *Psephenus* (invertebrate); "petro", *Petrophila* (invertebrate); "arg", *Argia* (invertebrate).

mgN.per.m2 Size of compartment, in mg of nitrogen per m2.

prop15N Proportion of 15N nitrogen in a compartment nitrogen pool (i.e. 15N / (15N + 14N)).

time.days Sampling time, in days.

Details

In the original study, 15N-enriched ammonium was dripped into two mountain streams in Trinidad (Upper Lalaja stream and Lower Lalaja stream) and samples of the different foodweb compartments were taken during the drip and after the drip in several transects in each stream. The transects were located at different locations downstream of each drip. There were three transects per stream. The drip phase lasted 10 days, and the post-drip phase lasted 30 days. The complete dataset from the original study is available in the `trini_mod` model shipped with the `isotracer` package.

The `lalaja` dataset is a subset of the full dataset and is used for illustrative purpose in the "Trinidadian streams" case study, which is part of the documentation of `isotracer`. It contains only the data for the Upper Lalaja stream, and for some but not all of the foodweb compartments.

For more details about the dripping regime and how to use this dataset in a network model, one should refer to the case study in the `isotracer` package documentation.

Source

This network model contains data from the original article: Collins, Sarah M., Steven A. Thomas, Thomas Heatherly, Keeley L. MacNeill, Antoine O.H.C. Leduc, Andrés López-Sepulcre, Bradley A. Lamphere, et al. 2016. "Fish Introductions and Light Modulate Food Web Fluxes in Tropical Streams: A Whole-Ecosystem Experimental Approach." Ecology, <doi:10.1002/ecy.1530>.

This dataset was also used in the paper: López-Sepulcre, Andrés, Matthieu Bruneaux, Sarah M. Collins, Rana El-Sabaawi, Alexander S. Flecker, and Steven A. Thomas. 2020. "A New Method to Reconstruct Quantitative Food Webs and Nutrient Flows from Isotope Tracer Addition Experiments." The American Naturalist 195 (6): 964–85. <doi:10.1086/708546>.

`li2017` *Protein degradation in Arabidopsis plants (Li et al. 2017)*

Description

Dataset built from the Dryad depository entry associated with the article "Protein degradation rate in Arabidopsis thaliana leaf growth and development" by Li et al. (2017)

Usage

```
li2017
```

Format

`li2017` is the main dataset and is a tibble with columns:

prot_id Protein identifier. Can be matched to a more explicit protein description in `li2017_prots`.

sample Sample identity. Different samples were used for relative abundance measurements and labelled fraction measurements.

rel_abundance Relative abundance compared to a reference sample.

labeled_fraction Proportion of 15N in the protein.

time_day Time elapsed since growth medium switch to 15N, in days.

leaf_id Leaf identity (3rd, 5th, or 7th leaf of individual plants).

`li2017_prots` maps protein identifiers to protein descriptions and is a tibble with columns:

prot_id Protein identifier. Can be matched with the same column in `li2017`.

description Protein description

`li2017_counts` is a summary table counting the number of available data points for relative abundance and labelled fraction for each protein in `li2017`. It is a tibble with columns:

prot_id Protein identifier. Can be matched with the same column in `li2017`.

n_abundance_data Number of relative abundance data points for a given protein.

n_labelling_data Number of labelled fraction data points for a given protein.

Details

In this study, the authors used a growth medium containing 15N to grow 21-day old Arabidopsis plants which were grown on a natural 14N/15N medium until that day. The third, fifth and seventh leaves were sampled from individuals at different time points after the medium switch (0, 1, 3 and 5 days). Proteins were identified and labelled fractions were measured using mass spectrometry. Relative protein abundances were determined in comparison with a reference sample.

The aim of the authors was to quantify in vivo degradation rates for as many proteins as possible (1228 proteins in the original paper) and examine which determinants had an effect or not on protein degradation rates (e.g. protein domains, protein complex membership, ...).

Three datasets were extracted from the large dataset available on Dryad for packaging inside isotracer: `li2017`, `li2017_prots`, and `li2017_counts`.

Source

Data was taken from the following Dryad repository: Li, Lei, Clark J. Nelson, Josua Troesch, Ian Castleden, Shaobai Huang, and A. Harvey Millar. "Data from: Protein Degradation Rate in Arabidopsis Thaliana Leaf Growth and Development." Dryad, 2018. https://doi.org/10.5061/DRYAD.Q3H85.

The Dryad repository was associated with the following paper: Li, Lei, Clark J. Nelson, Josua Trösch, Ian Castleden, Shaobai Huang, and A. Harvey Millar. "Protein Degradation Rate in Arabidopsis Thaliana Leaf Growth and Development." The Plant Cell 29, no. 2 (February 1, 2017): 207–28. https://doi.org/10.1105/tpc.16.00768.

`Math.mcmc.list` *Math generics for mcmc.list objects*

Description

Math generics for mcmc.list objects

Usage

```
## S3 method for class 'mcmc.list'
Math(x, ...)
```

Arguments

x	mcmc.list object
...	Other arguments passed to corresponding methods

Value

A mcmc.list object (with the added class derived.mcmc.list).

---

mcmc_heatmap *Draw a heatmap based on the correlations between parameters*

Description

Note that the colors represent the strength of the correlations (from 0 to 1), but do not inform about their sign. The method used to calculate correlation coefficients is Spearman's rho.

Usage

```
mcmc_heatmap(x, col = NULL, ...)
```

Arguments

x	A coda::mcmc.list object.
col	Optional, vectors of colors defining the color ramp. Default uses the divergent palette "Blue-Red 2" from the colorspace package.
...	Passed to heatmap.

Value

Called for side effect (plotting).

missing_priors *Get a table with parameters which are missing priors*

Description

Get a table with parameters which are missing priors

Usage

```
missing_priors(nm)
```

Arguments

nm	A networkModel object.

Value

A tibble containing the parameters which are missing a prior. If no priors are missing, the tibble contains zero row.

