Package 'cvasi'

April 11, 2026

Type Package

Title Calibration, Validation, and Simulation of TKTD Models

Version 1.5.1

Description Eases the use of ecotoxicological effect models. Can simulate common toxicokinetic-toxicodynamic (TK/TD) models such as General Unified Threshold models of Survival (GUTS) and Lemna. It can derive effects and effect profiles (EPx) from scenarios. It supports the use of 'tidyr' workflows employing the pipe symbol. Time-consuming tasks can be parallelized.

URL https://github.com/cvasi-tktd/cvasi

BugReports https://github.com/cvasi-tktd/cvasi/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports cli, rlang, stringr, dplyr, tibble, purrr, furrr, tidyr, magrittr, utils, stats, methods, grid, gridExtra, ggplot2, GGally, deSolve, lubridate, units, lifecycle

RoxygenNote 7.3.3

Config/testthat/edition 3

Collate 'batch.R' 'cache.R' 'class-CalibrationSet.R' 'class-ExposureSeries.R' 'class-EffectScenario.R' 'sequence.R' 'calibrate.R' 'class-ParameterSet.R' 'class-Transferable.R' 'data.R' 'dose_response.R' 'effect.R' 'epx.R' 'explore_space.R' 'fx.R' 'solver.R' 'man-lemna.R' 'model-lemna_setac.R' 'model-magma.R' 'model-lemna_schmitt.R' 'fit_growth.R' 'fit_tktd.R' 'get.R' 'get_param.R' 'get_tag.R' 'get_times.R' 'globals.R' 'has.R' 'import_morse.R' 'import_toxswa.R' 'is.R' 'lik_profile.R' 'log.R' 'man-deb.R' 'man-macrophytes.R' 'model-algae.R' 'model-deb_abj.R' 'model-debtox.R' 'model-deb_daphnia.R' 'model-guts.R' 'model-guts_red.R' 'numerics.R' 'package.R' 'pll.R' 'plot.R' 'plotting.r' 'pull.R' 'set.R' 'set_bounds.R' 'set_exposure.R' 'set_forcings.R'

'set_init.R' 'set_param.R' 'set_times.R' 'set_transfer.R'
'set_window.R' 'show.R' 'simulate.R' 'survival.R' 'tox_data.R'
'utils-pipe.R'

Suggests future, knitr, lemna, rmarkdown, roxyglobals, testthat, withr

Depends R (>= 3.5.0)

VignetteBuilder knitr

Config/roxyglobals/filename globals.R

Config/roxyglobals/unique FALSE

NeedsCompilation yes

Author Nils Kehrein [aut, cre],
Dirk Nickisch [aut],
Peter Vermeiren [aut],
Torben Wittwer [ctb],
Johannes Witt [ctb],
Andre Gergs [ctb]

Maintainer Nils Kehrein <nils.kehrein@gmail.com>

Repository CRAN

Date/Publication 2026-04-11 09:20:02 UTC

Contents

`Algae-models` *Algae models*

Description

Overview of supported *Algae* models

Details

- `Algae_Weber()` by Weber *et al.* (2012)
- `Algae_TKTD()` based on Weber *et al.* (2012), but with scaled damage
- `Algae_Simple()` simplified growth model without additional forcing variables

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is

reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

References

Weber D, Schaefer D, Dorgerloh M, Bruns E, Goerlitz G, Hammel K, Preuss TG and Ratte HT, 2012. Combination of a higher-tier flow-through system and population modeling to assess the effects of time-variable exposure of isoproturon on the green algae Desmodesmus subspictatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31(4), 899-908. doi:10.1002/etc.1765

EFSA Panel on Plant Protection Products and their Residues, 2018. Scientific opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms. EFSA journal 16:5377 doi:10.2903/j.efsa.2018.5377

See Also

Lemna-models, Transferable

Other algae models: `Algae_Simple()`, `Algae_TKTD()`, `Algae_Weber()`

Other models: `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`

`Algae_Simple` *Simple algae model without environment (Rendal et al. 2023)*

Description

The model is a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for algae. It follows the concept of a simplified algae model described in Rendal et al. (2023). The model simulates the development of algal biomass. The growth of the algae population is simulated on the basis of growth rates, which are, in contrast to the Weber model, independent on environmental conditions which are usually optimal in laboratory effect studies. The toxicodynamic sub-model describes the effects of growth-inhibiting substances through a corresponding reduction in the photosynthesis rate on the basis of either external or internal concentrations (depending on user choice of *scaled* parameter setting).

Usage

```
Algae_Simple()
```

Value

an S4 object of type AlgaeSimple

State variables

The model has two state variables:

- A, Biomass (µg fresh wt/mL, cells/mL *10^4)
- Dw, scaled internal damage, only takes effect if parameter scaled = 1

Model parameters

- Growth model
  - mu_max, Maximum growth rate (d-1)
- Concentration response (Toxicodynamics)
  - EC_50, Effect concentration of 50% inhibition of growth rate (µg L-1)
  - b, slope of concentration effect curve at EC_50 (-)
  - dose_response, shape of the dose response curve (0 = logit, 1 = probit)
- External concentration (Toxicokinetics)
  - scaled, 1 = scaled internal damage, 0 = no internal damage / 1 = yes (-)
  - kD, dominant rate constant of toxicant in aquatic environments (d-1), only takes effect if parameter scaled = 1

Forcings

Simplified model without additional forcings for e.g. irradiation or temperature as implemented in Algae_Weber. A constant growth over time is assumed. In case that growth is time dependent, a forcing variable (f_growth) can be set. Forcing time-series are represented by data.frame objects consisting of two columns. The first for time and the second for a scaling factor of mu_max. The input format for all forcings is a list of the data frames. If f_growth is not set, a default scaling factor of 1 is used.

Parameter boundaries

Upper and lower parameter boundaries are set by default for each parameter. This, to avoid extreme values during calibration (particularly likelihood profiling)

Simulation output

Simulation results will contain the state variables biomass (A) and scaled damage concentration (Dw).

It is possible to amend the output of simulate() with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument nout of simulate(). As an example, set nout=2 to enable reporting of external concentration (Cw) and growth scaling factor (f_growth). Set nout=0 to disable additional outputs (default).

The available output levels are as follows:

- `nout >= 1`: `Cw` external concentration (µg L-1)
- `nout >= 2`: `f_growth` growth scaling factor (-)
- `nout >= 3`: `dA`, biomass derivative (µg)
- `nout >= 4`: `dDw`, damage concentration derivative (µg L-1)

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- `hmax = 0.01` Maximum step length in time suitable for most simulations.

References

Rendal C, Witt J, Preuss TG, Ashauer R, 2023: A Framework for Algae Modeling in Regulatory Risk Assessment. Environ. Toxicol. Chem. 42(8), pp. 1823-1838. doi:10.1002/etc.5649

Weber D, Schaefer D, Dorgerloh M, Bruns E, Goerlitz G, Hammel K, Preuss TG and Ratte HT, 2012: Combination of a higher-tier flow-through system and population modeling to assess the effects of time-variable exposure of isoproturon on the green algae Desmodesmus subspictatus and Pseudokirchneriella subcapitata. Environ. Toxicol. Chem. 31(4), pp. 899-908. doi:10.1002/etc.1765

See Also

Scenarios, Transferable

Other algae models: `Algae-models`, `Algae_TKTD()`, `Algae_Weber()`

`Algae_TKTD` *Algae model variant (Weber et al. 2012) with scaled damage*

Description

The model is a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for algae. The model simulates the development of algal biomass under laboratory and environmental conditions. The growth of the algae population is simulated on the basis of growth rates, which are dependent on environmental conditions (radiation, temperature and phosphorus). The model is a variant of the `Algae_Weber()` model (Weber 2012) as cited in EFSA TKTD opinion (2018). This Algae model, `Algae_TKTD()`, provides an additional possibility (probit) to simulate the dose-response curve and considers a scaled internal damage instead of the external concentration.

Usage

```
Algae_TKTD()
```

Value

an S4 object of type AlgaeTKTD

State variables

The model has four state variables:

- A, Biomass (µg fresh wt/mL, cells/mL *10^4)
- Q, Mass of phosphorous internal (µg P/µg fresh wt)
- P, Mass of phosphorous external (µg P/L)
- Dw, Damage concentration (µg/L)

Model parameters

- Growth model
  - mu_max, Maximum growth rate (d-1)
  - Q_min, Minimum intracellular P (µg P/µg fresh wt)
  - Q_max, Maximum intracellular P (µg P/µg fresh wt)
  - v_max, Maximum P-uptake rate at non-limited growth (µg P/µg fresh wt/d)
  - k_s, Half-saturation constant for extracellular P (mg P/L)
  - m_max, Natural mortality rate (1/d)
  - I_opt, Optimum light intensity for growth (µE/m²/s)
  - T_opt, Optimum temperature for growth (°C)
  - T_max, Maximum temperature for growth (°C)
  - T_min, Minimum temperature for growth (°C)
  - D, Dilution rate (1/d)
  - R_0, Influx concentration of P (mg P/L)
- Concentration response (Toxicodynamics)
  - EC_50, Effect concentration of 50% inhibition of growth rate (µg L-1)
  - b, slope of concentration effect curve at EC_50 (-)
  - dose_resp, shape of the dose response curve (0 = logit, 1 = probit)
- External concentration (Toxicokinetics)
  - kD, dominant rate constant (d-1)

Forcings

The Weber model variant requires two environmental properties as time-series input:

- T_act, temperature (°C), and
- I, irradiance (uE/m²/s).

The following constant default values are used for these properties:

- T_act = 23 °C
- I = 100 uE/m²/s

Forcings time-series are represented by data.frame objects consisting of two columns. The first for time and the second for the environmental factor in question.

Entries of the data.frame need to be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant environmental conditions. See scenarios for more details.

Simulation output

Simulation results will contain the state variables Biomass (A), mass of internal phosphorous (Q), mass of external phosphorous (P) and the damage concentration (Dw).

It is possible to amend the output of simulate() with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument nout of simulate(). As an example, set nout=2 to enable reporting of model derivatives dA and dQ. Set nout=0 to disable additional outputs (default).

The available output levels are as follows:

- nout >= 1: C, external concentration (µg/L)
- nout >= 2: f(T), temperature dependence (-)
- nout >= 3: f(I), light dependence (-)
- nout >= 4: f(Q), nutrient dependence (-)
- nout >= 5: f(Q, P), uptake flow reduction (-)
- nout >= 6: f(C), effect of chemical stressor (-)
- nout >= 7: dA, biomass derivative (µg)
- nout >= 8: dQ, internal phosphorous derivative (mg P/µg fresh wt)
- nout >= 9: dP, external phosphorous derivative (mg P L-1)
- nout >= 10: dDw, damage concentration derivative (µg L-1)

Solver settings

The arguments to ODE solver deSolve::ode() control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. simulate() on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- hmax = 0.1 Maximum step length in time suitable for most simulations.

Model history and changes

- cvasi v1.5.0
  - Support for simulating flow-through conditions by introducing new parameters `D` and `R_0` and adapting the ODEs according to the Algae_Weber model.
  - ODE of external phosphorous concentration `P` corrected, which contained an erroneous growth term before.
  - Response functions added to optional simulation outputs, order of output levels modified.

References

Weber D, Schaefer D, Dorgerloh M, Bruns E, Goerlitz G, Hammel K, Preuss TG and Ratte HT, 2012. Combination of a higher-tier flow-through system and population modeling to assess the effects of time-variable exposure of isoproturon on the green algae Desmodesmus subspictatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31(4), 899-908. doi:10.1002/etc.1765

See Also

Scenarios, Transferable

Other algae models: `Algae-models`, `Algae_Simple()`, `Algae_Weber()`

`Algae_Weber` *Algae model,* SAM-X *(Weber et al. 2012)*

Description

The model is a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for algae. The model simulates the development of algal biomass under laboratory and environmental conditions and was developed by Weber et al. (2012) as cited in EFSA TKTD opinion (2018). The growth of the algae population is simulated on the basis of growth rates, which are dependent on environmental conditions (radiation, temperature, and phosphorus). The toxicodynamic sub-model describes the effects of growth-inhibiting substances through a corresponding reduction in the photosynthesis rate on the basis of internal concentrations.

Usage

```
Algae_Weber()

SamX()
```

Details

Deviating from the equations described by Weber et al., this model implementation uses a user-defined time-series to represent (environmental) concentrations. Therefore, state-variable `C` and its differential equation was removed and model output `C` is identical to the exposure time-series. The implementation of Weber et al. (2012) was followed where units differ from EFSA (2018).

Value

an S4 object of type AlgaeWeber

Functions

- `SamX()`: Alias using original model name.

State variables

The model has three state variables:

- `A`, Biomass (µg fresh wt/mL, cells/mL *10^4)
- `Q`, Mass of phosphorous internal (mg P/L, or µg P/mL)
- `P`, Mass of phosphorous external (mg P/L, or µg P/mL)

The original model by Weber et al. contains an additional state variable `C` which models the external stressor concentration. However, the model implementation in this packages uses a user-defined time-series to represent environmental concentrations. Therefore, state variable `C` and accompanying parameters are not present here.

Model parameters

- Growth model
  - `mu_max`, Maximum growth rate (d-1)
  - `Q_min`, Minimum intracellular P (µg P/µg fresh wt)
  - `Q_max`, Maximum intracellular P (µg P/µg fresh wt)
  - `v_max`, Maximum P-uptake rate at non-limited growth (µg P/µg fresh wt/d)
  - `k_s`, Half-saturation constant for extracellular P (mg P/L)
  - `m_max`, Natural mortality rate (1/d)
  - `I_opt`, Optimum light intensity for growth (uE/m²/s)
  - `T_opt`, Optimum temperature for growth (°C)
  - `T_max`, Maximum temperature for growth (°C)
  - `T_min`, Minimum temperature for growth (°C)
  - `D`, Dilution rate (1/d)
  - `R_0`, Influx concentration of P (mg P/L)
- Concentration response (Toxicodynamics)
  - `EC_50`, Effect concentration of 50% inhibition of growth rate (µg/L)
  - `b`, slope of concentration effect curve at EC_50 (-)

Forcings

The Weber model requires two environmental properties as time-series input:

- `T_act`, temperature (°C), and
- `I`, irradiance (uE/m²/s).

The following constant default values are used for these properties:

- T_act = 23 °C
- I = 100 uE/m²/s

Forcings time-series are represented by data.frame objects consisting of two columns. The first for time and the second for the environmental factor in question.

Entries of the data.frame need to be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant environmental conditions. See scenarios for more details.

Simulation output

Simulation results will contain the state variables Biomass (A), mass of internal phosphorous (Q), mass of external phosphorous (P) and the external concentration (C).

It is possible to amend the output of simulate() with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument nout of simulate(). As an example, set nout=2 to enable reporting of external concentration and model derivative dA. Set nout=0 to disable additional outputs. The default is nout=1.

The available output levels are as follows:

- nout >= 1: C, external concentration (µg/L)
- nout >= 2: f(T), temperature dependence (-)
- nout >= 3: f(I), light dependence (-)
- nout >= 4: f(Q), nutrient dependence (-)
- nout >= 5: f(Q, P), uptake flow reduction (-)
- nout >= 6: f(C), effect of chemical stressor (-)
- nout >= 7: dA, biomass derivative (µg)
- nout >= 8: dQ, internal phosphorous derivative (mg P/µg fresh wt)
- nout >= 9: dP, external phosphorous derivative (mg P L-1)

Solver settings

The arguments to ODE solver deSolve::ode() control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. simulate() on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- hmax = 0.1 Maximum step length in time suitable for most simulations.

Parameter boundaries

Default values for parameter boundaries are set for all parameters by expert judgement, for calibration purposes. Values can be access from the object, and defaults overwritten.

Model history and changes

- cvasi v1.5.0
  - Unused state variable `C` and parameter `k` removed from documentation and code. External concentration `C` added to simulation output by means of optional output level `nout=1`.

References

Weber D, Schaefer D, Dorgerloh M, Bruns E, Goerlitz G, Hammel K, Preuss TG and Ratte HT, 2012. Combination of a higher-tier flow-through system and population modeling to assess the effects of time-variable exposure of isoproturon on the green algae Desmodesmus subspictatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31(4), 899-908. doi:10.1002/etc.1765

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, Brock T, Duquesne S, Grilli S, Hernandez-Jerez AF, Bennekou SH,Klein M, Kuhl T, Laskowski R, Machera K, Pelkonen O, Pieper S, Smith RH, Stemmer M, Sundh I, Tiktak A,Topping CJ, Wolterink G, Cedergreen N, Charles S, Focks A, Reed M, Arena M, Ippolito A, Byers H andTeodorovic I, 2018. Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD)effect models for regulatory risk assessment of pesticides for aquatic organisms. EFSA Journal, 16(8), 5377. doi:10.2903/j.efsa.2018.5377

See Also

Scenarios, Transferable

Other algae models: `Algae-models`, `Algae_Simple()`, `Algae_TKTD()`

`americamysis` *A DEB abj scenario of Americamysis bahia*

Description

Species parameters were collected from the AddMyPet database entry on Americamysis bahia (Opossum shrimp). The exposure series consists of a constant exposure resulting in medium effects on length and reproduction.

Usage

```
americamysis
```

Format

An object of class `DebAbj` of length 1.

Source

https://www.bio.vu.nl/thb/deb/deblab/add_my_pet/entries_web/Americamysis_bahia/Americamysis_bahia_res.html

See Also

DEB_abj()

---

batch *Batch simulation of multiple exposure levels*

Description

[Experimental]

Usage

```
batch(
  scenario,
  exposure,
  id_col = "trial",
  format = c("long", "wide"),
  times_from = c("scenario", "exposure"),
  select = NULL
)
```

Arguments

scenario	a scenario object
exposure	a named list() or a data.frame with three columns
id_col	character, name of column in resulting ´data.frame‘ which contains a trial's name or ID
format	character, set to 'long' for long tabular format, or 'wide' for wide format
times_from	character, set to 'scenario' to use output times from scenario, or 'exposure' to take output times from each exposure series
select	optional character vector to select columns from the simulation output

Details

A convenience function to simulate a single base scenario with one or more exposure levels. The functions aims at reproducing the setup and result format of common effect studies.

Simulating a scenario is generally limited to assessing a single exposure series. However, laboratory experiments commonly examine the effects of multiple exposure levels on a biological system. A *batch simulation* approach involves running multiple simulations with varying exposure or *treatment* conditions. To illustrate: if the objective is to examine the impact of a chemical on cell growth, multiple scenarios need to be simulated to reproduce the cell growth dynamics under varying concentrations of the assessed chemical. Each simulation run will represent a specific exposure level, ranging from low to high concentrations of the chemical.

To simulate the conditions of such a laboratory experiment, the scenarios and exposure levels can either be created and simulated individually, or the batch() function can be used for ease of use.

Exposure series:

The set of exposure levels can be represented by one of the following types:

- A (named) list: Each element represents an exposure level or exposure series. An exposure level can be represented by a constant numeric, a `data.frame` with two columns, or an ExposureSeries object. The names of the list elements specify the study ID.
- Or alternatively, a `data.frame` with three columns: One column for time, one for the exposure level, and one character column to specify the study IDs.

Each exposure level will be simulated using the base scenario. If the exposure levels are provided as a named list, the names will also appear in the return value of `simulate()`. This behavior can be used, for example, to define unique study IDs for particular exposure levels.

Exposure IDs:

The list of exposure levels can be supplied as a named list. The names will be used as unique (study) IDs, so that the simulation results belonging to any exposure level can be identified in the output. If no IDs are defined by the user, generic IDs of the form `'trial{n}'` will be assigned, with `{n}` being replaced by consecutive integers starting at one.

If the batch is passed on to `simulate()`‘, the IDs will be contained in its return value, e.g. as a dedicated column (long format) or as part of the column names (wide format).

Output format:

The return value of `simulate()` is by default in long format, i.e. it will contain one row for each output time and exposure level. It is possible to pivot the tabular data to wide format, by setting the argument `format = 'wide'`.

In wide format, the output columns of each exposure level are pasted next to each other. If more than one column is pivoted per exposure level, then the exposure or study ID is added as a suffix to column names. If the output per exposure level contains only a single column (besides time and the exposure ID itself), then original column name is dropped and only exposure IDs are used. See the examples section for reference.

Select output columns:

Often, only a single output column is of interest in batch simulations, such as the number of surviving individuals. To ease the interpretation and handling of the output of batch simulations, the columns contained in the output of each simulated exposure level can be filtered. One or more columns can be selected. By default, no filtering of output columns is conducted.

