Package: cvasi (via r-universe)

September 6, 2024

Type Package

Title Calibration, Validation, and Simulation of TKTD Models

Version 1.1.3

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

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE, roclets = c(``collate", ``namespace", ``rd", ``roxyglobals::global_roclet"))

Config/testthat/edition 3

Collate 'cache_controls.R' 'class-CalibrationSet.R' 'class-ExposureSeries.R' 'class-EffectScenario.R' 'calibrate.R' 'class-Transferable.R' 'class-Algae.R' 'class-Deb.R' 'class-GutsRed.R' 'class-Lemna.R' 'class-Myriophyllum.R' 'class-ScenarioSequence.R' 'class-parameter_set.R' 'data.R' 'dose_response.R' 'effect.R' 'epx.R' 'explore_space.R' 'get.R' 'globals.R' 'has.R' 'helper_functions.r' 'import.R' 'import_toxswa.R' 'is.R' 'lik_profile.R' 'log.R' 'morse.R' 'package.R' 'pll.R' 'plotting.r' 'pull.R' 'sequence.R' 'set.R' 'set_bounds.R' 'set_exposure.R' 'set_forcings.R' 'set_init.R'

'set_param.R' 'set_transfer.R' 'set_window.R' 'show.R'
'simulate.R' 'solver.R' 'survival.R' 'utils-pipe.R'
'utils-stats.R'

Suggests future, testthat, knitr, rmarkdown, roxyglobals

Depends R (>= 3.5.0)

VignetteBuilder knitr

Config/roxyglobals/filename globals.R

Config/roxyglobals/unique FALSE

Repository https://cvasi-tktd.r-universe.dev

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

RemoteRef HEAD

RemoteSha cef9e02fc6b620fa9cfa9a7bd768d461777767b2

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()` Simple 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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

References

Weber D, Schaeffer 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

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 scenarios: `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`, `Myriophyllum-models`, `Scenarios`, `Transferable`

`Algae_Simple` *Algae model with exponential growth but without additional forcings*

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 AlgaeSimpleScenario

State variables

The model has two state variables:

- `A`, Biomass (ug fresh wt/mL, cells/mL *10^4)
- `Dw`, only used if 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 (ug 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)
  - `kD`, dominant rate constant of toxicant in aquatic environments (d-1)
  - `scaled`, 0 = no internal scaled damage / 1 = yes (-)

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 in case of scaled damage the damage concentration (`Dw`). The derivatives are also available as additional output.

- `nout >= 2`
  - `dA`, biomass derivative (µg)
  - `dDw`, damage concentration derivative (ug L-1)

References

Weber D, Schaeffer 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 subspicatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31, 899-908. doi:10.1002/etc.1765

See Also

Scenarios, Transferable

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

`Algae_TKTD` *Algae model with exponential growth, forcings (P, I) and 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 AlgaeTKTDScenario

State variables

The model has four state variables:

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

Model parameters

- Growth model
  - `mu_max`, Maximum growth rate (d-1)
  - `Q_min`, Minimum intracellular P (ug P/ug fresh wt)
  - `Q_max`, Maximum intracellular P (ug P/ug fresh wt)
  - `v_max`, Maximum P-uptake rate at non-limited growth (ug P/ug 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)
- Concentration response (Toxicodynamics)
  - `EC_50`, Effect concentration of 50% inhibition of growth rate (ug 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

Besides exposure events (Cw), the *Algae* model requires two environmental properties as time-series input: Irradiance (`I`, uE/m²/s) and temperature (`T_act`, deg C). 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. The input format for all forcings is a list of the data frames.

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`). The derivatives are also available as additional output.

- `nout >= 4`
    - `dA`, biomass derivative (µg)
    - `dQ`, internal phosphorous derivative (mg P/ug fresh wt)
    - `dP`, external phosphorous derivative (mg P L-1)
    - `dDw`, damage concentration derivative (ug L-1)

References

Weber D, Schaeffer 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 subspicatatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31, 899-908. doi:10.1002/etc.1765

See Also

Scenarios, Transferable

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

`Algae_Weber` *Algae model with exponential growth and forcings (I, T)*

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 submodel describes the effects of growth-inhibiting substances through a corresponding reduction in the photosynthesis rate on the basis of internal concentrations. (the implementation of Weber et al. (2012) is followed where units differ with EFSA)

Usage

```
Algae_Weber()
```

Value

an S4 object of type AlgaeWeberScenario

State variables

The model has four state variables:

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

Model parameters

- Growth model
  - `mu_max`, Maximum growth rate (d-1)
  - `Q_min`, Minimum intracellular P (ug P/ug fresh wt)
  - `Q_max`, Maximum intracellular P (ug P/ug fresh wt)
  - `v_max`, Maximum P-uptake rate at non-limited growth (ug P/ug 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 (ug/L)
  - `b`, slope of concentration effect curve at EC_50 (-)
- External concentration (Toxicokinetics)
  - `k`, Degradation rate of toxicant in aquatic environments (d-1)

Forcings

Besides exposure events (C_in), the *Algae* model requires three environmental properties as time-series input: Irradiance (`I`, uE/m²/s) and temperature (`T_act`, deg C). 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. The input format for all forcings is a list of the data frames.