Examples

```
# Using a subset of the topology from the Trinidad case study
m <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi, FBOM", "epi -> petro, pseph")

# No prior is set by default
priors(m)

# Set some priors
m <- set_priors(m, normal_p(0, 10), "lambda")
priors(m)

# Which parameters are missing a prior?
missing_priors(m)
```

new_networkModel *Create an empty network model*

Description

The first step in building a network model is to create a new, empty networkModel object. This model can then be completed using functions such as set_topo(), set_init(), etc...

Usage

```
new_networkModel(quiet = FALSE)
```

Arguments

quiet	Boolean, if FALSE print a message indicating which distribution family is used for proportions.

Value

An empty networkModel object. It is basically a zero-row tibble with the appropriate columns.

Examples

```
m <- new_networkModel()
m
class(m)
```

---

normal_p *Define a truncated normal prior (on [0;+Inf])*

Description

Define a truncated normal prior (on [0;+Inf])

Usage

```
normal_p(mean, sd)
```

Arguments

mean	Mean of the untruncated normal.
sd	Standard deviation of the untruncated normal.

Value

A list defining the prior.

Examples

```
normal_p(mean = 0, sd = 4)
```

obj_sum.prior *Function used for displaying* prior *object in tibbles*

Description

Function used for displaying `prior` object in tibbles

Usage

```
## S3 method for class 'prior'
obj_sum(x)
```

Arguments

`x`	An object of class `prior`.

Value

Input formatted with `format(x)`.

Ops.mcmc.list *Ops generics for* mcmc.list *objects*

Description

Ops generics for `mcmc.list` objects

Usage

```
## S3 method for class 'mcmc.list'
Ops(e1, e2)
```

Arguments

`e1`	First operand
`e2`	Second operand

Value

A `mcmc.list` object (with the added class `derived.mcmc.list`).

Examples

```
## Not run:
# aquarium_run is a coda::mcmc.list object shipped with the isotracer package
a <- aquarium_run
plot(a)
# The calculations below are just given as examples of mathematical
# operations performed on an mcmc.list object, and do not make any sense
# from a modelling point of view.
plot(a[, "upsilon_algae_to_daphnia"] - a[, "lambda_algae"])
plot(a[, "upsilon_algae_to_daphnia"] + a[, "lambda_algae"])
plot(a[, "upsilon_algae_to_daphnia"] / a[, "lambda_algae"])
plot(a[, "upsilon_algae_to_daphnia"] * a[, "lambda_algae"])
plot(a[, "upsilon_algae_to_daphnia"] - 10)
plot(a[, "upsilon_algae_to_daphnia"] + 10)
plot(a[, "upsilon_algae_to_daphnia"] * 10)
plot(a[, "upsilon_algae_to_daphnia"] / 10)
plot(10 - a[, "upsilon_algae_to_daphnia"])
plot(10 + a[, "upsilon_algae_to_daphnia"])
plot(10 * a[, "upsilon_algae_to_daphnia"])
plot(10 / a[, "upsilon_algae_to_daphnia"])

## End(Not run)
```

`Ops.prior` *Implementation of the '==' operator for priors*

Description

Implementation of the '==' operator for priors

Usage

```
## S3 method for class 'prior'
Ops(e1, e2)
```

Arguments

`e1, e2`	Objects of class "prior".

Value

Boolean (or throws an error for unsupported operators).

Examples

```
p <- constant_p(0)
q <- constant_p(4)
p == q

p <- hcauchy_p(2)
q <- hcauchy_p(2)
p == q
```

`Ops.topology` *Ops generics for* `topology` *objects*

Description

Ops generics for `topology` objects

Usage

```
## S3 method for class 'topology'
Ops(e1, e2)
```

Arguments

`e1`	First operand
`e2`	Second operand

Value

Boolean (or throws an error for unsupported operators).

Examples

```
topo(aquarium_mod) == topo(trini_mod)
topo(aquarium_mod) == topo(aquarium_mod)
```

params *Return the parameters of a network model*

Description

Return the parameters of a network model

Usage

```
params(nm, simplify = FALSE)
```

Arguments

nm	A networkModel object.
simplify	If TRUE, return a vector containing the names of all model parameters (default: FALSE).

Value

A tibble containing the parameter names and their current value (if set). If simplify is TRUE, only return a sorted character vector containing the parameters names.

Examples

```
params(aquarium_mod)
params(trini_mod)
params(trini_mod, simplify = TRUE)
```

pillar_shaft.prior *Function used for displaying* prior *object in tibbles*

Description

Function used for displaying prior object in tibbles

Usage

```
## S3 method for class 'prior'
pillar_shaft(x, ...)
```

Arguments

x	An object of class prior.
...	Not used.

Value

An object prepared with pillar::new_pillar_shaft_simple.

---

plot.networkModel *Plot observations/trajectories/predictions from a network model*

Description

Plot observations/trajectories/predictions from a network model

Usage

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

Arguments

x	A networkModel object.
...	Passed to plot_nm.

Value

Called for side effect (plotting).

---

plot.ready_for_unit_plot *Plot output from* split_to_unit_plot

---

Description

Plot output from split_to_unit_plot

Usage

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

Arguments

x	A ready_for_unit_plot object.
...	Passed to plot_nm.

Value

Called for side effect (plotting).

posterior_predict *Draw from the posterior predictive distribution of the model outcome*

Description

Draw from the posterior predictive distribution of the model outcome

Usage

```
posterior_predict(object, ...)
```

Arguments

object	Model from which posterior predictions can be made.
...	Passed to the appropriate method.

Value

Usually methods will implement a draw parameter, and the returned object is a "draw" by N matrix where N is the number of data points predicted per draw.

posterior_predict.networkModelStanfit *Draw from the posterior predictive distribution of the model outcome*

Description

Draw from the posterior predictive distribution of the model outcome

Usage

```
## S3 method for class 'networkModelStanfit'
posterior_predict(object, newdata, draw = NULL, cores = NULL, ...)
```

Arguments

object	A networkModelStanfit object.
newdata	Should be the model used to fit the networkStanfit object.
draw	Integer, number of draws to perform from the posterior. Default is 100.
cores	Number of cores to use for parallel calculations. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).
...	Not used for now.

Value

A "draw" by N matrix where N is the number of data points predicted per draw.