As an example, to create an overview of survival probabilities (S) in the GUTS-RED-IT example scenario `minnow_it`:

```
minnow_it %>%
  batch(exposure=list(0, 5, 10), select="S", format="wide") %>%
  simulate()
```

Value

a simulation batch object

Examples

```
# Simulate a batch experiment with three constant exposure levels of
# 0.0, 2.0, and 5.0 µmol/L
simulate(batch(minnow_it, list(0, 2, 5)))

# Alternatively, in tidyr style syntax
trials_list1 <- list(0, 2, 5)
minnow_it %>%
  batch(trials_list1) %>%
  simulate()

# Assign unique IDs to each exposure level
trials_list2 <- list(Control=0, TrialA=2, TrialB=5)
minnow_it %>%
  batch(trials_list2) %>%
  simulate()

# Alternatively, define multiple exposure levels in a single data.frame
trials_table <- data.frame(time=c(0, 0, 0),
                           conc=c(0, 2, 5),
                           trial=c("Control", "TrialA", "TrialB"))
minnow_it %>%
  batch(trials_table) %>%
  simulate()

# Limit simulation output to column 'S' (survival probability)
minnow_it %>%
  batch(trials_list2, select="S") %>%
  simulate()

# Return data in wide-format, unique IDs will be used as column names
minnow_it %>%
  batch(trials_list2, select="S", format="wide") %>%
  simulate()
```

---

cache_controls *Cache control simulations*

Description

[Deprecated]

Usage

```
cache_controls(x, ...)
```

Arguments

x	parameter not used
...	parameters not used

Details

Handling of cached control simulations has been modified and is solely managed by package functions. `cache_controls()` is no longer needed and will raise an error if called.

Value

nothing

`calibrate` *Fit model parameters to experimental data*

Description

The function `calibrate()` performs the calibration (fitting) of model parameters to observed data. The data can originate from one or more experiments or trials. Experimental conditions, such as model parameters and exposure level, can differ between trials; fitting can be performed on all datasets at the same time.

Usage

```
## S4 method for signature 'EffectScenario'
calibrate(x, output, data, ...)

## S4 method for signature 'list'
calibrate(
  x,
  par,
  output,
  err_fun = "sse",
  log_scale = FALSE,
  verbose = TRUE,
  ode_method = NULL,
  ...
)
```

Arguments

`x`	either a single scenario. sequence, or a list of caliset objects to be fitted. If only a scenario or sequence is supplied, the additional argument `data` is required.
`output`	`character`, name of a single output column of `simulate()` to optimize on
`data`	a `data.frame` or return value of `tox_data()`; the scenarios's output is fitted to the (observed) data. See `tox_data()` for valid tabular data formats.
`...`	additional parameters passed on to `stats::optim()` and `simulate()`
`par`	named numeric vector with parameters to fit and their start values

err_fun	a character choosing one of the pre-defined error functions, or alternatively a function implementing a custom error function. Defaults to *Sum of squared errors* ("sse").
log_scale	logical, if TRUE then observed and predicted values are log-transformed before the error function is evaluated. Defaults to FALSE.
verbose	logical, if TRUE then debug outputs are displayed during optimization
ode_method	optional character to select an ODE solver for simulate()

Details

Fitting of model parameters can be performed in two ways:

1. A single scenario is fitted to a single dataset. The dataset must represent a time-series of an output variable of the model, e.g. observed biomass over time (effect data). The dataset can represent results of one or more experimental replicates under identical conditions.
2. One or more datasets of observed data are fitted each to a scenario which describes the experimental conditions during observation, such as exposure level and environmental properties. Each combination of dataset and scenario is represented by a calibration set. During fitting, all *calibration sets* are evaluated and a total error term is calculated over all observed and predicted values.

Observed data:

Experimental, or effect, data must be supplied as a data.frame in long format with at least two columns: the first column contains numeric timestamps and the remaining columns must contain the observed quantity. The observed quantitiy must match in unit and meaning with the model output defined by argument output.

As an example, the simulation result of Lemna_Schmitt model contains the output column *biomass* (BM), amongst others. To fit model parameters of said *Lemna_Schmitt* scenario based on observed biomass, the observed data must represent biomass as well. A minimal observed dataset could look like this:

```
observed <- data.frame(time=c(0,  7, 14, 21),
                       BM=c( 12, 23, 37, 56))
```

Error function:

The error function quantifies the deviations between simulated and observed data. The decision for an error function can have influence on the result of the fitting procedure. Two error functions are pre-defined by the package and can be selected by the user, but custom error functions can be used as well.

Available pre-defined error functions:

- "sse": Sum of squared errors

By default, the sum of squared errors is used as the error function which gets minimized during fitting. A custom error function must accept four vectorized arguments and return a numeric of length one, i.e. the total error value. The arguments to the error function will all have the same length and are defined as follows:

- First argument: all observed data points
- Second argument: all simulated data points

- Third argument: optional weights for each data point
- Fourth argument: a list of optional caliset tags

You can choose to ignore certain arguments, such as weights and tags, in your custom error function. An example of a custom error function which returns the sum of absolute errors looks as follow:

```
my_absolute_error <- function(observed, predicted, weights, tags) {
  sum(abs(observed - predicted))
}
```

As tags are optional, the fourth argument may be a list containing `NULL` values. The fourth argument can be used to pass additional information to the error function: For example, the tag may identify the study from where the data originates from and the error function could group and evaluate the data accordingly.

Value

A list of fitted parameters (as produced by `stats::optim()`) is returned.

Methods (by class)

- `calibrate(EffectScenario)`: Fit single scenario to a (tox_data) dataset

Examples

```
# Create an artificial data set of observed frond numbers.
# It assumes exponential growth with an effective growth rate of 0.38
trial <- data.frame(time=0:14,
                    fronds=12 * exp(0:14 * 0.38))
plot(trial)

# Create a Lemna scenario that represents unrestricted, exponential growth.
scenario <- Lemna_Schmitt() %>%
  set_param(c(k_phot_max=1, k_resp=0, EC50=1, b=1, P_up=1)) %>%
  set_init(c(BM=12)) %>%
  set_noexposure()

# Fit scenario parameter 'k_phot_max' to observed frond numbers:
fit <- calibrate(
  scenario,
  par="k_phot_max",
  data=trial,
  output="BM"
)

# The fitted value of 'k_phot_max' matches the effective growth rate which
# was used to create the artificial data set:
fit$par
```

`CalibrationSet` *Calibration set*

Description

A *calibration set* combines a scenario, observed data, and an optional weighting factor into one object. The *calibration set* is used to fit model parameters to observed data using `calibrate()`.

Usage

```
caliset(scenario, data, weight = 1.0, tag = NULL)
```

Arguments

`scenario`	a scenario describing conditions during the experiment
`data`	a `data.frame` with observed data in long format containing two columns: the 1st column with `numeric` time points and 2nd column with `numeric` data to fit to. Rows with observed `NA` values will be removed.
`weight`	optional `numeric` weight to be applied when calculating the error term for each data point. Default value is `1.0`, i.e. no weighting.
`tag`	optional value to identify the data, e.g. a study number

Details

A *calibration set* usually represents a single experiment or trial. Multiple experimental replicates can be combined into a single *set*, if model parameters are identical between trials. If model parameters were modified during a trial, e.g. a pump failure occurred or flow rates changed, this can be represented by using a *scenario sequence* instead of a basic scenario. Please refer to `sequence()` for details.

Weighting:

An optional weighting factor can be used to scale the error term of a whole *set* or of individual data points when fitting parameters using e.g. `calibrate()`.

The vector of weights must either be of length one or have the same length as the dataset. In the former case, the same weight will be applied to all values in the dataset. In the latter, individual weights are applied for each data point.

Value

`caliset()` returns a *calibration set* object

Examples

```
library(dplyr)

# Get observed biomass during control experiment by Schmitt et al. (2013)
observed <- schmitt2013 %>%
```

```
  filter(trial == "T0") %>%
  select(time, obs)

# Create a scenario that represents conditions during experiment
scenario <- metsulfuron %>%
  set_param(c(k_phot_fix=TRUE, k_resp=0, Emax=1)) %>%
  set_init(c(BM=12)) %>%
  set_noexposure() %>%
  set_bounds(list(k_phot_max=c(0, 0.5)))

# Create a calibration set
cs <- caliset(scenario, observed)

# Fit parameter 'k_phot_max' to observed biomass growth from experiment
calibrate(
  cs,
  par=c(k_phot_max=1),
  output="BM",
  method="Brent" # Brent is recommended for one-dimensional optimization
) -> fit
fit$par
```

`ci_from_hessian` *Parameter confidence intervals from fit*

Description

Calculates parameter confidence intervals for a fit returned by `calibrate()`. The fit must provide a valid *Hessian* matrix.

Usage

```
ci_from_hessian(fit, dof, level = 0.95)
```

Arguments

`fit`	return value from `calibrate()`
`dof`	integer, Degrees of Freedom, commonly the number of independent observations minus the number of fitted parameters
`level`	numeric, desired confidence level, i.e a value between zero and one. Defaults to 95% (`0.95`).

Value

name list of numeric vectors of length two, the elements representing the lower and upper confidence limit, respectively.

DEB-models *Dynamic Energy Budget (DEB) models*

Description

Supported models:

- DEB_abj
- DEBtox

See Also

Other DEB models: `DEB_abj()`, `DEBtox()`

Other models: `Algae-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`

DEBtox *DEBtox model*

Description

Creates a *DEBtox* scenario as described by Jager (2020). It represents a simplified *DEBtox* model based on *DEBkiss*. In the *BYOM* application [link], this model is referred to as *DEBtox 2019*, version 4.7. It supports an optional feature of the *ERA special* model variant, which can consider a reference *Lm* parameter to compare results of multiple datasets.

Usage

```
DEBtox()

DEB_Daphnia()
```

Details

State variables:

The following list describes the names and units of the model's state variables:

- `D`, scaled damage ([C])
- `L`, body length (mm)
- `R`, cumulative reproduction (-)
- `S`, survival probability (-)

State variables `D`, `L`, and `R` are initialized with zero. Variable `S` is initialized with one (`1.0`). See `set_init()` on how to set the initial state manually.

Parameters:

The following parameters are required:

- General
  - L0, body length at start (mm)
  - Lp, body length at puberty (mm)
  - Lm, maximum body length (mm)
  - rB, von Bertalanffy growth rate constant (1/d)
  - Rm, maximum reproduction rate (#/d)
  - f, scaled functional response (-)
  - hb, background hazard rate (d-1)
  - a, Weibull background hazard coefficient (-). Set to 1 to disable.
- Extra parameters
  - Lf, body length at half-saturation feeding (mm)
  - Lj, body length at which acceleration stops (mm)
  - Tlag, lag time for start development (d)
- TK/TD parameters
  - kd, dominant rate constant (d-1)
  - zb, effect threshold energy budget ([C])
  - bb, effect strength energy-budget effects (1/[C])
  - zs, effect threshold survival ([C])
  - bs, effect strength survival (1/([C] d))
- Other parameters (formerly globals in *BYOM*)
  - FBV, dry weight egg as fraction of structural body weight (-)
  - KRV, part. coeff. repro buffer and structure (kg/kg) (for losses with reproduction)
  - kap, approximation for kappa (for starvation response)
  - yP, product of yVA and yAV (for starvation response)
  - Lm_ref, optional reference max length for scaling rate constants (mm). Set to zero to disable the reference length. Disabled by default.
  - len, a switch to control body length dynamics: 1 organism can shrink, 2 organism cannot shrink. Default value is 1.
  - Tbp, optional brood-pouch delay (d). Set to NA or zero to disable. Default value is 0.
  - MoA, mode of action switches (-). Default value is 0.
  - FB, feedback on damage dynamics switches (-). Default value is 0.

A reference Lm_ref is needed to properly compare different data sets, or when calibrating on more than one data set. If Lm differs, one would not want to have different rate constants at the same length.

Mode of Action:
Any combination of the following mode of actions (*MoA*) can be considered by the model:

- MoA = 1: assimilation/feeding
- MoA = 2: costs for maintenance
- MoA = 4: costs for growth and reproduction
- MoA = 8: costs for reproduction
- MoA = 16: hazard for reproduction

To activate more than one mode of action, simply add up the corresponding codes and set parameter `MoA` to the desired value. To disable all mode of actions, set parameter `MoA` to zero. See also `set_moa()`.
As an example, to consider effects on feeding and maintenance, set the mode of action to three (`3`):

```
DEBtox() %>% set_param(c(MoA=3))
```

Feedbacks:
Any combination of the following damage feedbacks can be considered by the model:

- `1`: surf:vol scaling uptake rate
- `2`: surf:vol scaling elimination rate
- `4`: growth dilution
- `8`: losses with reproduction

To activate more than one feedback, simply add up the corresponding codes. To disable all feedbacks, set the parameter to zero.

Effects:

The state variables *L* (body length), *R* (cumulative reproduction), and *S* (survival probability) are set as effect endpoints by default. All state variables are available as potential endpoints. The list of considered endpoints can be modified by using `set_endpoints()`.

To calculate effects, each *DEBtox* scenario is simulated twice: One simulation which considers exposure to a toxicant and one simulation without exposure, i.e. a control. See also `effect()`.

Simulation output:

The following intermediary model variables can be added to the model output on demand. Simply set the optional parameter `nout` to the required output level and pass it to `simulate()`.

- `nout` >= 1: `f`, actual scaled response
- `nout` >= 2: `fR`, actual f considering starvation
- `nout` >= 3: `kd`, actual kd
- `nout` >= 4: `s`, stress level
- `nout` >= 5: `h`, hazard rate
- `nout` >= 6: `sA`, stress factor on assimilation/feeding
- `nout` >= 7: `sM`, stress factor on maintenance
- `nout` >= 8: `sG`, stress factor on growth costs
- `nout` >= 9: `sR`, stress factor on reproduction costs
- `nout` >= 10: `sH`, stress factor on hazard to reproduction
- `nout` >= 11: `xu`, damage feedback factor for surf:vol scaling uptake rate
- `nout` >= 12: `xe`, damage feedback factor for surf:vol scaling elimination rate
- `nout` >= 13: `xG`, damage feedback factor for growth dilution
- `nout` >= 14: `xR`, damage feedback factor for losses with repro

Solver settings:

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all

deSolve settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- `method = 'ode45'` Selects the Dormand-Prince 4(5) method of the Runge-Kutta family, see `deSolve::rkMethod()` for details.

Model history and changes:

- cvasi v1.0.0
  - The `DEB_Daphnia()` model implemented BYOM's *DEBtox 2019* model version 4.5
- cvasi v1.2.0
  - The model equations were updated to conform with BYOM's *DEBtox 2019* version 4.7. This introduced a new model parameter `a`, the Weibull background hazard coefficient, and limited the maximum hazard rate to 99% per hour.
  - The scenario constructor was renamed to `DEBtox()`.
  - Additional intermediary model variables available as optional simulation output

Value

an S4 object of type DebTox

Functions

- `DEB_Daphnia()`: Deprecated model variant of `DEBtox()`

References

Jager T, 2020: Revisiting simplified DEBtox models for analysing ecotoxicity data. Ecol Model 416. doi:10.1016/j.ecolmodel.2019.108904

Romoli et al., 2024: Environmental risk assessment with energy budget models: a comparison between two models of different complexity. Environ Toxicol Chem 43(2):440-449. doi:10.1002/etc.5795

See Also

Other DEB models: `DEB-models`, `DEB_abj()`

DEB_abj *DEB_abj*

Description

Creates a *DEB abj* scenario. The *abj* model with type M acceleration is like model *std*, but acceleration occurs between birth and metamorphosis (V1-morph). Isomorphy is assumed before and after acceleration. Metamorphosis is before puberty and occurs at maturity `E_Hj`, which might or might not correspond with changes in morphology. The *abj* model is a one-parameter extension of model *std* (DEB Wiki).

Usage

```
DEB_abj()
```

Details

State variables:

The following list describes the default names and standard units of the model's state variables:

- `L`, structural length (cm)
- `E`, energy reserve (J)
- `H`, energy invested in maturity (J)
- `R`, reproduction buffer (J)
- `cV`, internal concentration (C)
- `Lmax`, maximum structural length (cm)

All state variables are initialized with zero. See `set_init()` on how to set the initial state.

Parameters:

The following model parameters are required:

- `p_M`, vol-spec somatic maintenance (J/d.cm^3)
- `v`, energy conductance (cm/d)
- `k_J`, maturity maint rate coefficient (1/d)
- `p_Am`, surface-area specific maximum assimilation rate (J/d.cm^2)
- `kap`, allocation fraction to soma (-)
- `E_G`, spec cost for structure (J/cm^3)
- `f`, scaled functional response (-)
- `E_Hj`, maturity at metamorphosis (J)
- `E_Hp`, maturity at puberty (J)
- `kap_R`, reproduction efficiency (-)
- `L_b`, structural length at birth (cm)
- `L_j`, structural length at metamorphosis (cm)
- `ke`, elimination rate constant (d-1)
- `c0`, no-effect concentration sub-lethal (C)

- cT, tolerance concentration (C)
- MoA, mode of action switch (-)

Mode of Actions:

Any combination of the following mode of actions (MoA) can be considered by the model:

- MoA = 1: effect on feeding
- MoA = 2: effect on maintenance costs
- MoA = 4: effect on overhead costs for making an egg
- MoA = 8: hazard during oogenesis
- MoA = 16: energy conductance

To activate more than one MoA, simply add up the corresponding codes. To disable all MoAs, set the parameter to zero. See also `set_mode_of_action()`.

Effects:

The state variables *L* (structural length) and *R* (reproduction buffer) are set as effect endpoints by default. All state variables are available as potential endpoints. The list of considered endpoints can be modified by using `set_endpoints()`.

To calculate effects, each *DEB* scenario is simulated twice: One simulation which considers exposure to a toxicant and one simulation without exposure, i.e. a control. See also `effect()`.

Value

an S4 object of type DebAbj

Simulation output

Simulation results will contain the state variables. It is possible to amend the output of `simulate()` with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument `nout` of `simulate()`. As an example, set `nout=2` to enable reporting of the acceleration factor (MV) and the mobilization flux (pC). Set `nout=0` to disable additional outputs (default).

The available output levels are as follows:

- nout >= 1: MV acceleration factor (-)
- nout >= 2: pC mobilization flux (J/d)
- nout >= 3: pA assimilation flux (J/d)
- nout >= 4: pJ energy invested in maturity flux (J/d)

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

See Also

Other DEB models: DEB-models, DEBtox()

Examples

```
# Create an abj scenario from scratch and simulate it
DEB_abj() %>%
  set_init(c(L=0.02,E=0.1,H=0.01)) %>%
  set_param(c(p_M=3000,v=0.02,k_J=0.6,p_Am=300,kap=0.9,E_G=4000,f=1,
              E_Hj=0.05,E_Hp=0.3,kap_R=0.9,ke=1,c0=0,cT=1,L_b=0.02,
              L_j=0.04,MoA=0)) %>%
  set_noexposure() %>%
  set_times(0:10) %>%
  simulate()

# Print information about sample scenario 'americamysis'
americamysis

# Simulate 'americamysis' scenario
americamysis %>% simulate()
```

---

diagnostics *Diagnostics of solvers*

Description

Prints several diagnostics of the simulation to the console, e.g. number of steps taken, the last step size, etc. The information is provided by deSolve::diagnostics().

Usage

```
diagnostics(obj, ...)

## Default S3 method:
diagnostics(obj, ...)

## S3 method for class 'cvasi_simulate'
diagnostics(obj, ...)
```

Arguments

obj	return value of a simulation
...	unused parameters

See Also

deSolve::diagnostics()

dmagna *A DEBtox scenario of Daphnia magna*

Description

Species and substance parameters were collected from test runs of the original DEBtox Daphnia model.

Usage

```
dmagna
```

Format

An object of class `DebTox` of length 1.

See Also

`DEBtox()`

dose_response *Calculate a dose response curve*

Description

Returns a `data.frame` with points on the dose response curve for the given effect scenario.

Usage

```
dose_response(
  scenario,
  range = c(1, 99),
  n = 20,
  strategy = c("exponential", "decadic", "vanilla"),
  verbose = FALSE,
  ...
)
```

Arguments

`scenario`	used for calculation
`range`	numeric vector specifying the required range of effect levels in percent (%), defaults to `c(1,99)`
`n`	minimum number of points on the dose response curve

strategy	controls how multiplication factors are chosen, vanilla uses a fixed set of multiplication factors, decadic and exponential have varying step lengths.
verbose	logical, set to TRUE for additional status messages
...	additional arguments passed on to effect()

Details

Derives a dose response curve from a scenario. The result will cover the requested range of effect levels. The tested multiplication factors can be chosen by different strategies, i.e. a vanilla approach using a fixed set of factors, or decadic and exponential approaches employing logarithmic and exponential factor scaling, respectively.

Value

data.frame with two columns, i.e. mf and effect

Examples

```
# basic dose response curve
minnow_sd %>% dose_response()

# modify the minimum number of points on the curve
minnow_sd %>% dose_response(n=10)

# select a subset of the effect range
minnow_sd %>% dose_response(range=c(10,20))

# use an alternative strategy for the selection of multiplication factors
minnow_sd %>% dose_response(strategy="decadic")

# provide additional output how multiplication factors were selected
minnow_sd %>% dose_response(verbose=TRUE)
```

---

effect *Effect level*

Description

Derives the effect level due to toxicant exposure in the supplied scenarios. Either relative to a control scenario or derived directly from model endpoints, depending on model type. For scenarios with moving exposure windows, the maximum effect is returned.