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`). The derivatives are also available as additional output.

- `nout >= 4`
  - `dA`, biomass derivative (µg)
  - `dQ`, internal phosphorous derivative (mg P/ug fresh wt)
  - `dP`, external phosphorous derivative (mg P L-1)
  - `dC`, external concentration derivative (ug L-1)

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.

References

Weber D, Schaeffer 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

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()`

`cache_controls` *Cache control simulations*

Description

Cache control simulations

Usage

```
cache_controls(x, windows, skipZeroExposure = FALSE, ...)
```

Arguments

`x`	vector of scenario objects
`windows`	`list` of window tuples
`skipZeroExposure`	`logical`, if `TRUE`, windows with zero exposure will not be included in calculations
`...`	additional parameters passed on to `effect()`

Value

Modified scenario objects

`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

```
calibrate(x, ...)

## S4 method for signature 'EffectScenario'
calibrate(
  x,
  par,
  data,
  endpoint = deprecated(),
  output,
  by,
  metric_fun = deprecated(),
  err_fun,
  as_tibble = deprecated(),
  catch_errors = deprecated(),
  verbose = FALSE,
  ...
)

## S4 method for signature 'CalibrationSet'
calibrate(x, par, output, err_fun, verbose = FALSE, ...)

## S4 method for signature 'list'
calibrate(
  x,
  par,
  endpoint = deprecated(),
  output,
  metric_fun = deprecated(),
  metric_total = deprecated(),
  err_fun,
  as_tibble = deprecated(),
  catch_errors = deprecated(),
  verbose = FALSE,
  ...
)
```

Arguments

`x`	either a single scenario or a list of CalibrationSet objects to be fitted
`...`	additional parameters passed on to `stats::optim()` and `simulate()`
`par`	named numeric vector with parameters to fit and their start values
`data`	`data.frame` with two or more columns with experimental data, 1st column must contain time points, the following columns may values which the scenario is fitted to.
`endpoint`	*deprecated* `character`, please use `output` instead
`output`	`character`, name of a single output column of `simulate()` to optimize on

`by`	optional `character`, groups and splits the experimental data into multiple distinct trials and datasets before fitting
`metric_fun`	*deprecated*, please use `err_fun` instead
`err_fun`	vectorized error function to calculate an error term that is minimized during optimization, must accept exactly two vectorized numeric arguments, defaults to sum of squared errors
`as_tibble`	*deprecated*, result can no longer be returned as a tibble
`catch_errors`	*deprecated*, simulation errors are always caught
`verbose`	`logical`, if `TRUE` then debug outputs are displayed during optimization
`metric_total`	*deprecated*

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 by summing the error of each *calibration set*.

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 dataset must contain a column that which matches with the contents of parameter `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 contain a column named `BM` which represents the observed biomass. 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:

By default, the total sum of squared errors is used as the target function which is minimized during fitting. A custom error function can be supplied by the user: The function must accept two numeric vectorized arguments and return a numeric of length one, i.e. the error value.

Example of a custom error function which returns the sum of absolute errors:

```
my_absolute_error <- function(observed, simulated) {
  sum(abs(observed - simulated))
}
```

When using *calibration sets*, the error term is calculated for each *calibration set* individually, the weighting factor is applied to the error of each set, and then all error terms are summed up.

Value

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

Methods (by class)

- `calibrate(EffectScenario)`: Fit single scenario using a dataset
- `calibrate(CalibrationSet)`: Fit using a CalibrationSet
- `calibrate(list)`: Fit using a list of CalibrationSet objects

Examples

```
library(dplyr)

# Get observed biomass during control experiment by Schmitt et al. (2013)
observed <- Schmitt2013 %>%
  filter(ID == "T0") %>%
  select(t, BM=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()

# Fit parameter 'k_phot_max' to observed biomass growth from experiment
calibrate(
  scenario,
  par=c(k_phot_max=1),
  data=observed,
  output="BM",
  method="Brent", # Brent is recommended for one-dimensional optimization
  lower=0,        # lower parameter boundary
  upper=0.5       # upper parameter boundary
) -> fit
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)
```

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.
`weight`	optional `numeric` weight to be applied to this dataset's error term during fitting, default values is 1.0.