`predict.networkModel` *Add a column with predictions from a fit*

Description

Add a column with predictions from a fit

Usage

```
## S3 method for class 'networkModel'
predict(
  object,
  fit,
  draws = NULL,
  error.draws = 5,
  probs = 0.95,
  cores = NULL,
  dt = NULL,
  grid_size = NULL,
  at = NULL,
  end = NULL,
  ...
)
```

Arguments

`object`	Network model
`fit`	Model fit (mcmc.list object)
`draws`	Integer, number of draws from the posteriors
`error.draws`	Integer, number of draws from the error distribution, for a given posterior draw.
`probs`	Credible interval (default 0.95).
`cores`	Number of cores to use for parallel calculations. Default is `NULL`, which means to use the value stored in `options()[["mc.cores"]]` (or 1 if this value is not set).
`dt, grid_size`	Time step size or grid points, respectively.
`at`	Timepoints at which the predictions should be returned.
`end`	Final timepoint used in the projections.
`...`	Not used.

Value

A network model object with an added column `"prediction"`.

print.networkModel *Print method for* networkModel *objects*

Description

Print method for networkModel objects

Usage

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

Arguments

x	A networkModel object.
...	Passsed to the next method.

Value

Called for the side effect of printing a network model object.

print.prior *Pretty printing of a* prior *object*

Description

Pretty printing of a prior object

Usage

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

Arguments

x	An object of class prior.
...	Not used.

Value

Mostly called for its side effect of printing, but also returns its input invisibly.

print.prior_tibble *Pretty printing of a* prior_tibble *object*

Description

Pretty printing of a prior_tibble object

Usage

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

Arguments

x	An object of class prior_tibble.
...	Not used.

Value

Mostly called for its side effect of printing, but also returns its input invisibly.

print.topology *Pretty printing of a* topology *object*

Description

Pretty printing of a topology object

Usage

```
## S3 method for class 'topology'
print(x, help = TRUE, ...)
```

Arguments

x	An object of class topology.
help	If TRUE, display a short help after the topology object explaining e.g. the steady state or the split compartment symbols.
...	Not used.

Value

Mostly called for its side effect (printing).

priors *Return the tibble containing the priors of a networkModel*

Description

Return the tibble containing the priors of a networkModel

Usage

```
priors(nm, fix_set_params = FALSE, quiet = FALSE)
```

Arguments

nm	A networkModel object.
fix_set_params	If TRUE, parameters for which a value is set are given a fixed value (i.e. their prior is equivalent to a point value).
quiet	Boolean to control verbosity.

Value

A tibble giving the current priors defined for the input network model.

Examples

```
priors(aquarium_mod)
priors(trini_mod)
```

project *Calculate the trajectories of a network model*

Description

Calculate the trajectories of a network model

Usage

```
project(
  nm,
  dt = NULL,
  grid_size = NULL,
  at = NULL,
  end = NULL,
  flows = "no",
  cached_ts = NULL,
  cached_ee = NULL
)
```

Arguments

nm	A networkModel object.
dt, grid_size	Either the time step size for trajectory calculations (dt) or the number of points for the calculation (grid_size) can be provided. If none is provided, then a default grid size of 256 steps is used.
at	Optional, vector of time values at which the trajectory must be evaluated.
end	Time value for end point. If not provided, the last observation or event is used.
flows	Return flow values? The default is "no" and no flows are calculated. Other values are "total" (total flows summed up from beginning to end timepoint), "average" (average flows per time unit, equal to total flows divided by the projection duration), and "per_dt" (detailled flow values are returned for each interval dt of the projection).
cached_ts, cached_ee	Used for optimization by other functions, not for use by the package user.

Value

A network model object with a "trajectory" column.

Examples

```
m <- aquarium_mod
m <- set_params(m, sample_params(m))
z <- project(m)
z <- project(m, flows = "per_dt")
z <- project(m, flows = "total")
z <- project(m, flows = "average")
```

---

prop_family *Return the distribution family for observed proportions*

Description

Return the distribution family for observed proportions

Usage

```
prop_family(nm, quiet = FALSE)
```

Arguments

nm	A networkModel object.
quiet	Boolean for being quiet about explaining the role of eta (default is FALSE).

Value

A character string describing the distribution family used to model observed proportions.

Examples

```
prop_family(aquarium_mod)
prop_family(trini_mod)
```

`quick_sankey` *Draw a Sankey plot with basic defaults*

Description

Draw a Sankey plot with basic defaults

Usage

```
quick_sankey(flows, ...)
```

Arguments

`flows`	A tibble containing flows (output from `tidy_flows`). For now it should have an "average_flow" column in the tibbles of the "flows" list column.
`...`	Passed to `sankey`.

Value

Mostly called for its side effect (plotting), but also returns invisible the scene object describing the Sankey plot. Note that the structure of this object is experimental and might change in the future!

`run_mcmc` *Run a MCMC sampler on a network model using Stan*

Description

Run a MCMC sampler on a network model using Stan

Usage

```
run_mcmc(
  model,
  iter = 2000,
  chains = 4,
  method = "matrix_exp",
  euler_control = list(),
  cores = NULL,
  stanfit = FALSE,
  ...
)
```

Arguments

model	A networkModel.
iter	A positive integer specifying the number of iterations for each chain (including warmup). The default is 2000.
chains	A positive integer specifying the number of Markov chains. The default is 4.
method	A character string indicating the method to use to solve ODE in the Stan model; available methods are "matrix_exp" and "euler". The default is "matrix_exp", which uses matrix exponential and is reasonably fast for small networks. For large networks, the "euler" method can be used. It implements a simple forward Euler method to solve the ODE and can be faster than the matrix exponential approach, but extra caution must be taken to check for numerical accuracy (e.g. testing different dt time step values, ensuring that the product between dt and the largest transfer rates expected from the priors is always very small compared to 1).
euler_control	An optional list containing extra parameters when using method = "euler". Allowed list elements are "dt" and "grid_size", which are respectively the time step size for trajectory calculations ("dt") or the number of points for the calculation ("grid_size"). Only one of "dt" or "grid_size" can be specified, not both. If none is provided, a default grid size of 256 steps is used.
cores	Number of cores to use for parallel use. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).
stanfit	If TRUE, returns a 'stanfit' object instead of the more classical 'mcmc.list' object. Note that when an 'mcmc.list' object is returned, the original 'stanfit' object is still accessible as an attribute of that object (see Examples).
...	Arguments passed to 'rstan::sampling' (e.g. iter, chains).