Usage

```
effect(x, ...)

## S4 method for signature 'EffectScenario'
effect(x, factor = 1, max_only = TRUE, ep_only = FALSE, marginal_effect, ...)
```

```
## S4 method for signature 'ScenarioSequence'
effect(x, ...)
```

Arguments

x	a scenario objects
...	additional parameters passed on to simulate()
factor	optional numeric value which scales the exposure time-series
max_only	logical, if TRUE only the maximum effect is returned, else results for all effect windows are reported
ep_only	logical, if TRUE only effect endpoints are returned as a vector
marginal_effect	numeric, if set, any effect smaller than this threshold will be reported as zero to exclude pseudo-effects originating from small numerical errors

Details

By default, only the maximum effect in all moving exposure windows will be returned. If argument max_only=FALSE is set, the returned table will be converted to long-format and will contain effect levels for each assessed exposure window.

Calculation:

Effects are calculated similarly to *relative errors*, i.e. the difference between control and treatment scenarios is divided by the absolute value of the control. Effects are usually in the interval [0,1], but values larger than one or smaller than zero can occur. As a special case, if the endpoint from the control scenario is zero, then the effect is either

- zero, if also the treatment is zero
- positive infinity, if the treatment is smaller than zero
- negative infinity, if the treatment is greater than zero

As an example, a control scenario achieves a biomass of 1.0 and the treatment scenario achieves a biomass of 0.9, the effect will be equal to 0.1 or 10%. However, effects can take on any real value. If, for example, the biomass of the previously mentioned treatment scenario drops below zero, then an effect larger than 1.0 will be calculated If, instead, the biomass in the treatment scenario is greater than in the control, then the effect will be negative.

Output formatting:
Start and end time of exposure windows can be disabled by setting ep_only=TRUE. Effect levels smaller than a certain threshold can be automatically set to zero (0.0) to avoid spurious effect levels introduced by numerical errors. Set marginal_effect to an adequate value less than 1%.

Computational efficiency:
Calculations can be sped up by providing a data.frame of pre-calculated control scenarios for each assessed time window. As control scenarios are by definition independent of any exposure multiplication factor, they can be reused for repeated calculations, e.g. to derive effect profiles or dose-response relationships.

Value

a `tibble`, by default containing scenarios, effect levels, and the exposure window where the maximum effect level occurred. The number of columns depends on the enabled effect endpoints and function arguments.

By default, the first column, named `scenarios`, contains the original scenario objects that were the basis of the calculation. For each effect endpoint, it will be followed by one column with the maximum effect level and two columns containing start and end time of the associated exposure window. If exposure windows are disabled, the columns will just contain the start and end time of the simulation. The effect level column will have the name of the effect endpoint, start and end time will additionally have the suffixes `.dat.start` and `.dat.end`, respectively.

Methods (by class)

- `effect(EffectScenario)`: Default for all generic scenarios
- `effect(ScenarioSequence)`: For scenario sequences

---

epx *Effect profiles (EPx values)*

Description

Derives one or more EPx/LPx values for the supplied effect scenarios, i.e. it calculates the multiplication factors of an exposure profile that cause x% of effect. Scenarios are processed in parallel, if possible.

Usage

```
epx(
  scenarios,
  level = c(10, 50),
  effect_tolerance = 0.001,
  factor_cutoff = NA,
  min_factor = 1e-30,
  max_factor = 1e+30,
  verbose = FALSE,
  ep_only = FALSE,
  long_format = FALSE,
  ...
)
```

Arguments

`scenarios`	table or vector of `EffectScenario` objects
`level`	effect levels in percent (%), defaults to `c(10,50)`
`effect_tolerance`	`numeric`, minimum absolute accuracy of effect levels

factor_cutoff	optional numeric, the search for a multiplication factor will be cut short if tried factors exceed this value; the result will report the cutoff value as the final EPx value.
min_factor	numeric, if tried factors fall below this threshold, the algorithm will halt with an error
max_factor	numeric, if tried factors exceed this threshold, the algorithm will halt with an error
verbose	logic, if TRUE then infos about model evaluations are displayed
ep_only	logical, if TRUE then only EPx values are part of the output, any contextual information such as EffectScenario objects are left out
long_format	logical, if TRUE then EPx values are returned as a table in long format, any contextual information will be duplicated
...	additional arguments passed on to effect()

Details

To estimate EPx values, a *binary search* on multiplication factors is conducted. The algorithm can achieve arbitrary precision in terms of effects. The same approach is implemented in the morse package in the MFx() function. Convergence is often achieved in less than 10 iterations per effect level and endpoint.

Internally, a knowledge base of all tried factors and resulting effect levels is kept to speed up convergence if more than one endpoint or effect level was requested. The algorithm will automatically sweep the range of multiplication factors as needed but hard cutoff values are implemented to avoid infinite loops; the algorithm will halt with an error message if tried factors are smaller than 1e-30 or greater than 1e30.

Numerical precision:

The precision of reported *EPx* values is controlled by the argument effect_tolerance and is given as the upper absolute error threshold of effects that is deemed acceptable. The default value of 0.001 ensures that a derived *EPx* will result in an effect of x% ± 0.1. Decreasing the effect_tolerance will result in additional model iterations and longer runtime. Setting an extremely small tolerance value may lead to a breakdown of the algorithm due to the occurrence of extremely small, quasi-random numerical errors in simulation results.

Value

The original tibble with additional columns named after the request effect levels, e.g. L.EP10. If no tibble was used as argument, then a new one is created. The first column scenario will contain the supplied EffectScenario objects.

Examples

```
minnow_sd %>% epx()
minnow_sd %>% epx(level=c(10,23,42))

# displays infos about tested multiplication factors
minnow_sd %>% epx(verbose=TRUE)
```

```
# return results as a table in wide format
minnow_sd %>% epx(long_format=TRUE)
```

---

epx_mtw *Calculate EPx values for a series of moving time window*

Description

Calls `epx()` to calculate the EPx value (i.e. the multiplication factors of an exposure profile that cause x% of effect) for moving windows with length `window_length` that move timesteps defined by `window_interval`.

Usage

```
epx_mtw(
  x,
  level = c(10, 50),
  factor_cutoff = 1000,
  window_length = 7,
  window_interval = 1,
  ...
)
```

Arguments

`x`	a scenario
`level`	The target effect level of the effect, ie. the x of EPx.
`factor_cutoff`	above which cutoff is the EPx is not relevant
`window_length`	the length of the moving time window
`window_interval`	the interval that the moving time window moves
`...`	arguments passed to epx

Value

a tibble with five columns

- window.start
- window.end
- endpoint
- level
- EPx

Examples

```
metsulfuron %>%
  set_window(length=7, interval=1) %>%
  epx_mtw()
```

`explore_space` *Explore parameter space*

Description

[Experimental]

The function is aimed at getting an idea of how the parameter space of a model behaves, so that parameter identifiability problems and correlations between parameters can be explored. Therefore, the function samples a large number of parameter sets by randomly drawing from each parameter's 95% confidence interval (generated by `lik_profile()`). It then checks how many of the parameter sets are within acceptable limits by comparing the likelihood ratio of a parameter set vs. the original parameter set against a chi-square distribution as degrees of freedom (df) the total number of profile parameters (outer rim) or one df (inner rim). If needed, the function resamples until at least `nr_accept` parameters sets are within the inner rim

Usage

```
explore_space(
  x,
  par,
  res,
  output,
  data,
  sample_size = 1000,
  max_iter = 30,
  nr_accept = 100,
  sample_factor = 1.2,
  individual = FALSE,
  log_scale = FALSE,
  data_type = c("continuous", "count"),
  max_runs = deprecated(),
  ...
)
```

Arguments

`x`	a list of caliset objects
`par`	best fit parameters from joined calibration
`res`	output of `lik_profile()` function
`output`	character vector, name of output column of `simulate()` that is used in calibration
`data`	only needed if `x` is a scenario
`sample_size`	number of samples to draw from each parameter interval
`max_iter`	max number of iterations to redraw samples (within a smaller space), and repeat the process

nr_accept	threshold for number of points sampled within the inner circle
sample_factor	multiplication factor for sampling (95% interval * sample factor)
individual	if FALSE (default), the log likelihood is calculated across the whole dataset. Alternatively, if TRUE, log likelihoods are calculated for each (group of) *set*(s) individually.
log_scale	FALSE (default), option to calculate the log likelihood on a log scale (i.e., observations and predictions are log transformed during calculation)
data_type	Character argument, "continuous" (default) or "count", to specify the data type for the log likelihood calculations.
max_runs	*deprecated* alias for max_iter parameter
...	additional parameters passed through to simulate()

Value

a list containing a plot to explore the parameter space, and the data.frame supporting it

See Also

plot.param_space

Examples

```
library(dplyr)
# Example with Lemna model - physiological params
# Before applying the function, a model needs to be calibrated and its parameters profiled
# Inputs for likelihood profiling

# observations - control run
obs <- schmitt2013 %>%
  filter(trial == "T0")

# parameters after calibration
params <- c(
  k_phot_max = 5.663571,
  k_resp = 1.938689,
  Topt = 26.7
)

# set parameter boundaries (if different from defaults)
bounds <- list(
  k_resp = list(0, 10),
  k_phot_max = list(0, 30),
  Topt = list(20, 30)
)

# update metsulfuron
myscenario <- metsulfuron %>%
  set_init(c(BM = 1.2, E = 1, M_int = 0)) %>%
  set_param(list(
    k_0 = 5E-5,
```

```
    a_k = 0.25,
    BM50 = 17600,
    mass_per_frond = 0.1
  )) %>%
  set_noexposure() %>%
  set_param(params) %>%
  set_bounds(bounds)

# Likelihood profiling
res <- lik_profile(
  x = myscenario,
  data = obs,
  output = "FrondNo",
  par = params,
  refit = FALSE,
  type = "fine",
  method = "Brent"
)
# plot
plot(res)

# parameter space explorer
set.seed(1) # for reproducibility
res_space <- explore_space(
  x = myscenario,
  data = obs,
  par = params,
  res = res,
  output = "FrondNo",
  sample_size = 1000,
  max_iter = 20,
  nr_accept = 100)

plot(res_space)
```

`ExposureSeries` *Exposure time-series*

Description

Creates an object that encapsulates an exposure time-series with its metadata, such as formatted datetime strings and file name where the series was loaded from. `no_exposure()` is shorthand to create a time-series of constant zero exposure.

Usage

```
ExposureSeries(series, dates, file, meta, context)
```

Arguments

series	data.frame with two columns containing a time-series
dates	vector, optional original list of time stamps
file	character, optional file name where data originates from
meta	list, optional metadata
context	list optional contextual metadata such as project ids

Value

an S4 object of type ExposureSeries

Slots

dates original time points of time-series, e.g. time stamps of the form 2000-01-01 12:00

file character, file name where data originates from, may be empty

meta list, contains metadata

context list, contains contextual metadata, such as project ids

series data.frame containing the actual time-series

See Also

no_exposure()

---

fit_growth *Fit growth parameters*

Description

[Experimental] High-level function to fit growth and/or loss parameters to observed data. It eases the use of calibrate() by providing sensible defaults for various (model dependent) settings.

Usage

```
fit_growth(x, data, ...)

## S4 method for signature 'ANY,ANY'
fit_growth(x, data, ...)

## S4 method for signature 'LemnaSetac,missing'
fit_growth(x, data, par, log_scale = TRUE, verbose = FALSE, ...)

## S4 method for signature 'LemnaSchmitt,missing'
fit_growth(x, data, par, log_scale = TRUE, verbose = FALSE, ...)

## S4 method for signature 'Magma,missing'
fit_growth(x, data, par, log_scale = TRUE, verbose = FALSE, ...)
```

Arguments

x	a scenario or a list of caliset objects
data	toxicological trial data to fit growth parameters to: required if x is either a scenario or the result of a fit. See Section *Data* for details.
...	additional arguments passed through to `calibrate()`
par	named vector, of parameters to fit and their starting values
log_scale	logical, if TRUE then fitting will be performed on log-transformed observations and predictions, else the data will be used as-is
verbose	logical, if TRUE then info messages are printed to the console

Value

a list

Methods (by class)

- `fit_growth(x = ANY, data = ANY)`: Default handler
- `fit_growth(x = LemnaSetac, data = missing)`: Fit growth parameters of Lemna_SETAC scenarios
- `fit_growth(x = LemnaSchmitt, data = missing)`: Fit growth parameters of Lemna_Schmitt scenarios
- `fit_growth(x = Magma, data = missing)`: Fit growth parameters of Magma scenarios

Examples

```
# Use experimental data from control trial ('T0') to fit growth rate
ctrl <- schmitt2013[schmitt2013$trial == "T0" ,]

# Set up a scenario, provide dummy parameter values where necessary
sc <- Lemna_Schmitt() %>%
  set_init(c(BM=0.0012)) %>%
  set_param(c(EC50=1, b=1, P_up=1))

# Run fitting routine
fit_growth(sc, data=ctrl, verbose=TRUE)
```

---

fit_tktd *Fit TK/TD parameters*

Description

[Experimental] High-level function to fit TK/TD parameters to observed data. It eases the use of `calibrate()` by providing sensible defaults for various (model dependent) settings.

Usage

```
fit_tktd(x, data, ...)

## S4 method for signature 'LemnaSetac,missing'
fit_tktd(x, data, par, log_scale = TRUE, verbose = FALSE, ...)

## S4 method for signature 'LemnaSchmitt,missing'
fit_tktd(x, data, par, log_scale = TRUE, verbose = FALSE, ...)

## S4 method for signature 'Magma,missing'
fit_tktd(x, data, par, log_scale = TRUE, verbose = FALSE, ...)
```

Arguments

x	a scenario or a list of caliset objects
data	toxicological trial data to fit TK/TD parameters to: required if x is either a scenario or the result of a fit. See Section *Data* for details.
...	additional arguments passed through to `calibrate()`
par	named vector, of parameters to fit and their starting values
log_scale	logical, if TRUE then fitting will be performed on log-transformed observations and predictions, else the data will be used as-is
verbose	logical, if TRUE then info messages are printed to the console

Details

Data:

The function can be used in three basic ways: Fit parameters of

1. a single scenario to a data set
2. a list of calisets
3. a list of calisets in a chain of `fit` functions

For option 1), scenario and data are supplied separately. In this case, supported types of argument `data` are `data.frames` and tox_data objects. Any `data.frame` must have a format that is compatible with the `tox_data()` function.

In option 2), all conditions are fully described by a list of calibration sets. The user is responsible to set up all caliset objects to their needs.

Option 3) is for convenience purposes and allows the chaining of fit functions. The latter alternative accepts the return value of e.g. `fit_growth()` as argument `x` and the fitted parameter values are applied to all calibration sets in argument `data`.

Value

a list

Methods (by class)

- `fit_tktd(x = LemnaSetac, data = missing)`: Fit TK/TD parameters of Lemna_SETAC scenarios
- `fit_tktd(x = LemnaSchmitt, data = missing)`: Fit TK/TD parameters of Lemna_Schmitt scenarios
- `fit_tktd(x = Magma, data = missing)`: Fit TK/TD parameters of Magma scenarios

Examples

```
# Use experimental data from control trial ('T0') to fit growth rate
ctrl <- schmitt2013[schmitt2013$trial == "T0" ,]

# Set up a scenario, provide dummy parameter values where necessary
sc <- Lemna_Schmitt() %>%
  set_init(c(BM=0.0012)) %>%
  set_param(c(EC50=1, b=1, P_up=1))

# Run fitting routine
fit_growth(sc, data=ctrl, verbose=TRUE)

# Use fitted growth parameter to adapt scenario
sc2 <- sc %>% set_param(c(k_phot_max=0.43925))

# Use experimental ecotox data for various concentrations of 'metsulfuron-methyl'
trials <- schmitt2013[schmitt2013$trial != "T0" ,]

# Fit remaining TK/TD parameters
fit_tktd(sc2, data=trials, verbose=TRUE)
```

---

focusd1 *A Lemna_SETAC scenario with variable environment*

Description

A mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for the aquatic macrophytes Lemna spp. as published by Klein *et al.* (2021).

Usage

```
focusd1
```

Format

An object of class `LemnaSetac` of length 1.

Details

The scenario will simulate a period of 365 days, a start population of 80 g/m² dry weight, variable environmental conditions, and a complex, time-varying exposure pattern.

The scenario setup was published by Hommen *et al.* (2015). Exposure pattern and substance specific parameters are of exemplary character and represent the herbicide *metsulfuron-methyl*. The parameters were derived by Schmitt et al. (2013) based on literature data.

References

Hommen U., Schmitt W., Heine S., Brock Theo CM., Duquesne S., Manson P., Meregalli G., Ochoa-Acuña H., van Vliet P., Arts G., 2015: How TK-TD and Population Models for Aquatic Macrophytes Could Support the Risk Assessment for Plant Protection Products. Integr Environ Assess Manag 12(1), pp. 82-95. doi:10.1002/ieam.1715

Klein J., Cedergreen N., Heine S., Reichenberger S., Rendal C., Schmitt W., Hommen U., 2021: Refined description of the *Lemna* TKTD growth model based on *Schmitt et al.* (2013) - equation system and default parameters. Report of the working group *Lemna* of the SETAC Europe Interest Group Effect Modeling. Version 1.1, uploaded on 09 May 2022. https://www.setac.org/group/effect-modeling.html

Schmitt W., Bruns E., Dollinger M., Sowig P., 2013: Mechanistic TK/TD-model simulating the effect of growth inhibitors on *Lemna* populations. Ecol Model 255, pp. 1-10. doi:10.1016/j.ecolmodel.2013.01.017

See Also

Lemna-models

Examples

```
# Simulate the example scenario
focusd1 %>% simulate()
```

`fx` *Generic to calculate effects for a particular scenario*

Description

Generic to calculate effects for a particular scenario

Usage

```
fx(scenario, ...)

## S4 method for signature 'ANY'
fx(scenario, ...)
```

```
## S4 method for signature 'ScenarioSequence'
fx(scenario, ...)

## S4 method for signature 'Lemna'
fx(scenario, ...)

## S4 method for signature 'Magma'
fx(scenario, ...)

## S4 method for signature 'Algae'
fx(scenario, ...)

## S4 method for signature 'GutsSd'
fx(scenario, ...)

## S4 method for signature 'GutsIt'
fx(scenario, ...)

## S4 method for signature 'GutsRedSd'
fx(scenario, ...)

## S4 method for signature 'GutsRedIt'
fx(scenario, ...)
```

Arguments

`scenario`	scenario object
`...`	additional parameters

Value

numeric named vector

Methods (by class)

- `fx(ANY)`: Use state variables at end of simulation
- `fx(ScenarioSequence)`: Wrapper for sequences
- `fx(Lemna)`: Effect at end of simulation of Lemna-models
- `fx(Magma)`: Effect at end of simulation of Magma scenarios
- `fx(Algae)`: Effect at end of simulation of Algae-models
- `fx(GutsSd)`: Calculates lethality of GUTS-SD scenarios
- `fx(GutsIt)`: Calculates lethality of GUTS-IT scenarios
- `fx(GutsRedSd)`: Calculates lethality of GUTS-RED-SD scenarios
- `fx(GutsRedIt)`: Calculates lethality of GUTS-RED-IT scenarios

get_model *Get model name*

Description

Returns the unique model name that is associated with a scenario, e.g. `GUTS-RED-IT`. The function supports vectorized arguments.

Usage

```
get_model(x)
```

Arguments

x	(vector of) scenarios or `parameter_set` objects

Value

vector of `character`

Examples

```
# returns `GUTS-RED-IT`
get_model(minnow_it)
```

get_param *Get scenario parameters*

Description

For scenario sequences, only the parameters of the first scenario in the sequence is returned. To access parameters of a specific scenario in the sequence, use `get_param()` on the individual scenario object.

Usage

```
get_param(x)

## S4 method for signature 'list'
get_param(x)

## S4 method for signature 'EffectScenario'
get_param(x)

## S4 method for signature 'ScenarioSequence'
get_param(x)
```

```
## S4 method for signature 'ParameterSet'
get_param(x)
```

Arguments

x	object to fetch parameters from

Value

(list of) list(s) with key-value pairs

Methods (by class)

- `get_param(list)`: Returns a list of parameter lists (if applicable)
- `get_param(EffectScenario)`: Returns a list parameters for a single scenario
- `get_param(ScenarioSequence)`: Returns a list of parameter lists, one for each scenario in the sequence
- `get_param(ParameterSet)`: Returns a list of parameters for a single parameter_set

See Also

`set_param()`

Examples

```
minnow_it %>% get_param()
```

---

get_tag *Get scenario tag*

Description

Returns the user-defined, custom tag of a scenario, if available. Tags can be helpful to quickly distinguish scenarios by e.g. a user-specified string. The function supports vectorized inputs. If more than one scenario is supplied in argument x, then a list of tags is returned.