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:

If more than one *calibration set* is used for fitting, then an optional weighting factor can be used to scale the error term of the affected *set*.

Value

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

Examples

```
library(dplyr)

# Get observed biomass during control experiment by Schmitt et al. (2013)
observed <- Schmitt2013 %>%
  filter(ID == "T0") %>%
  select(t, BM=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()

# 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
  lower=0,        # lower parameter boundary
  upper=0.5       # upper parameter boundary
```

```
) -> fit
fit$par
```

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

Description

Supported models:

- DEB_abj
- DEB_Daphnia

See Also

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

Other scenarios: `Algae-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`, `Myriophyllum-models`, `Scenarios`, `Transferable`

`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

See Also

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

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_exposure(no_exposure()) %>%
  set_times(0:10) %>%
  simulate()

# Print information about sample scenario 'americamysis'
americamysis

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

`DEB_Daphnia` *DEBtox Daphnia*

Description

Creates a *DEBtox Daphnia* scemarop.

Usage

```
DEB_Daphnia()
```

Details

State variables:

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

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

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

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)
- 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)
- TKTD 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))
- Global parameters
  - `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`, reference max length for scaling rate constants (mm)
  - `len`, switch to fit length 1) with shrinking, 2) without shrinking
  - `Tbp`, brood-pouch delay (d)
- `MoA`, mode of action switch (-)
- `FB`, feedback on damage dynamics switch (-)

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

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 *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()`.

Output variables:

The following intermediary model variables can be added to the model output on demand, see `simulate()` for details:

- `nout` >= `3`
  - `f`, actual f
  - `fR`, actual f considering starvation
  - `kd`, actual kd
- `nout` >= `6`
  - `s`, stress level for metabolic effects
  - `sA`, stress factor on assimilation/feeding
  - `h`, hazard rate for effects on survival
- `nout` >= `10`, damage feedbacks
  - `xu`, factor for surf:vol scaling uptake rate
  - `xe`, factor for surf:vol scaling elimination rate
  - `xG`, factor for growth dilution
  - `xR`, factor for losses with repro

Value

an S4 object of type DebDaphnia

See Also

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

`dmagna` *A DEBtox Daphnia magna scenario*

Description

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

Usage

```
dmagna
```

Format

An object of class `DebDaphnia` of length 1.

See Also

`DEB_Daphnia()`

`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`	`EffectScenario` 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 if this is feasible depends on the environment setup as defined by the `future` package
`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. `factor` 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, factor = 1, max_only = TRUE, ep_only = FALSE, marginal_effect, ...)
```

Arguments

`x`	vector of `EffectScenario` objects
`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
`...`	additional parameters passed on to `simulate()`

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.

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.

`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

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 (genereated 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,
  sample_size = 1000,
  max_runs = 30,
  nr_accept = 100,
  sample_factor = 1.2
)
```

Arguments

`x`	a list of CalibrationSet 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
`sample_size`	number of samples to draw from each parameter interval

`max_runs`	max number of times 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`	multiplicaton factor for sampling (95% interval * sample factor)

Value

a list containing a plot to explore the parameter space, and the dataframe supporting it

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

# exposure - control run
exp <- Schmitt2013 %>%
  filter(ID == "T0") %>%
  select(time=t, conc)

# observations - control run
obs <- Schmitt2013 %>%
  filter(ID == "T0") %>%
  select(t, BM=obs)

# 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 = 5, E = 1, M_int = 0)) %>%
  set_param(list(
    k_0 = 5E-5,
    a_k = 0.25,
    BM50 = 17600,
    mass_per_frond = 0.1
  )) %>%
  set_exposure(exp) %>%
  set_param(params) %>%
  set_bounds(bounds)
```

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

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

plot_param_space(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()`

`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 `LemnaSetacScenario` 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, uploaded on 22. Sept. 2021. `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 'GutsRedSd'
fx(scenario, ...)

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

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

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

## S4 method for signature 'Algae'
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(GutsRedSd)`: Effect at end of simulation of GUTS-RED-models
- `fx(GutsRedIt)`: Effect at end of simulation of GUTS-RED-models
- `fx(Lemna)`: Effect at end of simulation of Lemna-models
- `fx(Myriophyllum)`: Effect at end of simulation of Macrophyte-models
- `fx(Algae)`: Effect at end of simulation of Algae-models

`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_tag` *Get scenario tag*

Description

Returns the user-defined, custom tag of a scenario, if available. Tags can be helpful to quickly distinguish scenarios of the same model type. The function supports vectorized inputs.

Usage