Value

An object of class 'stanfit' returned by 'rstan::sampling' if stanfit = TRUE, otherwise the result of converting this stanfit object with stanfit_to_named_mcmclist (i.e. an object of class networkModelStanfit and mcmc.list, which still carries the original 'stanfit' object stored as an attribute).

Examples

```
aquarium_mod
## Not run:
  # The 'aquarium_run' object is shipped with the package, so you don't
  # actually need to run the line below to obtain it
  aquarium_run <- run_mcmc(aquarium_mod)

  plot(aquarium_run)
  summary(aquarium_run)

  # The original stanfit object returned by Stan
  sfit <- attr(aquarium_run, "stanfit")
  sfit

  # The stanfit object can be used for diagnostics, LOO cross-validation, etc.
  rstan::loo(sfit)

## End(Not run)
```

---

sample_from *Generate samples from a network model*

Description

Generate samples from a network model

Usage

```
sample_from(
  nm,
  at,
  dt = NULL,
  grid_size = NULL,
  end = NULL,
  error.draws = 1,
  cached_ts = NULL,
  cached_ee = NULL
)
```

Arguments

nm	A networkModel object.
at	Vector of time values at which the samples should be taken.
dt, grid_size	Time step size or grid points, respectively.
end	Final timepoint used in the projections.

`error.draws` Integer, number of draws from the error distribution for each sample (default: 1).

`cached_ts, cached_ee`
Used for optimization by other functions, not for use by the package user.

Value

A tibble containing the generated samples.

Examples

```
library(magrittr)
mod <- new_networkModel() %>%
   set_topo("NH4 -> algae -> daphnia -> NH4")
inits <- tibble::tribble(
     ~comps, ~sizes, ~props, ~treatment,
      "NH4",    0.2,    0.8,    "light",
    "algae",      1,  0.004,    "light",
  "daphnia",      2,  0.004,    "light",
      "NH4",    0.5,    0.8,     "dark",
    "algae",    1.2,  0.004,     "dark",
  "daphnia",    1.3,  0.004,     "dark")
mod <- set_init(mod, inits, comp = "comps", size = "sizes",
                prop = "props", group_by = "treatment")
mod <- add_covariates(mod, upsilon_NH4_to_algae ~ treatment)
mod <- mod %>%
  set_params(c("eta" = 0.2, "lambda_algae" = 0, "lambda_daphnia" = 0,
               "lambda_NH4" = 0, "upsilon_NH4_to_algae|light" = 0.3,
               "upsilon_NH4_to_algae|dark" = 0.1,
               "upsilon_algae_to_daphnia" = 0.13,
               "upsilon_daphnia_to_NH4" = 0.045, "zeta" = 0.1))
spl <- mod %>% sample_from(at = 1:10)
spl
```

---

sample_from_prior *Sample from a prior object*

Description

Sample from a prior object

Usage

```
sample_from_prior(x, n = 1)
```

Arguments

`x` A `prior` object.

`n` Integer, number of samples to draw.

Value

A numeric vector of length n.

Examples

```
sample_from_prior(constant_p(1))
sample_from_prior(constant_p(1), 10)
sample_from_prior(hcauchy_p(0.5), 1)
hist(sample_from_prior(hcauchy_p(0.5), 20))
hist(sample_from_prior(uniform_p(0, 3), 1000))
hist(sample_from_prior(scaled_beta_p(3, 7, 2), 1e4))
```

---

sample_params *Sample parameter values from priors*

Description

Sample parameter values from priors

Usage

```
sample_params(nm)
```

Arguments

nm	A networkModel object.

Value

A named vector containing parameter values.

Examples

```
library(magrittr)

p <- sample_params(aquarium_mod)
p

proj <- aquarium_mod %>% set_params(p) %>% project(end = 10)
plot(proj)
```

sankey *Draw a Sankey plot for a network and estimated flows*

Description

Draw a Sankey plot for a network and estimated flows

Usage

```
sankey(
  topo,
  nodes = NULL,
  flows = NULL,
  layout = NULL,
  new = TRUE,
  debug = FALSE,
  node_f = 1,
  edge_f = 1,
  node_s = "auto",
  edge_n = 32,
  cex_lab = NULL,
  cex.lab = NULL,
  fit = TRUE
)
```

Arguments

topo	A topology.
nodes	Optional, a tibble containing the properties of the nodes. It should have a ‘comp‘ column with the same entries as the topology. It cannot have ‘x‘ and ‘y‘ entries. If it has a ‘label‘ entry, it will replace the ‘comp‘ values for node labels.
flows	A tibble containing the values of the flows in the topology. If NULL (the default), all flows have same width in the plot.
layout	String, node-placing algorithm to use from the ggraph package (e.g. "stress"). The ggraph package itself uses some algoritms from the igraph package. See the Details in the help of layout_tbl_graph_igraph for available algorithms. The ggraph package must be installed for this argument to be taken into account. Currently, only the "left2right" and "stress" layout are implemented in detail, and any other layout will use rough defaults for the aesthetic adjustments. Other layouts which are kind of working are "kk", "lgl", "fr", "dh", "mds". Some of those produce non-reproducible node locations (at least I haven’t managed to reproduce them even by setting the RNG seed before calling the function).
new	Boolean, create a new page for the plot?
debug	Boolean, if TRUE then draw a lot of shapes to help with debugging.
node_f, edge_f	Multiplicative factor to adjust node and edge size.

node_s	String defining how node size is calculated. The effect of the string also depends on the chosen layout.
edge_n	Integer, number of interpolation points along each edge.
cex_lab, cex.lab	Expansion factor for label size (both arguments are synonyms).
fit	Boolean, if TRUE try to fit all the graphical elements inside the canvas.

Value

Mostly called for its side effect (plotting), but also returns invisible the scene object describing the Sankey plot. Note that the structure of this object is experimental and might change in the future!