Usage

```
get_tag(x)

## S4 method for signature 'list'
get_tag(x)

## S4 method for signature 'EffectScenario'
get_tag(x)
```

```
## S4 method for signature 'ScenarioSequence'
get_tag(x)

## S4 method for signature 'ParameterSet'
get_tag(x)
```

Arguments

x	(vector of) scenarios or parameter_set objects

Value

(list of) tag(s), returns NA if no tag was set

Methods (by class)

- `get_tag(list)`: Returns a list of tags (if applicable)
- `get_tag(EffectScenario)`: Returns the tag of a single scenario
- `get_tag(ScenarioSequence)`: Returns a list of tags, one for each scenario in the sequence
- `get_tag(ParameterSet)`: Returns the tag of a single parameter_set

See Also

`set_tag()`

Examples

```
# returns `fathead minnow`
get_tag(minnow_it)

# update or set a tag
myscenario <- GUTS_RED_IT() %>% set_tag("My Custom Tag")
# returns `My Custom Tag`
get_tag(myscenario)
```

---

get_times *Get output times*

Description

Get output times

Usage

```
get_times(x)
```

Arguments

x	(vector of) scenario objects

Value

(list of) times vector

See Also

`set_times()`

Examples

```
# Create a scenario
myscenario <- GUTS_RED_IT() %>% set_times(0:5)
# Returns the defined output times
get_times(myscenario)
```

`GUTS-RED-models` *GUTS-RED models*

Description

Reduced *General Unified Threshold models of Survival* (GUTS) with stochastic death (*SD*) and individual tolerance (*IT*)

Details

The TKTD models *GUTS-RED-SD* and *GUTS-RED-IT* were described by EFSA (2018). GUTS-RED models assume a one-compartment model which directly links external concentration to the scaled damage. The scaled damage is given in units of concentration, equal to the units of measurement in the external medium, e.g. ug/L. The damage dynamics is connected to an individual hazard state variable, resulting in simulated mortality when an internal damage threshold is exceeded. The death mechanisms stochastic death (*SD*) and individual threshold (*IT*) are extreme cases of the *GUTS* theory.

For *SD* models, the threshold parameter for lethal effects is fixed and identical for all individuals of a group, meaning that the variance of the threshold values is zero. Hence, the killing rate relates the probability of a mortality event in proportion to the scaled damage. For *IT* models, the thresholds for effects are distributed among individuals of a group. Mortality of an individual follows immediately once the individual's tolerance is exceeded. Meaning in model terms that the killing rate is set to infinity (EFSA 2018).

State variables

The following list describes the default names and standard units of *GUTS-RED* state variables:

- `D`, scaled damage (conc)
- `H`, cumulative hazard (-)

The state variables are initialized with zero by default.

SD model parameters

- kd, dominant rate constant (time^-1)
- hb, background hazard rate (time^-1)
- z, threshold for effects (conc)
- kk, killing rate constant (time^-1)

IT model parameters

- kd, dominant rate constant (time^-1)
- hb, background hazard rate (time^-1)
- alpha, median of thresholds (conc)
- beta, shape parameter (-)

Effects

The effect endpoint L (lethality) is available for *GUTS-RED* models. A value of zero (0.0) denotes *no effect* on organism survival. A value of one (1.0) denotes a lethality rate of 100%, i.e. no survivors.

The survival probability S is available in the return value of simulate().

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

Other GUTS-RED models: GUTS_RED_IT(), GUTS_RED_SD()

Other models: Algae-models, DEB-models, Lemna-models, Macrophyte-models

---

GUTS_IT *GUTS-IT scenario*

Description

[Experimental] Full *General Unified Threshold models of Survival* (GUTS) with Individual Tolerance compatible with the *Full GUTS* model as described by EFSA (2018), but some parameter names may differ.

Usage

```
GUTS_IT(scaled_ci = FALSE, dose_metric = c("D", "Ci", "Cw"))
```

Arguments

`scaled_ci`	`logical`, switch to enable scaling. If `TRUE`, the model will use the scaled internal concentration `Ci*`. Else no scaling. Default value is `FALSE`.
`dose_metric`	`character`, selects the dose metric. `'D'` to use damage, `'Ci'` to for internal concentration, or `'Cw'` for the external concentration. Default value is `'D'`.

Value

an S4 object of type `GutsIt-class`

State variables

The following list describes the default names and standard units of *GUTS* state variables:

- `Ci`, (scaled) internal concentration (conc)
- `D`, (scaled) damage (*)
- `H`, cumulative hazard (-)

The state variables are initialized with zero by default.

IT model parameters

The set of parameters and their names follows the definition by Jager et al (2011). The actual number of required parameters depends on the selected model variant, i.e. if the internal concentration is scaled or not, as well as the selected dose metric. The full set of parameters is as follows

- `ki`, accumulation rate into body (time^-1)
- `ke`, elimination rate (time^-1)
- `Kiw`, scaling constant for external concentration (*)
- `kr`, damage recovery rate (time^-1)
- `hb`, background hazard rate (time^-1)
- `alpha`, median of thresholds (conc)
- `beta`, shape parameter (-)

Effects

The effect endpoint `L` (lethality) is available for *GUTS* models. A value of zero (`0.0`) denotes *no effect* on organism survival. A value of one (`1.0`) denotes a lethality rate of 100%, i.e. no survivors.

The survival probability `S` is available in the return value of `simulate()`.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

Jager T., Albert C., Preuss T.G., and Ashauer R., 2011: General Unified Threshold Model of Survival - a Toxicokinetic-Toxicodynamic Framework for Ecotoxicology. Environ. Sci. Technol. 45(7), pp. 2529-2540. doi:10.1021/es103092a

See Also

GUTS-RED-IT

Other GUTS models: `GUTS_SD()`

`GUTS_RED_IT` *GUTS-RED-IT scenario*

Description

Reduced *General Unified Threshold models of Survival* (GUTS) with individual tolerance (*IT*).

Usage

```
GUTS_RED_IT(param, init)
```

Arguments

`param`	optional named `list` or `vector` with model parameters
`init`	optional named numeric `vector` to use as initial state

Value

an S4 object of type GutsRedIt

Simulation output

The return value of `simulate()` will contain values for the state variables, as well as an additional column `S` which represents the survival probability for each time point. `S` is calculated as described in EFSA (2018) as *S = (1- F(t))*. The background hazard rate `hb` is already considered in state variable `H` and therefore does not occur as an additional term to derive `S`.

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Model history and changes

- cvasi v1.0.0
  - Model and parameters as described by EFSA PPR (2018).
- cvasi v1.5.0
  - Cumulative damage over time will be calculated on ODE level, instead on results of `simulate()`, to avoid influence of selected output times on predicted survival.

State variables

The following list describes the default names and standard units of *GUTS-RED* state variables:

- `D`, scaled damage (conc)
- `H`, cumulative hazard (-)

The state variables are initialized with zero by default.

IT model parameters

- `kd`, dominant rate constant (time^-1)
- `hb`, background hazard rate (time^-1)
- `alpha`, median of thresholds (conc)
- `beta`, shape parameter (-)

Effects

The effect endpoint `L` (lethality) is available for *GUTS-RED* models. A value of zero (`0.0`) denotes *no effect* on organism survival. A value of one (`1.0`) denotes a lethality rate of 100%, i.e. no survivors.

The survival probability `S` is available in the return value of `simulate()`.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

Other GUTS-RED models: `GUTS-RED-models`, `GUTS_RED_SD()`

`GUTS_RED_SD` *GUTS-RED-SD scenario*

Description

Reduced *General Unified Threshold models of Survival* (GUTS) with stochastic death (*SD*).

Usage

```
GUTS_RED_SD(param, init)
```

Arguments

`param`	optional named `list` or `vector` with model parameters
`init`	optional named numeric `vector` to use as initial state

Value

an S4 object of type GutsRedSd

Simulation output

The return value of `simulate()` will contain values for the state variables, as well as an additional column `S` which represents the survival probability for each time point. `S` is calculated as described in EFSA (2018) as $S = exp(-H)$. The background hazard rate `hb` is already considered in state variable `H` and therefore does not occur as an additional term to derive `S`.

State variables

The following list describes the default names and standard units of *GUTS-RED* state variables:

- `D`, scaled damage (conc)
- `H`, cumulative hazard (-)

The state variables are initialized with zero by default.

SD model parameters

- `kd`, dominant rate constant (time^-1)
- `hb`, background hazard rate (time^-1)
- `z`, threshold for effects (conc)
- `kk`, killing rate constant (time^-1)

Effects

The effect endpoint `L` (lethality) is available for *GUTS-RED* models. A value of zero (`0.0`) denotes *no effect* on organism survival. A value of one (`1.0`) denotes a lethality rate of 100%, i.e. no survivors.

The survival probability `S` is available in the return value of `simulate()`.

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

Other GUTS-RED models: `GUTS-RED-models`, `GUTS_RED_IT()`

`GUTS_SD` *GUTS-SD scenario*

Description

[Experimental] Full *General Unified Threshold models of Survival* (GUTS) with stochastic death (*SD*). The model was defined by Jager et al. (2011). It is compatible with the *Full GUTS* model as described by EFSA (2018), but some parameter names may differ.

Usage

```
GUTS_SD(scaled_ci = FALSE, dose_metric = c("D", "Ci", "Cw"))
```

Arguments

`scaled_ci`	`logical`, switch to enable scaling. If `TRUE`, the model will use the scaled internal concentration `Ci*`. Else no scaling. Default value is `FALSE`.
`dose_metric`	`character`, selects the dose metric. `'D'` to use damage, `'Ci'` to for internal concentration, or `'Cw'` for the external concentration. Default value is `'D'`.

Value

an S4 object of type `GutsSd-class`

State variables

The following list describes the default names and standard units of *GUTS* state variables:

- `Ci`, (scaled) internal concentration (conc)
- `D`, (scaled) damage (*)
- `H`, cumulative hazard (-)

The state variables are initialized with zero by default.

SD model parameters

The set of parameters and their names follows the definition by Jager et al (2011). The actual number of required parameters depends on the selected model variant, i.e. if the internal concentration is scaled or not, as well as the selected dose metric. The full set of parameters is as follows

- `ki`, accumulation rate into body (time^-1)
- `ke`, elimination rate (time^-1)
- `Kiw`, scaling constant for external concentration (*)
- `kr`, damage recovery rate (time^-1)
- `kk`, killing rate constant (time^-1)
- `hb`, background hazard rate (time^-1)
- `z`, threshold for effects (*)

Effects

The effect endpoint `L` (lethality) is available for *GUTS* models. A value of zero (`0.0`) denotes *no effect* on organism survival. A value of one (`1.0`) denotes a lethality rate of 100%, i.e. no survivors.

The survival probability `S` is available in the return value of `simulate()`.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

Jager T., Albert C., Preuss T.G., and Ashauer R., 2011: General Unified Threshold Model of Survival - a Toxicokinetic-Toxicodynamic Framework for Ecotoxicology. Environ. Sci. Technol. 45(7), pp. 2529-2540. doi:10.1021/es103092a

See Also

GUTS-RED-SD

Other GUTS models: `GUTS_IT()`

`import_morse` *Import* `morse` *model parameters*

Description

Loads GUTS model parameters which were fitted by the morse package.

Usage

```
import_morse(
  fit,
  find_sd = TRUE,
  find_it = TRUE,
  reset_hb = FALSE,
  params = c("estim", "all"),
  mcmc_size,
  find.SD = deprecated(),
  find.IT = deprecated(),
  reset.hb = deprecated(),
  mcmc.size = deprecated(),
  file = deprecated()
)

morse(...)
```

Arguments

`fit`	Either a string with a file path to an *.Rdata* or *.RDS* file containing a *morse* fit, or a *morse* fit object itself
`find_sd`	a logical value. If `TRUE`, it will try to find fitted parameters of a *GUTS-RED-SD* model
`find_it`	a logical value. If `TRUE`, it will try to find fitted parameters of a *GUTS-RED-IT* model
`reset_hb`	a logical value. If `TRUE`, the background hazard rate `hb` is set to zero
`params`	`character`, if set to `"estim"` then only the best-fit parameters are imported, else all parameter sets in the MCM chains are returned
`mcmc_size`	optional `integer`, sets the maximum number of imported parameter sets per MCMC. By default, all MSMS parameter samples are imported.
`find.SD`	*deprecated*, alias for parameter `find_sd`
`find.IT`	*deprecated*, alias for parameter `find_it`
`reset.hb`	*deprecated*, alias for parameter `rest_hb`
`mcmc.size`	*deprecated*, alias for parameter `mcmc_size`
`file`	*deprecated*, alias for parameter `fit`
`...`	Arguments passed on to `import_morse()`

Value

list of parameter_set objects

Functions

- `morse()`: deprecated alias

Examples

```
# import all parameter fits
try(import_morse("path/to/morse_fit.RData"))

# import parameters for a specific model
try(import_morse("path/to/morse_fit.RData", find_it=TRUE, find_sd=FALSE))

# modify model objects
try(models %>% set_param(import_morse("path/to/morse_fit.RData")))
```

---

import_swash *SWASH project exposure profile import*

Description

Read all TOXSWA files within a SWASH project directory.

Usage

```
import_swash(swash_dir, ...)
```

Arguments

swash_dir	path to the SWASH project directory
...	arguments passed on to import_toxswa()

Value

a list of imported exposure series, see import_toxswa() for details

---

import_toxswa *Import* TOXSWA *exposure series*

Description

Read one or more *TOXSWA* exposure series from *TOXSWA*'s `.out` files. By default, the concentration dissolved in water (*ConLiqWatLay*) at the end of the simulated waterbody (i.e. at the maximum of the *x* dimension) is returned. The unit of the time scale as well as of the imported model output variable can be scaled as needed.

Usage

```
import_toxswa(
  files,
  alias = NA,
  output_var = "ConLiqWatLay",
  output_unit = "ug/L",
  time_unit = "days",
  substance = NULL,
  split = TRUE
)
```

Arguments

files	vector of strings with absolute or relative paths to files
alias	optional vector with strings, will be used as an alias to identify a TOXSWA series instead of its filename
output_var	character, single output variable from *TOXSWA* that is imported, defaults to *ConLiqWatLay*
output_unit	character, target unit of the imported output variable, defaults to *ug/L*, syntax must be compatible with `units::units()`
time_unit	character, target unit of the imported time scale, defaults to *days*, syntax must be compatible with `units::units()`
substance	optional vector of characters, if set, only the substance codes defined in this vector are imported
split	logical, if `TRUE` then one series will be returned for each substance found in the *TOXSWA* files, else all substances per file will be in one *data.frame*. Defaults to `TRUE`

Details

The numerical time scale is shifted to always start at time zero (0.0). Numerical columns of the returned *data.frame* objects will be of type units::units. Please be aware that the use of `units` objects may not be supported by all functions in this package. However, `set_times()` and `set_exposure()` can handle `units` objects safely.

Incomplete list of alternative *TOXSWA* v5.5.3 output variables:

- *ConLiqWatLay*: Concentration dissolved in water (g/m3)
- *ConLiqSed*: Concentration in pore water sediment (g/m3)
- *ConSysWatLay*: Total concentration in water (g/m3)
- *CntSorSusSol*: Content sorbed to suspended solids (g/kg)
- *CntSorSed*: Content sorbed to sediment (g/kg)

Value

list of *data.frame* objects with exposure series. Each *data.frame* has at least three columns:

- `time`: numerical time scale, always starts at zero
- `timestamp`: time as datetime objects such as `POSIXct`
- one or more additional columns for each imported substance

---

is_deb *Test if argument is a DEB scenario*

Description

Test if argument is a DEB scenario

Usage

```
is_deb(x)

is_DEB(x)
```

Arguments

`x`	vector of `EffectScenario` objects

Value

vector of `logical` values

Functions

- `is_DEB()`: Deprecated alias.

---

is_guts *Test if argument is a GUTS scenario*

Description

Test if argument is a GUTS scenario

Usage

```
is_guts(x)

is_GUTS(x)

is_guts_it(x)

is_GUTS_IT(x)

is_guts_sd(x)

is_GUTS_SD(x)
```

Arguments

x	vector of `EffectScenario` objects

Value

vector of `logical` values

Functions

- `is_GUTS()`: Deprecated alias.
- `is_guts_it()`: Test if argument is a GUTS-IT scenario
- `is_GUTS_IT()`: Deprecated alias.
- `is_guts_sd()`: Test if argument is a GUTS-SD scenario
- `is_GUTS_SD()`: Deprecated alias.

Examples

```
# returns `TRUE`
is_guts(minnow_it)
is_guts(GUTS_RED_IT())

# returns `c(TRUE,TRUE,TRUE)`
is_guts(c(minnow_it, minnow_it, minnow_it))

# returns `FALSE`
is_guts_sd(minnow_it)
```

is_lemna *Test if argument is a Lemna scenario*

Description

Test if argument is a Lemna scenario

Usage

```
is_lemna(x)

is_Lemna(x)
```

Arguments

x	vector of scenario objects

Value

vector of logical values

Functions

- `is_Lemna()`: Deprecated alias.

is_LemnaThreshold *Test if argument is a LemnaThreshold scenario*

Description

[Deprecated]

Usage

```
is_LemnaThreshold(x)
```

Arguments

x	vector of scenarios objects

Value

vector of `logical` values

See Also

`is_lemna()`

is_scenario *Test if argument is an effect scenario*

Description

Supports vectorized arguments.

Usage

```
is_scenario(x)
```

Arguments

x	Some value or object

Value

vector of `logical` values

Examples

```
# returns `TRUE`
is_scenario(minnow_it)

# returns `FALSE`
is_scenario(list())
```

Lemna-models *Lemna models*

Description

Overview of supported *Lemna* models

Details

- `Lemna_Schmitt()` by Schmitt *et al.* (2013)
- `Lemna_SETAC()` by Klein *et al.* (2021)

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

See Also

Macrophyte-models

Other Lemna models: `Lemna_SETAC()`, `Lemna_Schmitt()`

Other models: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Macrophyte-models`

`Lemna_Schmitt` *Lemna model (Schmitt et al. 2013)*

Description

The model is a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for the aquatic macrophytes *Lemna spp.* The model simulates the development of *Lemna* biomass under laboratory and environmental conditions and was developed by Schmitt *et al.* (2013). Growth of the *Lemna* population is simulated on basis of photosynthesis and respiration rates which are functions of environmental conditions. The toxicodynamic sub-model describes the effects of growth-inhibiting substances by a respective reduction in the photosynthesis rate based on internal concentrations. This is the historical version of the Lemna model. For current uses, we recommend the Lemna (SETAC) model, which is a more recent version of the Schmitt model.

Usage

```
Lemna_Schmitt(param, init)

Lemna_SchmittThold(param, init)
```

Arguments

`param`	optional named `list` or `vector` of model parameters
`init`	optional named numeric `vector` of initial state values

Details

Constructors to ease creation of scenarios based on the *Lemna* model by Schmitt *et al.* (2013). A variant of this *Lemna* model, `Lemna_SchmittThold()`, provides an additional cumulative exposure threshold parameter. The Lemna biomass stops growing if the integral of exposure over time exceeds the threshold. The integral of exposure is internally accounted for by an additional state variable `AUC` (Area Under Curve).

Value

an S4 object of type LemnaSchmitt

Functions

- `Lemna_SchmittThold()`: model variant with cumulative exposure threshold

State variables

The following list describes the default names and standard units of the model's state variables:

- BM, g_dw/m2, dry weight biomass per square meter
- E, -, effect [0,1]
- M_int, ug, internal toxicant mass
- AUC, ug/L, cumulative exposure (**only** for `LemnaThreshold` model)

Biomass (BM) and internal toxicant mass (M_int) are initialized to zero by default. See `set_init()` on how to set the initial states.