Examples

```
library(magrittr)

topo <- topo(trini_mod)
sankey(topo, debug = TRUE)
sankey(topo, layout = "stress")
sankey(topo(aquarium_mod), layout = "stress", edge_f = 0.5)

m <- new_networkModel() %>%
    set_topo(c("subs -> NH3 -> subs",
               "NH3 -> Q, E", "E -> Q -> E",
               "E -> D, M")) %>%
    set_steady("subs") %>%
    set_prop_family("normal_sd")
ggtopo(m)
sankey(topo(m), layout = "stress")

# Debug visualization

## Helper functions
flows_from_topo <- function(x) {
    x <- unclass(x) # Remove the "topo" class to treat it as a matrix
    n_comps <- ncol(x)
    links <- which(x > 0)
    from <- links %/% n_comps + 1
    to <- links %% n_comps
    links <- tibble::tibble(from = from, to = to)
    for (i in seq_len(nrow(links))) {
        if (links$to[i] == 0) {
            links$from[i] <- links$from[i] - 1
            links$to[i] <- n_comps
        }
        stopifnot(x[links$to[i], links$from[i]] > 0)
    }
    flows <- tibble::tibble(from = colnames(x)[links$from],
                            to = rownames(x)[links$to])
    return(flows)
}
```

```
nodes_from_topo <- function(x) {
    nodes <- tibble::tibble(comp = colnames(x),
                            label = colnames(x))
    return(nodes)
}

t <- topo(trini_mod)
nodes <- nodes_from_topo(t)
nodes$label <- as.list(nodes$label)
nodes$label[[2]] <- latex2exp::TeX("$\\beta$")
nodes$size <- runif(nrow(nodes), 1, 2)
flows <- flows_from_topo(t)
flows$width <- runif(nrow(flows), 0.2, 2)
z <- sankey(t, nodes = nodes, flows = flows, layout = "left2right",
            debug = TRUE, node_f = 1, edge_f = 0.9, edge_n = 32,
            cex_lab = 1.5)

# Stress layout
y <- new_networkModel() %>%
        set_topo(c("subs -> NH3 -> subs",
                   "NH3 -> Q, E", "E -> Q -> E",
                   "E -> D, M")) %>%
        set_steady("subs") %>%
            set_prop_family("normal_sd")
y <- topo(y)
nodes <- nodes_from_topo(y)
nodes$size <- runif(nrow(nodes), 1, 10)
ggtopo(y, edge = "fan")
flows <- flows_from_topo(y)
flows$width <- runif(nrow(flows), 0.2, 5)
z <- sankey(y, nodes = nodes, flows = flows, debug = FALSE, edge_n = 32,
            edge_f = 0.4, node_s = "prop")

# Another example
r <- new_networkModel() %>%
    set_topo("infusion -> plasma -> body -> plasma") %>%
    set_steady(c("infusion", "body"))
r <- topo(r)
ggtopo(r, edge = "fan")
sankey(r, debug = TRUE, edge_f = 0.2)
```

`scaled_beta_p` *Define a beta prior (on [0;scale])*

Description

If a random variable X follows a scaled beta distribution with parameters (alpha, beta, scale), then X/scale follows a beta distribution with parameters (alpha, beta).

Usage

```
scaled_beta_p(alpha, beta, scale = 1)
```

Arguments

alpha	Alpha parameter of the unscaled beta distribution.
beta	Beta parameter of the unscaled beta distribution.
scale	The upper boundary of the prior.

Value

A list defining the prior.

Examples

```
scaled_beta_p(0.8, 20, scale = 10)
```

---

select.mcmc.list *Select parameters based on their names*

Description

Select parameters based on their names

Usage

```
## S3 method for class 'mcmc.list'
select(.data, ...)
```

Arguments

.data	A coda::mcmc.list object.
...	Strings used to select variables using pattern matching with grepl.

Value

An mcmc.list object, with the same extra class(es) as .data (if any).

`set_half_life` *Set the half-life for radioactive tracers*

Description

Indicating a non-zero value for half-life will add a decay to the marked portion of the tracer element. The decay constant is calculated from the half-life value as:

Usage

```
set_half_life(nm, hl, quiet = FALSE)
```

Arguments

`nm`	A `networkModel` object.
`hl`	Half-life value, in the same time unit as the observations are (or will be) given. Setting half-life to zero is equivalent to using a stable isotope (no decay used in the model).
`quiet`	Boolean for verbosity.

Details

lambda_decay = log(2) / half_life

Note that for correct calculations the half-life value should be given in the same time unit (e.g. hour, day) that the time unit used for observations.

Value

A `networkModel` object.

Examples

```
library(magrittr)
x <- new_networkModel() %>%
    set_topo("32P -> root -> leaf") %>%
    set_half_life(hl = 14.268)
x
```

`set_init` *Set initial conditions in a network model*

Description

Set initial conditions in a network model

Usage

```
set_init(nm, data, comp, size, prop, group_by = NULL)
```

Arguments

`nm`	A `networkModel` object (e.g. output from `new_networkModel`)
`data`	A tibble containing the initial conditions
`comp`	String, name of the `data` column with the compartment names
`size`	String, name of the `data` column with the compartment sizes
`prop`	String, name of the `data` column with the compartment proportions of marked tracer
`group_by`	Optional vector of string giving the names of the columns to use for grouping the data into replicates

Value

A `networkModel` object.

Examples

```
# Using the topology from the Trinidad case study
m <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi, FBOM", "epi -> petro, pseph",
           "FBOM -> tricor", "petro, tricor -> arg")

# Taking initial condtions from the 'lalaja' dataset at t=0
inits <- lalaja[lalaja[["time.days"]] == 0, ]
inits
m <- set_init(m, inits, comp = "compartment", size = "mgN.per.m2",
              prop = "prop15N", group_by = "transect")
m
```

set_obs *Set observations in a network model*

Description

Set observations in a network model

Usage

```
set_obs(nm, data, comp, size, prop, time, group_by)
```

Arguments

nm	A networkModel object (e.g. output from new_networkModel)
data	A tibble containing the observations. If NULL, remove observations from the model.
comp	String, name of the data column with the compartment names
size	String, name of the data column with the compartment sizes
prop	String, name of the data column with the compartment proportions of heavy tracer
time	String, name of the data column with the sampling times
group_by	Optional vector of string giving the names of the columns to use for grouping the data into replicates

Value

A networkModel object.