Model parameters

The following model parameters are required:

- Fate and biomass
  - `k_phot_fix`, logical, TRUE then k_phot_max is not changed by environmental factors, else FALSE
  - `k_phot_max`, 1/d, maximum photosynthesis rate
  - `k_resp`, 1/d, respiration rate
  - `k_loss`, 1/d, rate of loss (e.g. flow rate)
  - `mass_per_frond`, g_dw/frond, dry weight per frond
  - `BMw2BMd`, g_fw/g_dw, Fresh weight/dry weight
- Effect
  - `Emax`, -, maximum effect [0,1]
  - `EC50`, ug/L, midpoint of effect curve
  - `b`, -, slope of effect curve
- Toxicokinetics
  - `P_up`, cm/d, Permeability for uptake
  - `AperBM`, cm2/g_dw, A_leaf / d_leaf = 1/d_leaf (for circular disc, d=0.05 cm)

  - Kbm, -, Biomass(fw) : water partition coefficient
  - P_Temp, logical, TRUE to enable temperature dependence of cuticle permeability, else FALSE
  - MolWeight, g/mol, Molmass of molecule (determines Q10_permeability)
- Temperature dependence
  - Tmin, deg C, minimum temperature for growth
  - Tmax, deg C, maximum temperature for growth
  - Topt, deg C, optimal temperature for growth
  - t_ref, deg C, reference temperature for respiration rate
  - Q10, -, temperature dependence factor for respiration rate
- Light dependence
  - k_0, 1/d, light dependence: intercept of linear part
  - a_k, (1/d)/(kJ/m2.d), light dependence: slope of linear part
- Phosphorus dependence (Hill like dep.)
  - C_P, mg/L, phosphorus concentration in water
  - CP50, mg/L, phosphorus conc. where growth rate is halfed
  - a_p, -, Hill coefficient
  - KiP, mg/L, p-inhibition constant for very high p-conc.
- Nitrogen dependence (Hill like dep.)
  - C_N, mg/L, nitrogen concentration in water
  - CN50, mg/L, n-conc. where growth rate is halfed
  - a_N, -, Hill coefficient
  - KiN, mg/L, n-inhibition constant for very high p-conc.
- Density dependence
  - BM50, g_dw/m2, cut off BM

The Lemna_SchmittThold model requires the following additional parameter:

- threshold, ug/L, cumulative exposure threshold

Forcings

Besides exposure, the model requires four environmental properties as time-series input:

- temp, temperature (°C)
- rad, global irradiation (kJ m-2 d-1)

The following constant default values are used for these properties:

- temp = 12 °C
- rad = 15,000 kJ m-2 d-1

Forcings time-series are represented by data.frame objects consisting of two columns. The first for time and the second for the environmental factor in question.

Entries of the data.frame need to be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant environmental conditions. See scenarios for more details.

Effects

Supported effect endpoints include *BM* (biomass) and *r* (average growth rate during simulation). The effect on biomass is calculated from the last state of a simulation. Be aware that endpoint *r* is incompatible with frond transfers.

Parameter boundaries

Default values for parameter boundaries are set for all parameters by expert judgement, for calibration purposes. Values can be access from the object, and defaults overwritten.

Simulation output

Simulation results will contain two additional columns besides state variables:

- `C_int`, ug/L, internal concentration of toxicant
- `FrondNo`, -, number of fronds

It is possible to amend the output of `simulate()` with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument `nout` of `simulate()`, see examples below. `nout=1` enables reporting of internal concentration (C_int), `nout=14` enables all additional outputs, and `nout=0` will disable additional outputs.

The available output levels are as follows:

- `nout >= 1`: `C_int`, internal concentration (ug/L)
- `nout >= 2`: `FrondNo`, number of fronds (-)
- `nout >= 3`: `C_int_u`, unbound internal concentration (ug/l)
- Growth and TK/TD
  - `nout >= 4`: `BM_fresh`, fresh weight biomass (g_fw/m2)
  - `nout >= 5`: `k_photo_eff`, current photosynthesis rate (1/d)
  - `nout >= 6`: `k_resp_eff`, current respiration rate (1/d)
  - `nout >= 7`: `f_Eff`, toxic effect factor (-)
  - `nout >= 8`: `P_up_eff`, current permeability for uptake (cm/d)
- Environmental variables
  - `nout >= 9`: `actConc`, current toxicant concentration in surrounding medium (ug/L)
  - `nout >= 10`: `actTemp`, current environmental temperature (deg C)
  - `nout >= 11`: `actRad`, current environmental radiation (kJ/m2.d)
- Derivatives
  - `nout >= 12`: `d BM/dt`, current change in state variable BM
  - `nout >= 13`: `d E/dt`, current change in effect
  - `nout >= 14`: `d M_int/dt`, current change in internal toxicant mass

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- `hmax = 0.1` Maximum step length in time suitable for most simulations.

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

References

Schmitt W., Bruns E., Dollinger M., and Sowig P., 2013: *Mechanistic TK/TD-model simulating the effect of growth inhibitors on Lemna populations*. Ecol Model 255, pp. 1-10. doi:10.1016/j.ecolmodel.2013.01.017

See Also

Lemna-models, Macrophyte-models, Transferable, Scenarios

Other Lemna models: `Lemna-models`, `Lemna_SETAC()`

Other macrophyte models: `Lemna_SETAC()`, `Macrophyte-models`, `Magma`

`Lemna_SETAC` *Lemna model v1.3 (Klein et al. 2021)*

Description

The model was described and published by the SETAC Europe Interest Group Effect Modeling (Klein et al. 2021). It is based on the *Lemna* model by Schmitt (2013). The model is a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for the aquatic macrophytes *Lemna spp.*. The model simulates the development of Lemna biomass under laboratory and environmental conditions. Growth of the Lemna population is simulated on basis of photosynthesis and respiration rates which are functions of environmental conditions. The toxicodynamic sub-model describes the effects of growth-inhibiting substances by a respective reduction in the photosynthesis rate based on internal concentrations.

Usage

```
Lemna_SETAC()
```

Value

an S4 object of type LemnaSetac

State variables

The model has two state variables:

- `BM`, Biomass (g dw)
- `M_int`, Mass of toxicant in plant population (ng)

The units of the state variables `BM` and `M_int` are to be interpreted as *per vessel* in laboratory tests, and as *per square meter* for field populations. The same interpretation applies to all parameters relating biomass and toxicant mass to other entities.

Model parameters

- Growth model
  - `k_photo_fixed`, Model switch for unlimited growth conditions (TRUE/FALSE)
  - `k_photo_max`, Maximum photosynthesis rate (d-1)
  - `k_loss`, Reference loss rate (d-1)
  - `BM_threshold`, Lower biomass abundance threshold, (g dw)
  - `BM_min`, Reservoir for biomass recovery, (g dw)
- Temperature response of photosynthesis
  - `T_opt`, Optimum growth temperature (°C)
  - `T_min`, Minimum growth temperature (°C)
  - `T_max`, Maximum growth temperature (°C)
- Temperature response of biomass loss rate

  - Q10, Temperature coefficient (-)
  - T_ref, Reference temperature for response=1 (°C)
- Irradiance reponse of photosynthesis
  - alpha, Slope of irradiance response (m2 d kJ-1)
  - beta, Intercept of irradiance response (-)
- Nutrient response of photosynthesis
  - N_50, Half-saturation constant of Nitrogen (mg N L-1)
  - P_50, Half-saturation constant of Phosphorus (mg P L-1)
- Density dependence of photosynthesis
  - BM_L, Carrying capacity (g dw)
- Concentration response (Toxicodynamics)
  - EC50_int, Internal concentration resulting in 50% effect (ug L-1)
  - E_max, Maximum inhibition (-)
  - b, Slope parameter (-)
- Internal concentration (Toxicokinetics)
  - P, Permeability (cm d-1)
  - r_A_DW, Area per dry-weight ratio (cm2 g-1)
  - r_FW_DW, Fresh weight per dry weight ratio (-)
  - r_FW_V, Fresh weight density (g cm-3)
  - r_DW_FN, Dry weight per frond ratio (g dw)
  - K_pw, Partitioning coefficient plant:water (-)
  - k_met, Metabolisation rate (d-1)

Forcings

External concentrations, alias *exposure*, have to be provided as *ug L-1*.

Besides exposure, the model requires four environmental properties as input:

- tmp, ambient temperature (°C)
- irr, irradiance (kJ m-2 d-1)
- P, Phosphorus concentration (mg P L-1)
- N, Nitrogen concentration (mg N L-1)

The following constant default values are used for these properties:

- tmp = 12 °C
- irr = 15,000 kJ m-2 d-1
- P = 0.3 mg L-1
- N = 0.6 mg L-1

Forcings time-series are represented by data.frame objects consisting of two columns. The first for time and the second for the environmental factor in question.

Entries of the data.frame need to be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant environmental conditions. See scenarios for more details.

Effects

Supported effect endpoints include *BM* (biomass) and *r* (average growth rate during simulation). The effect on biomass is calculated from the last state of a simulation. Be aware that endpoint *r* is incompatible with biomass transfers.

Simulation output

For reasons of convenience, the return value of `simulate()` contains by default two additional variables derived from simulation results: the internal concentration `C_int` as well as the number of fronds `FrondNo`. These can be disabled by setting the argument `nout = 0`.

The available output levels are as follows:

- `nout >= 1`: `C_int`, internal concentration (mass per volume)
- `nout >= 2`: `FrondNo`, frond number (-)
- Response functions
  - `nout >= 3`: `f_loss`, respiration dependency function (-)
  - `nout >= 4`: `f_photo`, photosynthesis dependency function (-)
  - `nout >= 5`: `fT_photo`, temperature response of photosynthesis (-)
  - `nout >= 6`: `fI_photo`, irradiance response of photosynthesis (-)
  - `nout >= 7`: `fP_photo`, phosphorus response of photosynthesis (-)
  - `nout >= 8`: `fN_photo`, nitrogen response of photosynthesis (-)
  - `nout >= 9`: `fBM_photo`, density response of photosynthesis (-)
  - `nout >= 10`: `fCint_photo`, concentration response of photosynthesis (-)
- Environmental variables
  - `nout >= 11`: `C_int_unb`, unbound internal concentration (mass per volume)
  - `nout >= 12`: `C_ext`, external concentration (mass per volume)
  - `nout >= 13`: `Tmp`, temperature (deg C)
  - `nout >= 14`: `Irr`, irradiance (kJ m-2 d-1)
  - `nout >= 15`: `Phs`, Phosphorus concentration (mg P L-1)
  - `nout >= 16`: `Ntr`, Nitrogen concentration (mg N L-1)
- Derivatives
  - `nout >= 17`: `dBM`, biomass derivative (g dw m-2 d-1)
  - `nout >= 18`: `dM_int`, mass of toxicant in plants derivative (mass per m2 d-1)

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- `hmax = 0.1` Maximum step length in time suitable for most simulations.

Model history and changes

- cvasi v1.0.0
  - Model and parameters as described by Klein et al. (2022) report version 1.1
- cvasi v1.5.0
  - Default value of parameter `beta` modified due to typo in previous report versions, now conforms with Klein et al. (2025), report version 1.2
  - New value: `beta=0.25`, old value: `beta=0.025`

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

References

Klein J., Cedergreen N., Heine S., Reichenberger S., Rendal C., Schmitt W., Hommen U., 2021: *Refined description of the Lemna TKTD growth model based on Schmitt et al. (2013) - equation system and default parameters*. Report of the working group *Lemna* of the SETAC Europe Interest Group Effect Modeling. Version 1.3, uploaded in July 2025. https://www.setac.org/group/effect-modeling.html

Schmitt W., Bruns E., Dollinger M., and Sowig P., 2013: *Mechanistic TK/TD-model simulating the effect of growth inhibitors on Lemna populations*. Ecol Model 255, pp. 1-10. doi:10.1016/j.ecolmodel.2013.01.017

See Also

Lemna-models, Macrophyte-models, Transferable, Scenarios

Other Lemna models: `Lemna-models`, `Lemna_Schmitt()`

Other macrophyte models: `Lemna_Schmitt()`, `Macrophyte-models`, `Magma`

`lik_profile` *Likelihood profiling*

Description

[Experimental]

The aim of the function is two-fold: 1) estimate a 95% confidence around each parameter of a calibrated model, and 2) see if perhaps a local minimum was found rather than a global minimum. To achieve this, the likelihood profiling goes through every parameter one by one. For each parameter, the model is sequentially refit with the parameter value set to increasingly lower and higher values, and the likelihood of the model given the data calculated (using `log_lik()`). The likelihood is then compared to the likelihood of the original model (using a likelihood ratio). This leads to the development of a likelihood profile, from which a plot a 95% confidence interval for the parameter is derived.

The idea of the function is a variable stepwise algorithm: When the likelihood ratio changes very little (less than `l_crit_min`), the stepsize is increased (up to a maximum, specified by `f_step_max`). When the lik. ratio changes too much (more than `l_crit_max`), the algorithm tries again with a smaller stepsize (also bound to a minimum: `f_step_min`). Note that the stepsize is used as a fraction of the parameter value that is tried. To prevent very small stepsizes when the value goes towards zero (as can be the case for effect thresholds), an absolute minimum stepsize (`f_step_abs`), which is specified as a fraction of the best parameter value (`Xhat`) (unless it is zero, then the algorithm takes something small).

Note that the likelihood of the model given the data can be calculated across all datasets provided in the calibration set `x`, or calculated separately for each individual dataset before being combined into one likelihood (by adjusting the optional parameter `individual`). The latter has the advantage that different datasets can be given different weights in the likelihood calculation (using the "weight" slot of the caliset objects,`x`). Further, for continuous data (e.g. biomass), the likelihood considers the variance (standard deviation) in the log likelihood calculation, which can vary between datasets when the likelihood is calculated for each dataset separately before combining into an overall likelihood. The latter could be relevant when factors might lead to variability between datasets (e.g. different labs, different animal culture,...)

To conduct the likelihood calculations on separate datasets, the parameter `individual` which by default is 'FALSE' can be set to 'TRUE'. Then, then log likelihoods are calculated for each dataset individually (or in subgroups, using the "tag" names of the caliset object, if provided, to group datasets with the same "tag" before calculating the log likelihood). Subsequently, the log likelihoods for the subsets are combined into an overall likelihood (considering the *set* weights provided in the "weight" slot of the caliset object). Note that for each *set* only 1 weight can be provided (i.e. not individual weights for each datapoint within the *set*), and that *set* with the same tag should have identical weight.

The function was inspired by a MatLab BYOM v.6.8 procedure, created by Tjalling Jager. For details, please refer to BYOM (http://debtox.info/byom.html) as well as Jager (2021).

Usage

```
lik_profile(
```

```
  x,
  par,
  output,
  data = NULL,
  bounds = NULL,
  refit = TRUE,
  type = c("coarse", "fine"),
  individual = FALSE,
  break_prof = FALSE,
  log_scale = FALSE,
  data_type = c("continuous", "count"),
  ...
)
```

Arguments

x	either a single scenario or a list of caliset objects
par	named vector - parameters (names and values) to be profiled
output	character vector, name of output column of `simulate()` that is used in calibration
data	only needed if x is a scenario
bounds	optional list of lists (including lower and upper bound): uses defaults in x object, but can be overwritten here (e.g. bounds <- list(k_resp = list(0,10), k_phot_max = list(0,30)))
refit	if TRUE (default), refit if a better minimum is found
type	"fine" or "coarse" (default) likelihood profiling
individual	if FALSE (default), the log likelihood is calculated across the whole dataset. Alternatively, if TRUE, log likelihoods are calculated for each (group of) *set*(s) individually.
break_prof	If TRUE, then stop the profiling if a better optimum is located. Default is FALSE.
log_scale	FALSE (default), option to calculate the log likelihood on a log scale (i.e., observations and predictions are log transformed during calculation)
data_type	Character argument, "continuous" (default) or "count", to specify the data type for the log likelihood calculations.
...	additional parameters passed on to `calibrate()` and `simulate()`. To avoid parameter confusion, use argument `method` to select optimization algorithms of `calibrate()` and argument `ode_method` to select numerical integration schemes of package `deSolve`.

Value

A list containing, for each parameter profiled, the likelihood profiling results as a dataframe; the 95% confidence interval; the original parameter value; the likelihood plot object; and the recalibrated parameter values (in case a lower optimum was found)

References

Jager T, 2021: Robust Likelihood-Based Optimization and Uncertainty Analysis of Toxicokinetic-Toxicodynamic Models. Integrated Environmental Assessment and Management 17:388-397. doi:10.1002/ieam.4333

Examples

```
# Example with Lemna model - physiological params
library(dplyr)

# observations - control run
obs <- schmitt2013 %>%
  filter(trial == "T0")

# update metsulfuron
myscenario <- metsulfuron %>%
  set_param(c(k_phot_fix = TRUE, Emax = 1)) %>%
  set_init(c(BM = 0.0012)) %>%
  set_noexposure() %>%
  set_bounds(list(k_phot_max=c(0, 1)))

fit <- calibrate(
  x = myscenario,
  par = c(k_phot_max = 1),
  data = obs,
  output = "FrondNo",
  method = "Brent"
)

# Likelihood profiling

res <- lik_profile(
  x = myscenario,
  data = obs,
  output = "FrondNo",
  par = fit$par,
  bounds = list(
    k_phot_max = list(0, 30)
  ),
  refit = FALSE,
  type = "fine",
  method = "Brent"
)
# plot
plot(res)
```

`log_enable` *Start and stop logging*

Description

Start and stop logging

Usage

```
log_enable(file = NULL, append = TRUE, envir = parent.frame())

log_disable()
```

Arguments

file	character, file name or path to a log file
append	logical, if TRUE output will be appended to an existing log file, otherwise the log file will be replaced
envir	log will be automatically disabled if environment is exited, set to NULL to disable

Value

no return value

---

log_envir *Log R environment properties*

Description

Log R environment properties

Usage

```
log_envir()
```

Value

no return value

log_lik *Calculate log likelihood*

Description

Calculates the sum of log likelihoods of each observation given the model parameterization.

Current implementations enable log likelihood calculations for:

1. continuous data, considering a normal distribution around the prediction for each datapoint,
2. count data, considering a multinomial distribution for data reporting the number of survivors over time.

The log likelihood calculation for count data was inspired by a MatLab BYOM v.6.8 procedure, created by Tjalling Jager. For details, please refer to BYOM (http://debtox.info/byom.html) as well as Jager (2021).

Usage

```
log_lik(
  obs,
  pred,
  data_type = c("continuous", "count"),
  log_scale = FALSE,
  npars = NULL
)
```

Arguments

obs	numeric vector of observed values
pred	numeric vector of predicted values
data_type	determines the if likelihood profiling is conducted for "continuous" (default) or "count" data
log_scale	FALSE (default), option to calculate the log likelihood on a log scale (i.e., observations and predictions are log transformed during calculation)
npars	named numeric vector of parameters that the model was calibrated on, required for "continuous" data type, optional for "count".

Value

the log likelihood value

References

Jager T, 2021. Robust Likelihood-Based Optimization and Uncertainty Analysis of Toxicokinetic-Toxicodynamic Models. Integrated Environmental Assessment and Management 17:388-397. doi:10.1002/ieam.4333

Examples

```
# simple example for continuous data #####
# observations
obs <- c(12, 38, 92, 176, 176, 627, 1283, 2640)
# intercept, a, and slope, b, of a Poisson regression fitted through obs
pars <- c(a = 2, b = 0.73)
# predictions with the Poisson regression
pred <- c(15.43, 32.15, 66.99, 139.57, 290.82, 605.94, 1262.52, 2630.58)
# example plot
plot(seq(1:length(obs)), obs)
lines(seq(1:length(obs)), pred)
log_lik(
  obs = obs,
  pred = pred,
  npars = length(pars),
)

# example with count data and GUTS model #####
library(dplyr)
# observational data
dt <- ringtest_c %>% filter(replicate == "E")
myexposure <- dt %>% select(time, conc)
obs <- dt %>%
  mutate(S=Nsurv / max(Nsurv)) %>%
  select(time, S)
# GUTS model
GUTS_RED_IT() %>%
  set_param(c(hb = 0)) %>%
  set_exposure(myexposure) -> myscenario
# fit
fit <- calibrate(
  x = myscenario,
  par = c(kd=1.2, alpha=9.2, beta=4.3),
  data = obs,
  output = "S")
# update
myscenario <- myscenario %>% set_param(fit$par)
# simulate
pred <- myscenario %>% simulate()
pred <- pred$S #* max(obs$S)
obs <- obs$S
# calc likelihood
log_lik(obs,
    pred,
    data_type = "count")
```

---

log_msg *Add a log message*

Description

Message will only appear in the console or in log file if logging was enabled using `log_enable()`.

Usage

```
log_msg(...)
```

Arguments

`...`	elements will be concatenated using `paste0()`

Value

no return value

Examples

```
log_msg("this message will not appear")

log_enable()
log_msg("this message will appear")
log_msg("a number of ","elements to ",42," concatenate")
```

---

log_scenarios *Log scenario properties*

Description

Log scenario properties

Usage

```
log_scenarios(x, header = TRUE)
```

Arguments

`x`	vector of `EffectScenario` objects
`header`	`logical`, if `TRUE` a header line will be printed

Value

unmodified argument `x`

`Macrophyte-models` *Macrophyte models*

Description

Population models of standard test macrophytes, such as *Lemna spp.*

Details

Available macrophyte models:

- Lemna
- Magma

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

See Also

Scenarios

Other macrophyte models: `Lemna_SETAC()`, `Lemna_Schmitt()`, `Magma`

Other models: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Lemna-models`

Magma *Magma model (Witt et al., submitted)*

Description

The *Magma* model interprets the Tier 2C version of the *Lemna* model by Klein et al. (2021), as a generic macrophyte model. It is mathematically equivalent to the Tier 2C version of the model by Klein et al. (2021) with the recommended Tier 2C settings `k_photo_fixed=TRUE` and `k_resp=0`.