Examples

```
# Using the topology from the Trinidad case study
m <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi, FBOM", "epi -> petro, pseph",
           "FBOM -> tricor", "petro, tricor -> arg")

# Taking initial condtions from the 'lalaja' dataset at t=0
inits <- lalaja[lalaja[["time.days"]] == 0, ]
inits
m <- set_init(m, inits, comp = "compartment", size = "mgN.per.m2",
              prop = "prop15N", group_by = "transect")
m

# Taking observations from 'lalaja'
m <- set_obs(m, lalaja[lalaja[["time.days"]] > 0, ], time = "time.days")
m
plot(m)
```

set_params *Set the parameters in a network model*

Description

Set the parameters in a network model

Usage

```
set_params(nm, params, force = TRUE, quick = FALSE)
```

Arguments

nm	A networkModel object.
params	A named vector or a tibble with columns c("parameter", "value") containing the (global) parameter values.
force	Boolean, if FALSE will not overwrite already set parameters.
quick	Boolean, if TRUE take some shortcuts for faster parameter settings when called by another function. This should usually be left to the default (FALSE) by a regular package user.

Value

A networkModel object.

Examples

```
m <- aquarium_mod
p <- sample_params(m)
m2 <- set_params(m, p)
m2$parameters
```

set_prior *Set prior(s) for a network model*

Description

Set prior(s) for a network model

Usage

```
set_prior(x, prior, param = "", use_regexp = TRUE, quiet = FALSE)

set_priors(x, prior, param = "", use_regexp = TRUE, quiet = FALSE)
```

Arguments

x	A networkModel object.
prior	A prior built with e.g. uniform_p() or hcauchy_p(). Call available_priors() to see a table of implemented priors. Alternatively, if prior is a tibble, the function will try to use it to set parameter priors. The format of such an argument is the same as the format of the output of the getter function priors() (see examples). Note that if 'prior' is given as a tibble, all other arguments (except 'x') are disregarded.
param	String, target parameter or regexp to target several parameters. Default is the empty string "", which will match all parameters.
use_regexp	Boolean, if TRUE (the default) then param is used as a regular expression to match one or several parameter names.
quiet	Boolean, if FALSE print a message indicating which parameters had their prior modified.

Value

A networkModel object.

Examples

```
# Copy `aquarium_mod`
m <- aquarium_mod
priors(m)

# Modify the priors of `m`
m <- set_priors(m, exponential_p(0.5), "lambda")
priors(m)

# Re-apply priors from the original `aquarium_mod`
prev_priors <- priors(aquarium_mod)
prev_priors
m <- set_priors(m, prev_priors)
priors(m)
```

---

set_prop_family *Set the distribution family for observed proportions*

Description

Set the distribution family for observed proportions

Usage

```
set_prop_family(nm, family, quiet = FALSE)
```

Arguments

nm	A networkModel object (output from new_networkModel).
family	Allowed values are "gamma_cv", "beta_phi", "normal_cv", and "normal_sd".
quiet	Boolean, if FALSE print a message indicating which distribution family is used for proportions.

Value

A networkModel object.

Examples

```
library(magrittr)

m <- new_networkModel() %>%
  set_topo(links = "NH4, NO3 -> epi -> pseph, tricor")
m <- m %>% set_prop_family("beta_phi")
m
attr(m, "prop_family")
```

---

set_size_family *Set the distribution family for observed sizes*

Description

Set the distribution family for observed sizes

Usage

```
set_size_family(nm, family, by_compartment, quiet = FALSE, quiet_reset = FALSE)
```

Arguments

nm	A networkModel object (output from new_networkModel).
family	Allowed values are "normal_cv" and "normal_sd".
by_compartment	Boolean, if TRUE then zeta is compartment-specific.
quiet	Boolean, if FALSE print a message indicating which distribution family is used for proportions.
quiet_reset	Boolean, write a message when model parameters (and covariates and priors) are reset?

Value

A networkModel object.

Examples

```
library(magrittr)

m <- new_networkModel() %>%
  set_topo(links = "NH4, NO3 -> epi -> pseph, tricor")
m <- m %>% set_size_family("normal_sd")
m
attr(m, "size_family")

m <- m %>% set_size_family(by_compartment = TRUE)
attr(m, "size_zeta_per_compartment")
```

`set_split` *Flag some network compartments as being split compartments*

Description

This function automatically adds a default prior (uniform on [0,1]) for the active portion of split compartments.

Usage

```
set_split(nm, comps = NULL, which = NULL)
```

Arguments

`nm`	A `networkModel` object.
`comps`	Vector of strings, the names of the compartments to set split.
`which`	Vector of integers giving the nm rows to update. Default is to update all rows.

Value

A `networkModel` object.

Examples

```
library(magrittr)
x <- new_networkModel() %>%
   set_topo("NH4 -> algae -> daphnia") %>%
   set_split("algae")
topo(x)
```

set_steady *Flag some network compartments as being in a steady state*

Description

Flag some network compartments as being in a steady state

Usage

```
set_steady(nm, comps = NULL, which = NULL)
```

Arguments

nm	A networkModel object.
comps	Vector of strings, names of the compartments to set steady.
which	Vector of integers giving the nm rows to update. Default is to update all rows.

Value

A networkModel object.

Examples

```
library(magrittr)
x <- new_networkModel() %>%
   set_topo("NH4 -> algae -> daphnia") %>%
   set_steady("NH4")
topo(x)
```

set_topo *Set the topology in a network model.*

Description

Set the topology in a network model.

Usage

```
set_topo(nm, ..., from = NULL, to = NULL)
```

Arguments

nm	A networkModel object (output from new_networkModel).
...	One or more strings describing the links defining the network topology. Optionally, links can be given as a data frame. See the examples for more details about acceptable input formats.
from	Optional, string containing the column name for sources if links are provided as a data frame.
to	Optional, string containing the column name for destinations if links are provided as a data frame.

Value

A networkModel object.

Examples

```
# A single string can describe several links in one go.
m <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi -> pseph, tricor")
m
topo(m)

# Several strings can be given as distinct arguments.
m2 <- new_networkModel() %>%
  set_topo("NH4, NO3 -> epi -> pseph, tricor",
           "NH4 -> FBOM, CBOM", "CBOM <- NO3")
m2
topo(m2)

# Multiple strings can be also be combined into a single argument with `c()`.
links <- c("NH4, NO3 -> epi -> pseph, tricor", "NH4 -> FBOM, CBOM",
           "CBOM <- NO3")
m3 <- new_networkModel() %>%
  set_topo(links)
m3
topo(m3)

# A data frame can be used to specify the links.
links <- data.frame(source = c("NH4", "NO3", "epi"),
                    consumer = c("epi", "epi", "petro"))
links
m4 <- new_networkModel() %>%
  set_topo(links, from = "source", to = "consumer")
m4
m4$topology[[1]]
```

size_family *Return the distribution family for observed sizes*

Description

Return the distribution family for observed sizes

Usage

```
size_family(nm, quiet = FALSE)
```

Arguments

nm	A networkModel object.
quiet	Boolean for being quiet about explaining the role of zeta (default is FALSE).

Value

A character string describing the distribution family used to model observed sizes.

Examples

```
size_family(aquarium_mod)
size_family(trini_mod)
```

stanfit_to_named_mcmclist *Convert a Stanfit object to a nicely named mcmc.list object*

Description

When running run_mcmc with stanfit = FALSE (typically for debugging purposes), the parameters in the returned stanfit object are named using a base label and an indexing system. This function provides a way to convert this stanfit object into a more conventional mcmc.list object where parameters are named according to their role in the original network model used when running run_mcmc.

Usage

```
stanfit_to_named_mcmclist(stanfit)
```

Arguments

stanfit	A stanfit object returned by rstan::sampling.

Value

An `mcmc.list` object. It also has the original stanfit object stored as an attribute `"stanfit"`.

---

tidy_data *Extract data from a networkModel object into a tidy tibble.*

Description

Extract data from a networkModel object into a tidy tibble.

Usage

```
tidy_data(x)
```

Arguments

x	A networkModel object.

Value

A tibble (note: row ordering is not the same as in the input).

Examples

```
tidy_data(aquarium_mod)
tidy_data(trini_mod)
```

---

tidy_dpp *Prepare tidy data and posterior predictions*

Description

This function prepares both tidy data from a model and tidy posterior predictions from a model fit. Having those two tibbles prepared at the same time allows to merge them to ensure that observed data, predicted data and original variables other than observations are all in sync when using y and y_rep objects for bayesplot functions.

Usage

```
tidy_dpp(model, fit, draw = NULL, cores = NULL)
```

Arguments

model	A networkModel object.
fit	A networkModelStanfit object.
draw	Integer, number of draws to sample from the posterior.
cores	Number of cores to use for parallel calculations. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).

Value

A list with y, y_rep and vars.

---

tidy_flows *Build a tidy table with the flows for each iteration*

Description

If neither n_per_chain and n are provided, all iterations are used.

Usage

```
tidy_flows(
  nm,
  mcmc,
  n_per_chain = NULL,
  n = NULL,
  n_grid = 64,
  steady_state = FALSE,
  dt = NULL,
  grid_size = NULL,
  at = NULL,
  end = NULL,
  use_cache = TRUE,
  cores = NULL
)
```

Arguments

nm	A networkModel object.
mcmc	The corresponding output from run_mcmc.
n_per_chain	Integer, number of iterations randomly drawn per chain. Note that iterations are in sync across chains (in practice, random iterations are chosen, and then parameter values extracted for those same iterations from all chains).
n	Integer, number of iterations randomly drawn from mcmc. Note that iterations are *not* drawn in sync across chains in this case (use n_per_chain if you need to have the same iterations taken across all chains).

n_grid	Size of the time grid used to calculate trajectories
steady_state	Boolean (default: FALSE). If TRUE, then steady state compartment sizes are calculated for each iteration and steady state flows are calculated from those compartment sizes.
dt, grid_size	Time step size or grid points, respectively.
at	Timepoints at which the predictions should be returned.
end	Final timepoint used in the projections.
use_cache	Boolean, use cache for faster calculations?
cores	Number of cores to use for parallel calculations. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).

Details

Warning: This function is still maturing and its interface and output might change in the future.

Value

A tidy table containing the mcmc iterations (chain, iteration, parameters), the grouping variables from the network model and the flows. The returned flow values are the average flow per unit of time over the trajectory calculations (or steady state flows if steady_state is TRUE).

Examples

```
tf <- tidy_flows(aquarium_mod, aquarium_run, n_per_chain = 25, cores = 2)
tf
tfmcmc <- as.mcmc.list(tf)
plot(tfmcmc)
```

---

tidy_mcmc *Extract a tidy output from an mcmc.list*

Description

Extract a tidy output from an mcmc.list

Usage

```
tidy_mcmc(x, spread = FALSE, include_constant = TRUE)
```

Arguments

x	An mcmc.list object
spread	Boolean, spread the parameters into separate columns?
include_constant	Boolean, include constant parameters as proper parameter traces?

Value

A tidy table containing one iteration per row

Examples

```
fit <- lapply(1:4, function(i) {
  z <- matrix(rnorm(200), ncol = 2)
  colnames(z) <- c("alpha", "beta")
  coda::as.mcmc(z)
})
fit <- coda::as.mcmc.list(fit)
tidy_mcmc(fit)
tidy_mcmc(fit, spread = TRUE)
```

---

tidy_posterior_predict

Draw from the posterior predictive distribution of the model outcome

---

Description

Draw from the posterior predictive distribution of the model outcome

Usage

```
tidy_posterior_predict(object, newdata, draw = NULL, cores = NULL, ...)
```

Arguments

object	A networkModelStanfit object.
newdata	The original model used to fit the networkStanfit object.
draw	Integer, number of draws to sample from the posterior. Default is 100.
cores	Number of cores to use for parallel calculations. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).
...	Not used for now.