Usage

```
Magma(growth = c("exp", "log"))

Myrio()

Myrio_log()
```

Arguments

`growth`	`character`, growth model to simulate: `"exp"` for exponential growth or `"log"` for logistic growth. Default is `"exp"`.

Details

In particular, the growth model is a simple exponential growth model, which is considered to be the typical situation for a laboratory macrophyte study. Instead of frond numbers as for Lemna, the biomass is also returned as total shoot length (*TSL*) in simulation results. Consequently, the model has the additional parameter `r_DW_TSL` (dry weight per total shoot length ratio) instead of `r_DW_FN` (dry weight per frond number ratio). A model variant with an option for logistic growth is provided as well.

Value

an S4 object of type `Magma`

State variables

The model has two state variables:

- `BM`, Biomass (g dw)
- `M_int`, Mass of toxicant in plant population (ng)

Model parameters

The growth model can either simulate exponential growth (the default) or logistic growth. For logistic growth, an additional parameter `D_L` describing the limit density or carrying capacity needs to be provided.

- Growth model
  - mu_control, Maximum photosynthesis rate (d-1), default: 0.47
  - (optional) D_L, Limit density (g dw)
- Concentration response (Toxicodynamics)
  - EC50_int, Internal concentration resulting in 50% effect (ug L-1)
  - E_max, Maximum inhibition (-), default: 1
  - b, Slope parameter (-)
- Internal concentration (Toxicokinetics)
  - P, Permeability (cm d-1)
  - r_A_DW, Area per dry-weight ratio (cm2 g-1), default: 1000
  - r_FW_DW, Fresh weight per dry weight ratio (-), default: 16.7
  - r_FW_V, Fresh weight density (g cm-3), default: 1
  - r_DW_TSL, Dry weight per total shoot length ratio (g dw cm-1)
  - K_pw, Partitioning coefficient plant:water (-), default: 1
  - k_met, Metabolisation rate (d-1), default: 0

Environmental factors

None.

Parameter boundaries

Default values for parameter boundaries are set for all parameters by expert judgement, for calibration purposes. Values can be modified using set_bounds().

Simulation output

Simulation results will contain the state variables biomass (BM) and mass of internal toxicant (M_int).

It is possible to amend the output of simulate() with additional model quantities that are not state variables, for e.g. debugging purposes or to analyze model behavior. To enable or disable additional outputs, use the optional argument nout of simulate(). As an example, set nout=2 to enable reporting of total shoot length (TSL) and internal concentration (C_int). Set nout=0 to disable additional outputs. The default is nout=1.

The available output levels are as follows:

- nout >= 1: TSL, total shoot length (cm)
- nout >= 2: C_int, internal concentration (ug L-1)
- nout >= 3: f_photo, photosynthesis dependency function (-)
- nout >= 4: C_int_unb, unbound internal concentration (ug L-1)
- nout >= 5: C_ext, external concentration (ug L-1)
- nout >= 6: dBM, biomass derivative (g dw d-1)
- nout >= 7: dM_int, mass of toxicant in plants derivative (ng d-1)

Solver settings

The arguments to ODE solver `deSolve::ode()` control how model equations are numerically integrated. The settings influence stability of the numerical integration scheme as well as numerical precision of model outputs. Generally, the default settings as defined by *deSolve* are used, but all *deSolve* settings can be modified in *cvasi* workflows by the user, if needed. Please refer to e.g. `simulate()` on how to pass arguments to *deSolve* in *cvasi* workflows.

Some default settings of *deSolve* were adapted for this model by expert judgement to enable precise, but also computationally efficient, simulations for most model parameters. These settings can be modified by the user, if needed:

- `hmax = 0.1` Maximum step length in time suitable for most simulations.

Effects

Supported effect endpoints include *BM* (biomass) and *r* (average growth rate during simulation). The effect on biomass is calculated from the last state of a simulation. Be aware that endpoint *r* is incompatible with biomass transfers.

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

References

Witt et al., submitted

Klein J., Cedergreen N., Heine S., Reichenberger S., Rendal C., Schmitt W., Hommen U., 2021: *Refined description of the Lemna TKTD growth model based on Schmitt et al. (2013) - equation system and default parameters*. Report of the working group *Lemna* of the SETAC Europe Interest Group Effect Modeling. Version 1.1, uploaded on 09 May 2022. https://www.setac.org/group/effect-modeling.html

See Also

Macrophyte-models, Lemna-models, Transferable, Scenarios

Other macrophyte models: `Lemna_SETAC()`, `Lemna_Schmitt()`, `Macrophyte-models`

Magma-class *Magma scenario class*

Description

This entry documents the class definition used for Magma-type scenarios. For details regarding the model, please refer to the Magma model manual.

Slots

`growth_model` *character*, selects the growth model, such as *exponential* or *logistic* growth.

See Also

Macrophyte-models

metsulfuron *Lemna model fitted to* metsulfuron *effect data*

Description

Data set for the parametrisation of a mechanistic combined toxicokinetic-toxicodynamic (TK/TD) and growth model for the aquatic macrophytes Lemna spp. as published by Schmitt *et al.* (2013). The growth model was parameterised by Schmitt et al. based on these data while toxicokinetic and toxicodynamic parameters were determined by fitting the model using substance specific effect data of the herbicide *metsulfuron-methyl*.

Usage

```
metsulfuron
```

Format

An object of class `LemnaSchmitt` of length 1.

Details

The scenario is based on the Lemna_Schmitt model.

References

Schmitt W., Bruns E., Dollinger M., and Sowig P., 2013: *Mechanistic TK/TD-model simulating the effect of growth inhibitors on Lemna populations*. Ecol Model 255, pp. 1-10. doi:10.1016/j.ecolmodel.2013.01.017

See Also

Lemna-models, schmitt2013

`minnow_it` *A fitted GUTS-RED-IT scenario of the fathead minnow*

Description

The example scenario consists of a fitted GUTS-RED-IT model and a constant exposure series. Model parameters were derived from a typical four-day acute fish toxicity study of the *fathead minnow* by Geiger *et al.* (1988). The study evaluated the effect of *chlorpyrifos* concentrations in water on survival of *fathead minnows*.

Usage

```
minnow_it
```

Format

An object of class `GutsRedIt` of length 1.

Details

The toxicity dataset used for parameter calibration is also referred to as *GUTS Ring-test dataset C* by EFSA (2018). Fitted parameters were estimated using the *morse* package.

The exposure series of the example scenario is a constant concentration of 1.0 µmol/L over a period of four days with a daily time step.

Source

https://mosaic.univ-lyon1.fr/guts

References

Geiger D.L., Call D.J., and Brooke L.T., 1988: *Acute toxicities of organic chemicals to fathead minnows (Pimephales promelas): Volume IV*, pp. 195-197. University of Wisconsin-Superior, Center for Lake Superior Environmental Studies. ISBN 9780961496838.

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

GUTS-RED-models

Examples

```
# Print scenario parameters
minnow_it

# Run the example scenario
minnow_it %>% simulate()
```

`minnow_sd` *A fitted GUTS-RED-SD scenario of the fathead minnow*

Description

The example scenario consists of a fitted GUTS-RED-SD model and a constant exposure series. Model parameters were derived from a typical four-day acute fish toxicity study of the *fathead minnow* by Geiger *et al.* (1988). The study evaluated the effect of *chlorpyrifos* concentrations in water on survival of *fathead minnows*.

Usage

```
minnow_sd
```

Format

An object of class `GutsRedSd` of length 1.

Details

The toxicity dataset used for parameter calibration is also referred to as *GUTS Ring-test dataset C* by EFSA (2018). Fitted parameters were estimated using the *morse* package.

The exposure series of the example scenario is a constant concentration of 1.0 µmol/L over a period of four days with a daily time step.

Source

https://mosaic.univ-lyon1.fr/guts

References

Geiger D.L., Call D.J., and Brooke L.T., 1988: *Acute toxicities of organic chemicals to fathead minnows (Pimephales promelas): Volume IV*, pp. 195-197. University of Wisconsin-Superior, Center for Lake Superior Environmental Studies. ISBN 9780961496838.

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

GUTS-RED-models

Examples

```
# Print scenario parameters
minnow_sd

# Run the example scenario
minnow_sd %>% simulate()
```

`no_exposure` *Zero exposure*

Description

Creates an ExposureSeries with zero concentration. When setting the zero exposure, pay attention not to accidentally reset the output times of your scenario as the zero exposure series contains only a single time point. See the examples.

Usage

```
no_exposure()
```

Value

an S4 object of type ExposureSeries

See Also

`set_noexposure()`

Examples

```
# Set exposure to zero, but keep the original output times
minnow_it %>%
  set_noexposure() %>%
  simulate()
```

num_info *Print information about numerical solver result*

Description

[Experimental]

Usage

```
num_info(obj)

## Default S3 method:
num_info(obj)

## S3 method for class 'cvasi_simulate'
num_info(obj)

## S3 method for class 'cvasi_fit'
num_info(obj)
```

Arguments

obj	Return value of `simulate()`

Details

Prints information on the status of a return value from `simulate()`, e.g. if it was successful and what, if any, issues occurred. Also provides tips on solving frequently occurring issues.

The function requires certain metadata which is created by `deSolve::ode()` and is passed through to the result of `simulate()`. The metadata may be lost if the `data.frame` returned by `simulate()` is converted or cast to other types.

See Also

`diagnostics()`

Examples

```
# A simulation without any issues
minnow_it %>% simulate() %>% num_info()

# A simulation which terminated early due to the solver
# taking too many numerical steps
rs <- suppressWarnings(minnow_it %>% simulate(hmax=1e-80))
num_info(rs)

# Print deSolve diagnostics for additional information
diagnostics(rs)
```

`parameter_set` *Set of model parameters*

Description

Set of model parameters

Usage

```
parameter_set(model, param = list(), tag = NA_character_)
```

Arguments

`model`	`character`, a string containing a model name, e.g. `"GUTS-RED-IT"`
`param`	named `list` of model parameters
`tag`	`character`, an optional identifier

Value

an S4 object of type ParameterSet

Slots

`model` `character`, a string containing a model name, e.g. `"GUTS-RED-IT"`

`tag` `character`, an optional identifier

`param` named `list` of model parameters

Examples

```
# create a parameter set and assign it
ps <- parameter_set("GUTS-RED-IT", list(kd=0.12, hb=0.3))
GUTS_RED_IT() %>% set_param(ps)

# multiple scenarios can be modified at once
c(GUTS_RED_IT(), GUTS_RED_IT()) %>%
  set_param(ps)

# model names must match, otherwise an error will be raised
try(GUTS_RED_SD() %>% set_param(ps))
```

pll_debug *Disable parallelization for debugging*

Description

In certain cases it might be beneficial to disable parallel execution of e.g. effect profile calculations. By disabling, all processes run sequentially and instantly pass messages to the console which would be delayed during parallel processing. This makes it easier to pinpoint problems within the data or algorithm.

Usage

```
pll_debug(state = TRUE)
```

Arguments

state	logical, if TRUE then parallelization is disabled

Value

no return value

plot *S3 plotting functions*

Description

These functions overload base::plot() to provide simple plotting routines to display various time-series and scenario objects.

Usage

```
## S3 method for class 'cvasi_drc'
plot(x, y, scale_x = c("auto", "log10", "none"), ...)

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

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

plot_lik_profile(x)
```

Arguments

x	object to plot
y	unused parameter
scale_x	character, controls how the x-axis is scaled. log10 for a log10-scaled axis, none for no scaling, and auto for automatic selection
...	unused parameters

Value

(*ggplot2*) plot object

Methods (by class)

- plot(cvasi_drc): Plot dose response curves
- plot(cvasi_simulate): Plot return value of simulate()
- plot(lik_profile): Plot likelihood profiles.

Functions

- plot_lik_profile(): Alias of plot.lik_profile() for backwards-compatibility.

---

plot.param_space *Plot profiled parameter space*

Description

The function provides bivariate parameter space plots indicating parameter draws (from the 95% confidence intervals per parameter obtained through likelihood profiling) that fall within the inner rim (in green, i.e. parameter sets which are not significantly different from the original, based on a chi-square test). The original parameter set is also indicated (in orange), and, if different from the original set, the best fit parameter set is indicated (in red).

Usage

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

plot_param_space(x)
```

Arguments

x	Return value of explore_space()
y	Unused argument
...	Unused arguments

Value

plot object

Functions

- `plot_param_space()`: Alias of `plot.param_space()` for backwards-compatibility.

`plot_epx` *Plot EPx values*

Description

[Experimental]

Usage

```
plot_epx(
  EPx_ts,
  exposure_ts,
  draw = TRUE,
  time_col = "time",
  conc_col = "conc",
  epx_x_title = "Start time",
  conc_y_title = "Exposure conc."
)
```

Arguments

`EPx_ts`	the result of `epx_mtw`, ie. a tibble with window.start, window.end, endpoint, level and EPx
`exposure_ts`	an exposure time series with columns for time 'time' and concentration 'conc'
`draw`	Should the whole plot be drawn? If FALSE the exposure plot and the EPx plot are returned as a list for later modification
`time_col`	the name of the time column in the exposure dataset
`conc_col`	the name of the concentration column in the exposure dataset
`epx_x_title`	title of the x-axis of the epx panel
`conc_y_title`	title of the y-axis of the concentration panel

Value

a grid of ggplots

Examples

```
ti <- 0:21
expo <- abs(0.01*ti + rnorm(length(ti), 0, 0.05))
exposure <- data.frame(time = ti, conc = expo)
metsulfuron_epx_mtw <- metsulfuron %>%
set_exposure(exposure) %>%
epx_mtw(level = 10, factor_cutoff = 1000)
metsulfuron_epx_mtw
plot_epx(EPx_ts = metsulfuron_epx_mtw,
exposure_ts = exposure, conc_y_title = "env. concentration [µg/L]")
```

---

plot_ppc *Creates a PPC plot for a single dataset*

Description

[Experimental]

Usage

```
plot_ppc(
  rs_mean,
  rs_range,
  col_number = 2,
  obs_mean = NULL,
  obs_full = NULL,
  xy_lim = NULL,
  study = NULL
)
```

Arguments

rs_mean	data.frame, model results best fit params
rs_range	data.frame, predictions (min, max from param.sample run)
col_number	column to plot, default = 2
obs_mean	data.frame, observations with means per treatment level
obs_full	data.frame, full data set including results for replicates
xy_lim	optional numeric, limits of x and y axis for plotting
study	optional string, name of study which can be used as key

Details

A sample of parameters representing the uncertainty within the dataset is passed to the function. All parameter combinations and exposure patterns are simulated and the range of predicted frond numbers is derived for a single study. The uncertainty is displayed by a Posterior Predictive Plot (PPC). The data (rs_mean, obs_mean and obs_full) must have the following format (col1 = time, col2 = data of interest, col3 = trial name). Data for uncertainties (rs_range) must have the format: col1 = time, col2 = lower boundaries, col3 = upper boundaries, col4 = trial. The user should take care of the input data and consider whether control data and data at time zero should be included in the model check.

Value

a ggplot2 plot object

`plot_ppc_combi` *Create PPC plot for one or more datasets*

Description

[Experimental]

Usage

```
plot_ppc_combi(table, xy_lim = NULL)
```

Arguments

`table`	`data.frame` containing return values of calls to `plot_ppc()`
`xy_lim`	optional `numeric`, limits of x and y axis for plotting

Details

The function expects a data.frame with five mandatory and one optional column. The mandatory columns are as follows:

- `pred`: mean of predictions e.g. frond number for lemna
- `max`: maximum of predictions
- `min`: minimum of predictions
- `obs`: observations
- `PPC`: color code The optional column is to be named `study` and contains a study identifier. If more than one study identifier is present in the table, individual studies will be plotted in different colors and a legend will be displayed. The function is called by plot_ppc where the column names are defined (see rs_ppc object).

Value

a ggplot2 plot object

plot_scenario *Creates a prediction plot for one effect scenario*

Description

[Deprecated]

Usage

```
plot_scenario(model_base, plot_col = 2, trial_number = NULL)
```

Arguments

model_base	effect scenario object with mean parameters
plot_col	output column which should be plotted, default = 2
trial_number	name for model run (if available tag is used)

Details

This function has been deprecated and replaced by the generic `plot()`.

Sometimes it is helpful if the user can plot results of one effect scenario. This is for instance the case for test simulations or predictions for one profile. This function runs the simulation for one effect scenario and plots the results. Function plots the time (column 1) and the predictions (column 2, can be changed by the user plot_col)

Value

plot of the results for one `effect scenario`

Examples

```
# Please use `plot()` instead
metsulfuron %>%
  simulate() %>%
  plot()
```

plot_sd *Creates plot of model results (uncertainties optional)*

Description

[Experimental]

Usage

```
plot_sd(
  model_base,
  treatments,
  rs_mean,
  rs_range = NULL,
  obs_mean = NULL,
  obs_full = NULL,
  x_breaks = NULL,
  y_lim = NULL,
  grid_labels = NULL,
  grid_ncol = 2,
  plot_col = 2,
  y_title = NULL,
  ...
)
```

Arguments

model_base	effect scenario object with mean parameters
treatments	treatments exposure levels as data frame
rs_mean	data.frame, model results best fit params
rs_range	data.frame, uncertainties as data frame
obs_mean	data.frame, observation data with means per treatment level
obs_full	data.frame, full set including results for replicates
x_breaks	optional vector of breaks of x-axis
y_lim	optional vector containing limits of y-axis
grid_labels	optional labels of grid headers
grid_ncol	optional number of grid columns
plot_col	output column which should be plotted
y_title	optional title of y-axis
...	any additional parameters

Details

All parameter combinations and exposure patterns are simulated and the mean of predictions is derived for a single study. The uncertainty is passed to the function due to computation time. Results are displayed by plotting the time series including the uncertainty interval. Observation data can be optionally displayed. Data should be provided in long format. Function plots the time (column 1) and the predictions (column 2, can be changed by the user plot_col)

Value

a ggplot2 plot object

Examples

```
set.seed(124)
exposure <- data.frame(
  time = 0:21,
  conc = rnorm(n = 22, mean = 0.1, sd = 0.06),
  trial = "T1"
)
forcings <- list(temp = 12, rad = 15000)
param <- list(EC50 = 0.3, b = 4.16, P_up = 0.0054)
inits <- list(BM = 0.0012, E = 1, M_int = 0)

scenario <- Lemna_Schmitt() %>%
  set_forcings(forcings) %>%
  set_param(param) %>%
  set_init(inits)

sim_result <- simulate_batch(
  model_base = scenario,
  treatments = exposure,
  param_sample = NULL
)

plot_sd(
  model_base = scenario,
  treatments = exposure,
  rs_mean = sim_result
)
```

---

pull_metadata *Pull metadata from scenarios*

Description

The method pulls available metadata from scenario objects and returns a table with additional columns. If the argument already was a `data.frame` object, the columns are appended. May overwrite existing columns of the same name.

Usage

```
pull_metadata(x, model = TRUE, exposure = TRUE)
```

Arguments

x	vector of scenarios or a data.frame containing a column scenario with EffectScenario objects
model	logical, if TRUE then model metadata is pulled
exposure	logical, if TRUE then exposure series metadata is pulled

Value

a data.frame

Examples

```
metsulfuron %>%
  pull_metadata()
```

---

ringtest_c *EFSA Ringtest Dataset C*

Description

This dataset is part of EFSA's *GUTS* ringtest to compare results from software implementations of GUTS-RED models (EFSA 2018). The ringtest focused on GUTS-RED-IT and GUTS-RED-SD models.

Usage

```
ringtest_c
```

Format

An object of class data.frame with 30 rows and 4 columns.

Details

Dataset C in this ringtest came from the work of Geiger et al. (1988) on effects of *Diazinon* on *fathead minnow*. The dataset contains four columns, identifying the control and treatment levels, the concentrations in these treatments, the timepoint of measurement and the number of survivors.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

Geiger DL, Call DJ, and Brooke LT, 1988: Acute Toxicities of Organic Chemicals to Fathead Minnow (*Pimephales promelas*), Vol IV. Center for Lake Superior Environmental Studies, University of Wisconsin-Superior, Superior, WI, USA.

https://cran.r-project.org/web/packages/GUTS/vignettes/ringTest.html

See Also

GUTS-RED-models

`ringtest_it` *EFSA Ringtest Dataset A SD*

Description

This dataset is part of EFSA's *GUTS* ringtest to compare results from software implementations of GUTS-RED models (EFSA 2018). The ringtest focused on GUTS-RED-IT and GUTS-RED-SD models.

Usage