Value

A tidy table.

tidy_steady_states *Build a tidy table with the calculated steady states for each iteration*

Description

If neither n_per_chain and n are provided, all iterations are used.

Usage

```
tidy_steady_states(nm, mcmc, n_per_chain = NULL, n = NULL)
```

Arguments

nm	A networkModel object.
mcmc	The corresponding output from run_mcmc.
n_per_chain	Integer, number of iterations randomly drawn per chain. Note that iterations are in sync across chains (in practice, random iterations are chosen, and then parameter values extracted for those same iterations from all chains).
n	Integer, number of iterations randomly drawn from mcmc. Note that iterations are *not* drawn in sync across chains in this case (use n_per_chain if you need to have the same iterations taken across all chains).

Value

A tidy table containing the mcmc iterations (chain, iteration, parameters), the grouping variables from the network model and the steady state sizes.

tidy_trajectories *Build a tidy table with the trajectories for each iteration*

Description

If neither n_per_chain and n are provided, all iterations are used.

Usage

```
tidy_trajectories(
  nm,
  mcmc,
  n_per_chain = NULL,
  n = NULL,
  n_grid = 64,
  dt = NULL,
  grid_size = NULL,
  at = NULL,
```

```
  end = NULL,
  use_cache = TRUE,
  cores = NULL
)
```

Arguments

nm	A networkModel object.
mcmc	The corresponding output from run_mcmc.
n_per_chain	Integer, number of iterations randomly drawn per chain. Note that iterations are in sync across chains (in practice, random iterations are chosen, and then parameter values extracted for those same iterations from all chains).
n	Integer, number of iterations randomly drawn from mcmc. Note that iterations are *not* drawn in sync across chains in this case (use n_per_chain if you need to have the same iterations taken across all chains).
n_grid	Size of the time grid used to calculate trajectories
dt, grid_size	Time step size or grid points, respectively.
at	Timepoints at which the predictions should be returned.
end	Final timepoint used in the projections.
use_cache	Boolean, use cache for faster calculations?
cores	Number of cores to use for parallel calculations. Default is NULL, which means to use the value stored in options()[["mc.cores"]] (or 1 if this value is not set).

Details

Warning: This function is still maturing and its interface and output might change in the future.

Value

A tidy table containing the mcmc iterations (chain, iteration, parameters), the grouping variables from the network model and the trajectories.

Examples

```
tt <- tidy_trajectories(aquarium_mod, aquarium_run, n = 10, cores = 2)
tt
```

topo *Return the list of topologies, or a unique topology if all identical*

Description

Return the list of topologies, or a unique topology if all identical

Usage

```
topo(nm, simplify = TRUE)
```

Arguments

nm	A networkModel object.
simplify	Boolean, return only a unique topology if all topologies are identical or if there is only one? Default is TRUE.

Value

A list of the networkModel topologies or, if all topologies are identical (or if there is only one) and simplify is TRUE, a single topology (not wrapped into a single-element list).

Examples

```
aquarium_mod
topo(aquarium_mod)

trini_mod
topo(trini_mod)
```

traceplot *Plot mcmc.list objects*

Description

Plot mcmc.list objects

Usage

```
traceplot(x, ...)
```

Arguments

x	A coda::mcmc.list object.
...	Passed to plot_traces.

Value

Called for side effect (plotting).

`trini_mod` *Network model for nitrogen fluxes in Trinidadian streams (Collins et al. 2016)*

Description

This model is used in the package case study about Trinidadian streams and is based on an original dataset taken from Collins et al. (2016).

Usage

```
trini_mod
```

Format

An object of class `networkModel` (inherits from `tbl_df`, `tbl`, `data.frame`) with 6 rows and 6 columns.

Details

The model is complete, with topology, initial conditions, observations, covariates and priors.

It is ready for an MCMC run as shown in the example. Note that it might be a good idea to relax the priors for uptake rates from seston to Leptonema (e.g. using hcauchy_p(10)), seston being a compartment that is flowing with the stream water and that can be replenished from upstream.

Source

This network model contains data from the original article: Collins, Sarah M., Steven A. Thomas, Thomas Heatherly, Keeley L. MacNeill, Antoine O.H.C. Leduc, Andrés López-Sepulcre, Bradley A. Lamphere, et al. 2016. "Fish Introductions and Light Modulate Food Web Fluxes in Tropical Streams: A Whole-Ecosystem Experimental Approach." Ecology, <doi:10.1002/ecy.1530>.

This dataset was also used in the paper: López-Sepulcre, Andrés, Matthieu Bruneaux, Sarah M. Collins, Rana El-Sabaawi, Alexander S. Flecker, and Steven A. Thomas. 2020. "A New Method to Reconstruct Quantitative Food Webs and Nutrient Flows from Isotope Tracer Addition Experiments." The American Naturalist 195 (6): 964–85. <doi:10.1086/708546>.

Examples

```
trini_mod
ggtopo(trini_mod)

## Not run:
# Warning: the run below can take quite a long time!
# (about 15 min with 4 cores at 3.3 Ghz).
```

```
run <- run_mcmc(trini_mod, iter = 500, chains = 4, cores = 4)

## End(Not run)
```

`type_sum.prior` *Function used for displaying* `prior` *object in tibbles*

Description

Function used for displaying `prior` object in tibbles

Usage

```
## S3 method for class 'prior'
type_sum(x)
```

Arguments

`x`	An object of class `prior`.

Value

Input formatted with `format(x)`.

`uniform_p` *Define a uniform prior*

Description

Define a uniform prior

Usage

```
uniform_p(min, max)
```

Arguments

`min, max`	Minimum and maximum boundaries for the uniform prior.

Value

A list defining the prior.

Examples

```
uniform_p(min = 0, max= 1)
```

`[.networkModelStanfit` *Subset method for* `networkModelStanfit` *objects*

Description

Subset method for `networkModelStanfit` objects

Usage

```
## S3 method for class 'networkModelStanfit'
x[i, j, drop = TRUE]
```

Arguments

`x`	A `networkModelStanfit` object.
`i`	A vector of iteration indices.
`j`	A vector of parameter names or indices.
`drop`	Boolean.

Value

A `networkModelStanfit` object.

Index