```
ringtest_it
```

Format

An object of class `data.frame` with 42 rows and 4 columns.

Details

Dataset A IT for the GUTS ring test is a Hypothetical data set constructed with IT.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

https://cran.r-project.org/web/packages/GUTS/vignettes/ringTest.html

See Also

GUTS-RED-models

ringtest_sd *EFSA Ringtest Dataset A SD*

Description

This dataset is part of EFSA's *GUTS* ringtest to compare results from software implementations of GUTS-RED models (EFSA 2018). The ringtest focused on GUTS-RED-IT and GUTS-RED-SD models.

Usage

```
ringtest_sd
```

Format

An object of class `data.frame` with 42 rows and 4 columns.

Details

Dataset A SD for the GUTS ring test is a Hypothetical data set constructed with SD.

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

https://cran.r-project.org/web/packages/GUTS/vignettes/ringTest.html

See Also

GUTS-RED-models

rsubcapitata *A Weber scenario of algae exposed to isoproturon*

Description

The scenario recreates the conditions of algae species *R. subcapitata* exposed to isoproturon in a flow-through reactor experiment, see Weber et al. (2012) publication. Scenario parameters and exposure series were set according to reported values to represent conditions in reactor A, as depicted in Figure 4A in Weber et al.

Usage

```
rsubcapitata
```

Format

An object of class `AlgaeWeber` of length 1.

References

Weber D, Schaefer D, Dorgerloh M, Bruns E, Goerlitz G, Hammel K, Preuss TG and Ratte HT, 2012. Combination of a higher-tier flow-through system and population modeling to assess the effects of time-variable exposure of isoproturon on the green algae Desmodesmus subspictatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31, 899-908. doi:10.1002/etc.1765

See Also

Algae_Weber

`Scenarios` *Effect scenario classes*

Description

The `EffectScenario` class is the base for all of the basic scenario types and models. It contains slots for data and settings that are required by most models such as a vector of model parameters and a vector of initial states. For each particular model, the class's slots are filled with certain default or fixed values. Some models derive from this class and add slots to store additional data.

Details

Certain behaviors that are required to model complex processes cannot be represented by a single EffectScenario. As an example, the parameters of a scenario are generally fixed during the simulated time period. In order to represent a change in parameter values, the original scenario would need to split into two scenarios *A* and *B* which differ by parameter values and simulated time period. By combining these scenarios to a *scenario sequence*, the sequence would be treated as a single, complex scenario. See `sequence()` for more information.

Parameters:

Most parameters are represented by numerical types but other types are possible depending on model. Please refer to the model description which parameters are required and in which unit. Some or all parameters may be required to start a simulation. If required parameters are missing, simulation will fail with an error message.

Initial state:

The *initial state* represents the starting values of state variables when starting a simulation. A scenario's default initial state may be insufficient to get sensible results. It is advisable to set an initial state explicitly when creating a new scenario, see `set_init()`.

In theory, a scenario's state variables can be renamed by modifying the names of the initial state vector. However, this is strongly discouraged as this will affect other routines such as `effect()` and `epx()` and may render results useless.

Exposure:

Exposure refers to the concentration of toxicant an organism is exposed to. In case of aquatic organisms, this would commonly be the concentration of a toxicant in water. Other interpretations are possible depending on model assumptions.

Exposure time-series are generally represented by a `data.frame` containing two columns. The first column representing time, the second representing the exposure level. The ordering of columns is mandatory. The column names are essentially irrelevant but sensible names may help documenting the scenario and its data. The rows must be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant exposure. Exposure time-series are set to a scenario using `set_exposure()`.

Handling time-series is a costly task for the ODE solver due to consistency checks and interpolation between time steps. How the solver interpolates the time-series can be controlled by certain arguments to functions such as `simulate()` and `effect()`. Please refer to `simulate()` for a brief overview and deSolve::forcings for a detailed description.

Exposure time-series should be kept as short as possible and as complex as needed for optimal computational efficiency.

Environmental forcings:

Forcings generally refer to model parameters that change over time as part of an external function such as environmental temperature and exposure levels. Due to the importance of exposure in regulatory assessments, this R package explicitly distinguishes between environmental forcings and exposure. However, the same restrictions and features apply to both of them.

Forcing time-series are handled the same way as exposure time-series, i.e. they are represented by a `data.frame` containing two columns. The first column representing time, the second representing the parameter that is a function of time. The ordering of columns is mandatory. The rows must be ordered chronologically. Forcings time-series are set using `set_forcings()`. Please refer to the *Exposure* section for more information on how time-series are handled.

Output times:

A scenario's simulated time period is defined by its minimum and maximum output time. Simulation results will only be returned for the defined output times even though the ODE solver may use smaller time steps between output times. Output times can be explicitly set using `set_times()`. The number and distance of output times may have influence on the precision of simulation results and numerical stability, cf. `simulate()`.

Be aware that `set_exposure()` will overwrite previously defined output times if not requested otherwise.

Effects:

Generally, all state variables can be used as effect endpoints but models may provide additional endpoints. Use `set_endpoints()` to enable or disable endpoints for a scenario.

Some scenarios or models require control runs to calculate effects under exposure. Generally, control simulations will run automatically where needed.

Moving exposure windows:

The time frame relevant for effects may be much shorter than the assessed exposure time-series for certain organisms. This fact can be represented by moving exposure windows which divide a long time period in a number of consecutive windows of the same length. Each window is simulated

individually and effects are calculated. By default, methods such as `effect()` will only return the maximum effect of all considered windows but detailed results can be presented on demand.

To use moving exposure windows, the exposure time-series must be regular, i.e. must have an equidistant step length in time. The length of the window is defined as the number of time steps of the exposure time-series. As an example, assume the time-series has daily granularity and a moving window of seven days length is required. In this case, the moving window must have a length of seven (7) time steps. If the exposure time-series had hourly granularity, the same window would need to have a length of 168 (=7*24) time steps. Please refer to `set_window()` for details.

Slots

`name character`, unique model name

`tag character`, an optional identifier

`param list` of parameter key-value pairs

`param.bounds` named `list` of parameter boundaries

`param.req character` vector of required parameters

`forcings list` of `data.frames` representing forcing time-series

`forcings.req character` vector or required model forcings data, e.g. temperature

`init list` of initial model states

`times numeric` vector of output times, beginning and end also define the simulated period

`endpoints character` vector of endpoints to calculate results for

`exposure data.frame` with two columns representing an exposure time-series

`control list` of named numerical vectors, contains the control values for all relevant moving windows

`control.req logical`, if `TRUE` then control values are required to calculate effects

`window.length numeric`, maximum length of the simulated period, if `window.length` is shorter than the exposure pattern, then all possible exposure sub-patterns are evaluated for effect calculation. This is also referred to as a moving window approach.

`window.interval numeric`, interval determining distance between moving windows during effect calculation. First window starts at first time point in exposure pattern.

See Also

Other scenario-related: `Transferable`

`schmitt2013` *A Lemna effect study data set with multiple treatment levels*

Description

Data was collected from Schmitt et al. (2013). The study observed the number of Lemna fronds during the study period of 14 days. The organisms were exposed to multiple concentrations of the sulfonyl urea herbicide *metsulfuron-methyl*.

Usage

```
schmitt2013
```

Format

An object of class `data.frame` with 56 rows and 4 columns.

Details

The number of Lemna fronds was recorded at the beginning of the experiment (t=0) and after 3, 5, 7, 10, 12, and 14 days. The exposure to *metsulfuron-methyl* was only present during the first seven days, which was followed by a recovery period of another seven days without exposure to the substance.

The tabular dataset consists of four columns:

- First column: time (days)
- Second column: observed number of fronds (-)
- Third column: trial id (-)
- Fourth column: metsulfuron-methyl concentration (ug/L)

An example scenario with parameters fitted to the experimental data set as conducted by Schmitt et al. is available as `metsulfuron`.

References

Schmitt W., Bruns E., Dollinger M., and Sowig P., 2013: *Mechanistic TK/TD-model simulating the effect of growth inhibitors on Lemna populations*. Ecol Model 255, pp. 1-10. doi:10.1016/j.ecolmodel.2013.01.017

See Also

Lemna-models, metsulfuron

`sequence` *Sequence of scenarios*

Description

Scenario sequences can be used to implement changes in model parameters over time, which otherwise would remain constant for the duration of a simulation. A sequence of scenarios is treated as a single scenario and each scenario is simulated one after the other. If scenario `n` in a sequence was simulated, scenario `n+1` will start off in the model state where `n` had ended.

Usage

```
sequence(seq, breaks = NULL)
```

Arguments

`seq`	list of scenario objects
`breaks`	optional vector of *numerics*, scenarios' output times will be modified so that one scenario ends at the break and the next one begins

Details

Sequences are generally treated the same as scenarios: Sequences can be simulated, as well as effects and EPx can be derived.

Requirements:

All scenarios in a sequence must fulfill the following requirements:

- All scenarios must have identical state variables
- The *output times* of all scenarios must represent a continuous time series without gaps or overlaps

Using the `breaks` argument, the function can split up the scenarios' output times at the given break points. The break points must be within the interval defined by the superset of all output times in the sequence.

Value

an S4 object of type ScenarioSequence

See Also

sequence.extract

Examples

```
# Create a scenario with background mortality only
scen1 <- minnow_it %>%
  set_noexposure() %>%
  set_times(0:10)
# Modify a scenario parameter, e.g. set background mortality to zero
scen2 <- scen1 %>% set_param(c(hb=0))

# Create a sequence of scenarios, scenario #1 will be simulated for the
# time period [0, 4], and #2 for [4, 10]
sq <- sequence(list(scen1, scen2), breaks=c(4))

# Simulate the sequence: the mortality stops after t=4.0, due to scenario #2
# being simulated after t=4.0, which disabled the background mortality
simulate(sq)
# Effect endpoints can also be calculated
effect(sq)
```

---

sequence.extract *Extract and replace elements of a sequence*

Description

The array accessor generics allow extracting and replacing scenarios within am existing sequence. `[` and `[[` work identical to

Usage

```
## S4 method for signature 'ScenarioSequence,numeric,missing,missing'
x[i]

## S4 method for signature 'ScenarioSequence,numeric'
x[[i]]

## S4 replacement method for signature 'ScenarioSequence,numeric,missing,EffectScenario'
x[[i, j]] <- value

## S4 method for signature 'ScenarioSequence'
length(x)
```

Arguments

x	sequence
i	index of elements to extract or replace
j	*not used*
value	new scenario

Value

various

Functions

- `x[i`: Returns a list of scenarios from the sequence.
- `x[[i`: Returns a single scenario from the sequence.
- `` `[[`(x = ScenarioSequence, i = numeric, j = missing) <- value ``: Replaces a single scenario in the sequence.
- `length(ScenarioSequence)`: Returns the number of scenarios in the sequence.

Examples

```
# create a sequence
seq <- sequence(list(minnow_it, minnow_it), breaks=3)

seq[1]       # first element, as a list of scenarios
seq[c(1)]    # the same
seq[c(1, 2)] # both elements as a list of scenarios
seq[[1]]     # first element as a scenario

# replacing single elements
seq[[1]] <- minnow_sd %>% set_times(1:3)
```

---

set_bounds *Set boundaries of model parameters*

Description

Modifies the boundaries of model parameters for one or more scenario or caliset objects.

Usage

```
set_bounds(x, bounds)

## S4 method for signature 'EffectScenario,list'
set_bounds(x, bounds)

## S4 method for signature 'CalibrationSet,list'
set_bounds(x, bounds)

## S4 method for signature 'list,list'
set_bounds(x, bounds)

## S4 method for signature 'ScenarioSequence,list'
set_bounds(x, bounds)
```

Arguments

x	vector of scenario or caliset objects
bounds	named list of numerical vectors, where the first level lists the parameters by name, and the second level lists the lower and upper boundary

Value

scenario or caliset with modified parameter boundaries

Examples

```
metsulfuron %>%
   set_bounds(list(k_phot_max = c(0, 30),
                   k_resp = c(0, 10)))
```

---

set_endpoints *Set effect endpoints*

Description

Effect endpoints calculated by functions such as `effect()` and `epx()` can be enabled and disabled. If an endpoint is not required for an assessment, it should be disabled for reasons of computational efficiency. Please refer to the model description for a list of available endpoints.

Usage

```
set_endpoints(x, endpoints)
```

Arguments

x	vector of `EffectScenario` objects
endpoints	`character` vector of endpoint names

Value

Modified `EffectScenario` objects

Examples

```
# Only enable reproduction (R) endpoint for americamysis scenario
americamysis %>%
  set_endpoints("R") %>%
  effect()

# Enable endpoints length (L) and reproduction (R)
americamysis %>%
  set_endpoints(c("L","R")) %>%
  effect()
```

set_exposure *Set exposure time-series*

Description

Exposure refers to the toxicant concentration an organism is exposed to. In case of aquatic organisms, this would commonly be the concentration of a toxicant in water. Other interpretations are possible depending on model assumptions.

Usage

```
set_exposure(scenarios, series, ...)

## S4 method for signature 'ANY,ANY'
set_exposure(scenarios, series, ...)

## S4 method for signature 'EffectScenario,data.frame'
set_exposure(scenarios, series, ...)

## S4 method for signature 'EffectScenario,ExposureSeries'
set_exposure(scenarios, series, reset_times = TRUE)

## S4 method for signature 'EffectScenario,NoExposureSeries'
set_exposure(scenarios, series, ...)

## S4 method for signature 'EffectScenario,list'
set_exposure(scenarios, series, ...)

## S4 method for signature 'list,list'
set_exposure(scenarios, series, ...)

## S4 method for signature 'list,ANY'
set_exposure(scenarios, series, ...)

## S4 method for signature 'ScenarioSequence,ANY'
set_exposure(scenarios, series, ...)
```

Arguments

scenarios	vector of scenarios
series	vector of ExposureSeries objects or a single data.frame
...	additional arguments
reset_times	logical, if TRUE, the exposure time-series' time points will be set as output times. Defaults to TRUE

Details

Exposure time-series are generally represented by a `data.frame` containing two columns. The first column for time, the second representing the exposure level. The ordering of columns is mandatory. The column names are non-relevant but sensible names may help documenting the scenario and its data. The `data.frame`'s rows must be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant exposure.

For convenience, a time-series with zero exposure can be set using `set_noexposure()`.

Computational efficiency:

Handling time-series is a costly task for the ODE solver due to consistency checks and interpolation between time steps. How the solver interpolates the time-series can be controlled by optional arguments to functions such as `simulate()` and `effect()`. Please refer to `simulate()` for a brief overview and deSolve::forcings for a detailed description.

Exposure time-series should be kept as short as possible and as complex as needed for optimal computational efficiency.

Output times:

By default, the exposure time-series' time points will also be used as output times of the scenario. Any output times previously set by `set_times()` will be lost. If this behavior is undesired, set the function argument `reset_times=FALSE`.

Multiple exposure series and scenarios:

The functions supports modifying multiple scenarios at once: by calling it with lists of scenario and ExposureSeries objects. The cartesian product of all scenarios and exposure series will be returned, iff the parameter `expand = TRUE` is set.

As an example for the *expand* mode, two scenarios `A` and `B` and one exposure series `g` will result in two scenarios `Ag` and `Bg`, both using exposure series `g`. Two scenarios `A` and `B` as well as two exposure seres `g` and `h` will result in four scenarios `Ag`,`Ah`,`Bg`, and `Bh`.

Value

`list` of `EffectScenario` objects

Examples

```
# Set a data.frame as exposure series
df <- data.frame(time=c(0, 1, 2, 3), conc=c(1, 1, 0, 0))
Lemna_Schmitt() %>% set_exposure(df)

# Create and set an ExposureSeries object
es1 <- ExposureSeries(df)
Lemna_Schmitt() %>% set_exposure(es1)

# By default, the time points of the exposure series will also be used as
# as output times. To avoid overriding existing output times, set reset_times=FALSE
Lemna_Schmitt() %>%
  set_times(0:10) %>%
  set_exposure(es1, reset_times=FALSE)
```

```
# Setting two series with one function call, creates two scenarios
es2 <- ExposureSeries(data.frame(time=5:10, conc=1))
Lemna_Schmitt() %>% set_exposure(c(es1, es2))
```

`set_forcings` *Set time-dependent parameters*

Description

Parameters which change their value over time are referred to as *forcings*. If and what parameters can vary over time depends on the model in question. In many cases, *forcings* represent time-series of environmental properties.

Usage

```
set_forcings(x, ...)

## S4 method for signature 'EffectScenario'
set_forcings(x, ...)

## S4 method for signature 'list'
set_forcings(x, ...)
```

Arguments

`x`	(vector of) scenario objects
`...`	named argument list to set as forcings

Details

Forcing time-series are always represented by a `data.frame` containing two columns. The first column representing time, the second representing the parameter that is a function of time. The ordering of columns is mandatory. The column names are essentially irrelevant but may help documenting the scenario and its data. The rows must be ordered chronologically. A time-series can consist of only a single row; in this case it will represent constant conditions.

Handling forcing time-series is a costly task for the ODE solver due to consistency checks and interpolation between timesteps. How the solver interpolates the forcing time-series can be controlled by certain arguments to functions such as `simulate()` and `effect()`. Please refer to `simulate()` for a brief overview and deSolve::forcings for a detailed description.

Forcing time-series should be kept as short as possible and as complex as needed for optimal computational efficiency.

Value

Modified scenarios

Examples

```
# constant values will be automatically converted to a data.frame
Lemna_Schmitt() %>% set_forcings(temp=20) -> lemna
lemna@forcings

# setting multiple forcings at once
df <- data.frame(t=0:14, temp=rnorm(15, mean=20)) # random temperature series
Lemna_Schmitt() %>% set_forcings(temp=df, rad=15000) -> lemna
lemna@forcings

# forcings can also be supplied as a named list
Lemna_Schmitt() %>% set_forcings(list(temp=20, rad=15000)) -> lemna
lemna@forcings
```

---

set_init *Set initial state*

Description

The *initial state* represents the starting values of a scenario's state variables when starting a simulation. A scenario's default initial state may be insufficient to get sensible results.

Usage

```
set_init(x, init)

## S4 method for signature 'EffectScenario'
set_init(x, init)

## S4 method for signature 'ScenarioSequence'
set_init(x, init)

## S4 method for signature 'vector'
set_init(x, init)
```

Arguments

x	vector of `EffectScenario` objects
init	named numeric vector

Details

In theory, a scenarios's state variables can be renamed by modifying the names of the initial state vector. However, this is strongly discouraged as this will affect other routines such as `effect()` and `epx()` and may render results useless.

Value

modified `EffectScenario` objects

Examples

```
# Set initial biomass to 1.0
metsulfuron %>% set_init(c(BM=1.0)) %>% simulate()
```

`set_mode_of_action` *Set mode of action*

Description

Updates the model parameter `MoA` to a certain value

Usage

```
set_mode_of_action(x, code)

set_moa(x, code)
```

Arguments

x	vector of scenarios
code	a code for a mode of action, refer to model description for details

Value

modified scenarios

Functions

- `set_moa()`: Shorthand version

Examples

```
# Set MoA=8, i.e. hazard during oogenesis
americamysis %>%
  set_mode_of_action(8) %>%
  effect(method="ode45")

# alternative approach using the parameter directly
americamysis %>%
  set_param(c(MoA=8)) %>%
  effect(method="ode45")
```

set_noexposure *Set zero exposure*

Description

The scenarios current exposure is replaced by a constant exposure time-series of value zero(`0.0`). Output times are unaffected.

Usage

```
set_noexposure(x)
```

Arguments

x	vector of scenarios

Value

vector of scenarios

Examples

```
# Derive effect size in sample scenario without toxicant exposure
minnow_it %>%
  set_noexposure() %>%
  effect()
```

set_param *Set scenario parameters*

Description

Modifies the parameters of one or more scenario objects.

Usage

```
set_param(x, param)

## S4 method for signature 'EffectScenario,vector'
set_param(x, param)

## S4 method for signature 'EffectScenario,ParameterSet'
set_param(x, param)

## S4 method for signature 'list,ParameterSet'
set_param(x, param)
```

```
## S4 method for signature 'list,vector'
set_param(x, param)

## S4 method for signature 'ScenarioSequence,ANY'
set_param(x, param)

## S4 method for signature 'CalibrationSet,ANY'
set_param(x, param)
```

Arguments

x	object(s) to modify
param	named numeric vector with parameter names and value OR a list of parameter_set objects

Details

Most parameters are represented by numerical types but other types are possible depending on model. Please refer to the model description which parameters are required and in which unit. Some or all parameters may be required to start a simulation. If required parameters are missing, simulation will fail with an error message.

Value

Vector of modified objects

Examples

```
Lemna_Schmitt() %>% set_param(c(Emax=1,EC50=0.12))
```

---

set_tag *Set a tag*

Description

Sets the user-defined, custom tag of a scenario. Tags can be helpful to quickly distinguish scenarios by e.g. a user-specified string.

Usage

```
set_tag(x, tag)
```

Arguments

x	(vector of) scenario objects
tag	(vector of) character or any other object

Details

The function supports vectorized inputs. If arguments `x` and `tag` are vectors, then tags are assigned to scenarios on a 1:1 basis. If the length of both vectors do not match, an error is raised.

Value

(vector of) modified scenarios

See Also

`get_tag()`

Examples

```
# set a custom tag
myscenario <- GUTS_RED_SD() %>% set_tag("My Custom Tag")

# returns `My Custom Tag`
get_tag(myscenario)

# the tag also appears in the scenario overview
myscenario
```

---

set_times *Set output times*

Description

Minimum and maximum output times define the simulated period for a scenario. Simulation results will be returned for each output time, see `simulate()`.

Usage

```
set_times(x, times)
```

Arguments

`x`	vector of scenarios
`times`	`numerical` vector

Details

Be aware that output times may be modified by `set_exposure()`. Precision of simulation results may be influenced by chosen output times, see `simulate()` for more information.

Value

Vector of modified scenarios

See Also

simulate(), get_times()

Examples

```
# Set simulated period to [2,4] with output intervals of length 1
minnow_it %>% set_times(c(2,3,4))

# Decrease output interval length to 0.1
minnow_it %>% set_times(seq(2, 4, 0.1))
```

---

set_transfer *Set transfer events*

Description

A *transfer* refers to an event where a certain amount of biomass is moved to a new medium after a period of time. Effectively, this resets the scenario's state variable representing biomass and re-scales all state variables which are correlated with biomass, such as adsorbed chemical mass. This feature replicates a procedure occurring e.g. in *Lemna* effect studies and may be necessary to recreate study results.

Usage

```
set_transfer(x, interval, times, biomass, scaled_comp)

## S4 method for signature 'ANY'
set_transfer(x, interval, times, biomass, scaled_comp)

## S4 method for signature 'Transferable'
set_transfer(x, interval, times, biomass, scaled_comp)

set_notransfer(x)
```

Arguments

x	vector of EffectScenario objects
interval	optional numeric, interval in time units of the scenario, set to -1 to disable transfers.
times	optional numeric vector of time points where transfers occur
biomass	optional numeric vector, amount of biomass that is being transferred at each transfer
scaled_comp	optional character vector of affected compartments that are scaled according to new biomass levels

Details

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

Transferred biomass:

At each transfer, a defined amount of biomass is transferred to a new medium. This is modeled by interrupting the simulation at a transfer time point, modifying the biomass level `BM`, and scaling affected compartments according to new biomass levels. Scaling of compartments depending on biomass, such as *internal toxicant mass*, is necessary to correctly reflect mass balances and concentrations over time.

Transferred biomass is set using the `biomass` parameter. Is is either a single numerical value in which case the same biomass level is set at each transfer. Or it is a vector of numerical values with the same length as the `times` parameter in which case a custom biomass level can be set for each transfer. Multiple biomass levels can only be set in conjunction with custom transfer time points.

Some scenario types define default values for transferred biomass based on common study set ups.

Regular and custom transfer time points:

Transfers can occur either in regular intervals of time or at selected, custom time points. For regular intervals, the parameter `interval` is set to a single numeric value which has the same unit as the scenario's time dimension. As an example: if a scenario uses the unit of *days* for time, the transfer interval is also specified in *days*:

Transfers occurring at custom time points are set by passing a numerical vector to the parameter `times`. The time points' units must match with the unit of time in the scenario. A custom transfer time point **must not occur at the starting time point of a simulation**.

Affected compartments:

Some compartments depend on biomass to correctly reflect mass balances and concentrations over time, such as *internal toxicant mass*. These compartments need to be scaled linearly to reflect the change in biomass levels. The parameter `scaled_comp` accepts a character vector of compartment names which are scaled at each transfer. This parameter should only be used with custom, user-defined models. If no compartment needs to be scaled, set or use the default value of `character(0)`.

Value

Modified scenario objects

Functions

- `set_notransfer()`: Disable biomass transfers

See Also

Lemna-models

Examples

```
# Simulate biomass transfer of 50 *g/m²* at a regular interval of 7 *days*
metsulfuron %>%
  set_transfer(interval=7, biomass=50) %>%
  simulate()

# Simulate irregular biomass transfers occuring at days 5, 10, and 12
metsulfuron %>%
  set_transfer(times=c(5, 10, 12), biomass=50) %>%
  simulate()

# Simulate irregular transfers with changing amounts of transferred biomass
metsulfuron %>%
  set_transfer(times=c(5, 10, 12), biomass=c(50, 20, 10)) %>%
  simulate()

# Disable all biomass transfers
metsulfuron %>%
  set_notransfer() %>%
  simulate()
```

---

set_window *Set window length*

Description

Exposure windows are defined as a period of time at the scale of the exposure series. As an example: if an exposure series has an hourly time step, a window length of 24 will consider the exposure within 24 hours intervals for effect calculation. The same applies for the window interval, i.e. the period between considered exposure windows. Set `length=-1` to disable moving windows.

Usage

```
set_window(x, length, interval)

set_nowindow(x)
```

Arguments

`x`	vector of `EffectScenario` objects
`length`	`numeric`, length of exposure window to consider for effect calculation, set `length=-1` to disable moving windows
`interval`	`numeric`, interval between considered exposure windows

Value

modified `EffectScenario` objects

Functions

- `set_nowindow()`: Disable moving windows

Examples

```
# Calculate the maximum effect for all windows of 10 days length
metsulfuron %>%
  set_window(length=10, interval=1) %>%
  effect()

# Disable moving exposure windows
metsulfuron %>%
  set_nowindow() %>%
  effect()
```

`simulate` *Simulate an effect scenario*

Description

The supplied `EffectScenario` is passed on to the ODE solver for numerical integration. Internally, `simulate()` is split up into several functions dedicated to particular models, e.g. one for GUTS and one for Lemna type models. The package will take care of using the correct function for each model when `simulate()` is called.

Usage

```
simulate(x, ...)

## S4 method for signature 'EffectScenario'
simulate(x, ...)

## S4 method for signature 'Transferable'
simulate(x, ...)

## S4 method for signature 'ScenarioSequence'
simulate(x, ...)

## S4 method for signature 'SimulationBatch'
simulate(x, ...)
```

Arguments

`x`	scenario to simulate
`...`	additional parameters passed on to ODE solver

Details

Simulation results are returned as a time-series for each state variable. Some models provide additional properties describing the model state, e.g. the internal concentration of a toxicant within the organism. Refer to the respective scenario for more information.

Additional arguments to `simulate()` will be passed on to `deSolve::ode()` which enables control of the numerical integration parameters.

Output times and windows:

The minimum and maximum of given time points define the simulated period. However, the simulation can also be limited to a subset of time points by enabling a moving exposure window, see `set_window()`.

Results will be returned for each output time point. Precision of the numeric solver may be affected by chosen output times in certain cases. Hence, small deviations in results should be expected if different output times are set. This effect can be mitigated by either defining are sufficiently small time step for the solver using argument `hmax` or by decreasing the error tolerances `atol` and `rtol`. These arguments are passed to the solver, see e.g. `deSolve::lsoda()` for details.

Optional output variables:

Some models support adding intermediary model variables to the return value of `simulate()`. Analyzing the additional outputs may be helpful to understand model behavior and support finding potential issues in model parameterization.

Optional outputs are enabled by setting the parameter `nout` to a value greater than zero. If `nout` is set to `n`, then the first `n` optional output columns will be returned along the normal simulation result.

Which optional outputs are available depends on the model/scenario at hand. Please refer to the model documentation for details. As an example, the GUTS-RED-IT model supports adding the external toxicant concentration to the output by setting `nout=1`:

```
minnow_it %>% simulate(nout=1)
```

Numerical precision and stability:

Each model was assigned a default ODE solver which handles most of the occurring inputs well. In most cases, this will be an explicit numerical scheme of the Runge-Kutta family with variable step width. For certain extreme parameters settings, such as very high uptake/permeability of the contaminant or exposure series which represent step functions, the numerical approximation might deteriorate and yield invalid results. In this case try to decrease the allowed max step width by setting the argument `hmax` with various values. Start with `hmax=1` and decrease the value by orders of 10. It is not possible or desirable to reduce `hmax` to extremely small values, as the ODE solver will require more CPU time and simulation will become inefficient.

Oftentimes, it will be computational more efficient to adapt the solver's error tolerances `atol` and `rtol` than reducing the step width `hmax` to achieve stable numerics. Start by decreasing deSolve's default values by orders of ten until the simulation yields acceptable results, see e.g. `deSolve::lsoda()` for more information on error tolerances.

As an alternative to adapting solver parameters, it might be worthwhile to try other numerical schemes which can handle stiff ODEs, such as Radau, LSODA, or LSODES. To change solvers, set the `method` argument. To select e.g. the Radau scheme, set `method="radau"`. For LSODA, set `method="lsoda"`. Radau performs better than LSODA in some cases, as the latter method can return biologically nonsensical results without raising an error. See `deSolve::ode()` for details on available ODE solvers.

Value

A `data.frame` with the time-series of simulation results

Examples

```
# base R syntax
simulate(minnow_sd)
# tidy syntax with the same result
minnow_sd %>% simulate()

# Extend the simulated time frame to the interval [0, 10]
minnow_sd %>%
  set_times(seq(0, 10)) %>%
  simulate()

# Use an alternative exposure profile, but keep the original output times
minnow_sd %>%
  set_exposure(data.frame(t=0, c=10), reset_times=FALSE) %>%
  simulate()

##
## Precision of results

# A large number of output times forces smaller solver time steps
minnow_it %>%
  set_times(seq(0, 10, 0.001)) %>%
  simulate() %>%
  tail()

# Defining only two output times allows the ODE solver to make larger steps
# in time during numerical integration. However, results can become
# imprecise.
minnow_long <- minnow_it %>% set_times(c(0, 10))
minnow_long %>% simulate()

# Numerical precision of results can be increased by limiting the solver's
# maximum step length in time using argument `hmax`.
minnow_long %>% simulate(hmax=0.005)

# A similar numerical precision can be achieved by switching to an alternative
# numerical integration scheme, such as the Radau scheme, without limiting
# the step length.
minnow_long %>% simulate(method="radau")

# Reducing the step length even further may increase numerical precision, but
# may exceed the solver's allowed number of integration steps per output interval.
# The following simulation will be aborted with a solver error:
try(
  minnow_long %>% simulate(hmax=0.001)
)

# However, the solver's maximum number of allowed steps can be increased,
```

```
# if needed, using the argument `maxsteps`:
minnow_long %>% simulate(hmax=0.001, maxsteps=10^5)
```

`simulate_batch` *Batch simulation using multiple exposure series*

Description

[Deprecated]

Usage

```
simulate_batch(model_base, treatments, param_sample = deprecated(), ...)
```

Arguments

`model_base`	effect scenario object with mean parameters
`treatments`	treatments exposure levels as data frame (time, conc, trial)
`param_sample`	*deprecated* parameter, no longer in use
`...`	additional parameters passed through to `simulate()`

Details

A convenience function to simulate a single base scenario with one or more exposure series. This aims at reproducing the setup and results of common effect studies.

A scenario contains only one exposure series. However, laboratory experiments commonly examine the effects of multiple exposure levels on a biological system. A batch simulation approach would involve running multiple simulations with varying exposure or treatment conditions. To illustrate, if the objective is to examine the impact of a substance on cell growth, the simulation model could be designed to replicate the cell growth dynamics under varying concentrations of the substance. Each simulation run would represent a specific exposure level, ranging from low to high concentrations of the chemical. To simulate such a laboratory experiment, the simulate_batch function can be used. All exposure series are saved in the treatment argument. The first column contains the time, the second column the concentration, and the third column the trial name (exposure level, e.g. 'T1', 'T2', 'T3').

Value

a `data.frame` with simulation results

Examples

```
t1 <- data.frame(time=0:10, conc=0, trial="control")  # 1st treatment level
t2 <- data.frame(time=0:10, conc=1, trial="T1")       # 2nd treatment level
treatments <- rbind(t1, t2)

metsulfuron %>%
  simulate_batch(treatments)
```

`solver` *Calls ODE solver for a particular model*

Description

Please refer to the *Modeling Howto* vignette on how to implement custom models by overloading the `solver` function.

Usage

```
solver(scenario, ...)

## S4 method for signature 'ANY'
solver(scenario, ...)

## S4 method for signature 'LemnaSetac'
solver(scenario, ...)

## S4 method for signature 'Magma'
solver(scenario, ...)

## S4 method for signature 'LemnaSchmitt'
solver(scenario, ...)

## S4 method for signature 'AlgaeWeber'
solver(scenario, method = "lsoda", hmax = 0.1, nout = 1, ...)

## S4 method for signature 'AlgaeTKTD'
solver(scenario, method = "lsoda", hmax = 0.1, ...)

## S4 method for signature 'AlgaeSimple'
solver(scenario, method = "lsoda", hmax = 0.1, ...)

## S4 method for signature 'DebAbj'
solver(scenario, ...)

## S4 method for signature 'DebTox'
solver(scenario, method = "ode45", ...)

## S4 method for signature 'DebDaphnia'
solver(scenario, ...)

## S4 method for signature 'GutsSd'
solver(scenario, ...)

## S4 method for signature 'GutsIt'
solver(scenario, ...)
```

```
## S4 method for signature 'GutsRedSd'
solver(scenario, ...)

## S4 method for signature 'GutsRedIt'
solver(scenario, ...)
```

Arguments

scenario	scenario object
...	additional parameters passed on to deSolve::ode()
method	string, selects the numerical solver used by deSolve::ode()
hmax	numeric, sets the maximum step length in time, see deSolve::ode()
nout	numeric, the number of additional output variables returned by the model, see vignette("compiledCode", "deSolve") for details. If and which output variables are available depends on the scenario type, please refer to the documentation of the model in question.

Details

Some solvers may set reasonable default values for e.g. maximum step length in time (hmax), but not all do. Please check the model documentation for details.

Value

data.frame with simulation results

Methods (by class)

- solver(ANY): Default solver, raises an error
- solver(LemnaSetac): Numerically integrates Lemna_SETAC scenarios
- solver(Magma): Numerically integrates Magma scenarios
- solver(LemnaSchmitt): Numerically integrates Lemna_Schmitt scenarios
- solver(AlgaeWeber): numerically integrates Algae_Weber scenarios
- solver(AlgaeTKTD): numerically integrates Algae_TKTD scenarios
- solver(AlgaeSimple): numerically integrates Algae_Simple scenarios
- solver(DebAbj): Numerically integrates DEB_abj scenarios
- solver(DebTox): Numerically integrates *DEBtox* scenarios
- solver(DebDaphnia): (deprecated) Numerically integrates *DEBtox_Daphnia* scenarios
- solver(GutsSd): Numerically integrates GUTS-SD scenarios
- solver(GutsIt): Numerically integrates GUTS-IT scenarios
- solver(GutsRedSd): Numerically integrates GUTS-RED-SD scenarios
- solver(GutsRedIt): Numerically integrates GUTS-RED-IT scenarios

`survival` *Survival rate*

Description

[Deprecated] Derives the survival rate of individuals for *Reduced GUTS* models. Function was replaced by output of `simulate()` and will be removed in a later version.

Usage

```
survival(scenario, ...)
```

Arguments

`scenario`	an `EffectScenario` to simulate
`...`	additional parameters passed on to `simulate()`

Details

The survival rate describes the survival probability at each time point. The function simulates the *GUTS* scenario and appends a column `survival` to the simulation result. A value of one (`1.0`) denotes that all individuals survive. A value of zero (`0.0`) denotes that no individuals survived.

Only available for *Reduced GUTS* models, see GUTS-RED-models. The equations were described by EFSA (2018).

Value

a `data.frame` containing simulation results

References

EFSA PPR Panel (EFSA Panel on Plant Protection Products and their Residues), Ockleford C, Adriaanse P, Berny P, et al., 2018: *Scientific Opinion on the state of the art of Toxicokinetic/Toxicodynamic (TKTD) effect models for regulatory risk assessment of pesticides for aquatic organisms*. EFSA Journal 2018; 16(8):5377, 188 pp. doi:10.2903/j.efsa.2018.5377

See Also

GUTS-RED-models

Examples

```
# calculate survival rate
minnow_it %>% survival()

# plot survival over time based on a random exposure profile
minnow_sd %>%
  set_exposure(data.frame(t=1:100, c=runif(100)*10)) %>%
```

```
  survival() -> df
plot(df$time, df$survival, "l")
```

---

td2cs *Create* calisets *from* tox_data

Description

This is a convenience function which eases the creation of calisets from a base scenario and a tox_data object. The scenario will be used as is. If exposure series are defined by the `tox_data` object, then these will be assigned to the scenario(s) accordingly.

Usage

```
td2cs(scenario, data, output_var = NULL)
```

Arguments

`scenario`	a base scenario
`data`	return value of `tox_data()`
`output_var`	optional *character*, rename observed data column to this value

Value

list of calisets

Examples

```
# Import trial data from Schmitt et al. (2013), including exposure
mydata <- tox_data(schmitt2013)

# Example trial contained in data set: trial 'T0.32'
mydata@data[["T0.32"]]        # observed quantities
mydata@exposure[["T0.32"]]    # associated exposure series

# Create list of calisets from full data set
lst <- td2cs(Lemna_SETAC(), mydata)

# Example caliset representing conditions of trial 'T0.32'
lst[[2]]
```

`tox_data` *Prepare ecotox study data for fitting*

Description

Takes ecotox study data in *long-form* tabular format and prepares it for parameter fitting. It supports extracting (optional) exposure concentration from tabular data (useful for e.g. studies of acute toxicity) or exposure series can also provided as individual time-series.

Usage

```
tox_data(data, exposure = NULL)
```

Arguments

`data`	a `data.frame` with at least two and at most four columns; the first column must represent time, the second an observed quantity, the optional third a trial or treatment ID, and the optional fourth the concentration during the experiment
`exposure`	an optional named list; names must correspond to trial IDs used for the `data` argument; values can be numeric constants, data.frames, or an ExposureSeries object

Value

a `ToxData` object

Tabular format

The long-form tabular data must have at least two and at most four columns. The position of the columns define what they represent, the column names are ignored:

- First column: time
- Second column: observed quantity, e.g. number of individuals
- (optional) Third column: Trial or treatment ID
- (optional) Fourth column: Concentration

The first two columns, time and observed quantity, must always be present. The third column, trial ID, is used to split the table by treatment so that trials can later be handled individually. The fourth column, concentration, can be used to also define the exposure level during the experiments.

Explicit exposure series

As an alternative to defining concentrations along observed data, exposure can also be passed as a list of exposure levels and series with argument `exposure`. It can be used to provide exposure series for each trial. The following object types are supported to define exposure:

- Numerical constants

- Tabular data, e.g. `data.frames`
- exposure series objects

If exposure is constant over time, exposure can be defined using a single constant value. More complex exposure time-series can be defined using e.g. `data.frames`. Tabular data must have two columns with the first column representing time and the second column representing exposure/concentrations.

Examples

```
library(dplyr)

mydata <- schmitt2013 %>% tox_data()
```

`Transferable` *Biomass transfer class*

Description

By inheriting from class `Transferable`, a scenario's behavior can be extended to support transfer and reset of biomass at dedicated points during simulation.

Slots

`transfer.times numeric`, vector of custom time points at which transfers occur, e.g. `c(2,5,14)`

`transfer.interval numeric`, length of regular interval until biomass transfer to new medium, regular transfers always occur relative to time point zero

`transfer.biomass numeric`, amount of biomass transferred to new medium

`transfer.comp.biomass character` state variable which describes biomass

`transfer.comp.scaled character` vector of state variable which will be scaled 1:1 when biomass is modified, e.g. internal toxicant mass

Biomass transfer

Models supporting biomass transfer can be instructed to move a fixed amount of biomass to a new medium after a period of time. This feature replicates a procedure occurring in e.g. *Lemna* effect studies and may be necessary to recreate study results.

The biomass transfer feature assumes that always a fixed amount of biomass is transferred. Transfers can occur at any fixed point in time or in regular intervals. During a transfer, the biomass is reset to the transferred amount and additional compartments can be scaled 1:1 accordingly, to e.g. reflect the change in internal toxicant mass when biomass is modified. Transfer settings can be modified using `set_transfer()`.

If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that if transfers are defined using the `interval` argument, the transfers will always occur relative to time point zero (`t = 0`). As an example, setting a regular transfer of seven days, `interval = 7`, will result at transfers occurring at time points which are integer multiplicates of seven, such as `t=0`, `t=7`, `t=14` and so forth. The starting and end times of a scenario do not influece **when** a regular transfer occurs, only **if** it occurs.

See Also

set_transfer()

Other scenario-related: Scenarios

Examples

```
# Simulation without biomass transfers
metsulfuron %>%
  set_noexposure() %>%
  set_notransfer() %>%
  simulate()

# With biomass transfer every 7 days, biomass is reset to 50 *g/m²* on transfer
metsulfuron %>%
  set_noexposure() %>%
  set_transfer(interval=7, biomass=50) %>%
  simulate()
```

Index