```
get_tag(x)
```

Arguments

`x`	(vector of) scenarios or `parameter_set` objects

Value

vector of `character`

See Also

`set_tag()`

Examples

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

# update or set a tag
myscenario <- minnow_it %>% set_tag("My Custom Tag")
# returns `My Custom Tag`
get_tag(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 scenarios: `Algae-models`, `DEB-models`, `Lemna-models`, `Macrophyte-models`, `Myriophyllum-models`, `Scenarios`, `Transferable`

`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`.

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()`.

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()`

`import_exposure_text` *Import exposure profiles from text files*

Description

Read several exposure profiles at their corresponding paths at 'pathtofiles'. The exposure profiles are read for each file that are character seperated text files in a table format. The expected table format includes the columns `time` and `conc` for the concentration at each timestep.

Usage

```
import_exposure_text(pathtofiles, profileNames = NA, sep = "\t")
```

Arguments

`pathtofiles`	path to files
`profileNames`	the names of the profiles
`sep`	the field separator character

Value

list of exposure profiles

`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

Value

a list of exposure profiles each with two columns (time and concentration)

`import_toxswa` *Read TOXSWA exposure profiles from out-files*

Description

Read several TOXSWA exposure profiles from out-files at their corresponding paths given by'pathtofiles'.

Usage

```
import_toxswa(pathtofiles, profileNames = NA)
```

Arguments

`pathtofiles`	paths to toxswa files that should be read
`profileNames`	a vector of names to be used for the loaded toxswa files. if "NA" the filename without extension will be used

Value

list of exposure profiles each with two columns (time and concentration)

`is_DEB` *Test if argument is a DEB model*

Description

Test if argument is a DEB model

Usage

```
is_DEB(x)
```

Arguments

`x`	vector of `EffectScenario` objects

Value

vector of `logical` values

`is_GUTS` *Test if argument is a GUTS model*

Description

Test if argument is a GUTS model

Usage

```
is_GUTS(x)

is_GUTS_IT(x)

is_GUTS_SD(x)
```

Arguments

`x`	vector of `EffectScenario` objects

Value

vector of `logical` values

Functions

- `is_GUTS_IT()`: Test if argument is a GUTS-IT model
- `is_GUTS_SD()`: Test if argument is a GUTS-IT model

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 model*

Description

Also returns `TRUE` for `LemnaThreshold` models

Usage

```
is_Lemna(x)
```

Arguments

`x`	vector of scenarios objects

Value

vector of logical values

See Also

`is_LemnaThreshold()`

`is_LemnaThreshold` *Test if argument is a LemnaThreshold model*

Description

Test if argument is a LemnaThreshold model

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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

See Also

Macrophyte-models

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

Other scenarios: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Macrophyte-models`, `Myriophyllum-models`, `Scenarios`, `Transferable`

`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 LemnaSchmittScenario

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 Lemna model requires two environmental properties as time-series input: global radiation (`rad`, kJ/m2.d) and temperature (`temp`, deg C). 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)
- `nout=8`, growth and TK/TD
  - `BM_fresh`, fresh weight biomass (g_fw/m2)
  - `k_photo_eff`, current photosynthesis rate (1/d)
  - `k_resp_eff`, current respiration rate (1/d)
  - `f_Eff`, toxic effect factor (-)
  - `P_up_eff`, current permeability for uptake (cm/d)
- `nout=11`, environmental factors
  - `actConc`, current toxicant concentration in surrounding medium (ug/L)
  - `actTemp`, current environmental temperature (deg C)
  - `actRad`, current environmental radiation (kJ/m2.d)
- `nout=14`, derivatives
  - `d BM/dt`, current change in state variable BM
  - `d E/dt`, current change in effect
  - `d M_int/dt`, current change in 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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

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`, `Myrio()`, `Myrio_log()`

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

Description

The model was described and published by the SETAC Europe Interest Group Effect Modeling (Klein et al. 2022). 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 LemnaSetacScenario

State variables

The model has two state variables:

- `BM`, Biomass (g dw m-2)
- `M_int`, Mass of toxicant in plant population (mass per m2, e.g. ug m-2)

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 m-2)
  - `BM_min`, Reservoir for biomass recovery, (g dw m-2)
- 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 m-2)
- 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

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

- `tmp`, temperature (°C)
- `irr`, irradiance (kJ m-2 d-1)
- `P`, Phosphorus concentration (mg P L-1)
- `N`, Nitrogen concentration (mg N 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 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 (-)
- `nout` >= `4`
  - `f_loss`, respiration dependency function (-)
  - `f_photo`, photosynthesis dependency function (-)
- `nout` >= `10`
  - `fT_photo`, temperature response of photosynthesis (-)
  - `fI_photo`, irradiance response of photosynthesis (-)
  - `fP_photo`, phosphorus response of photosynthesis (-)
  - `fN_photo`, nitrogen response of photosynthesis (-)
  - `fBM_photo`, density response of photosynthesis (-)
  - `fCint_photo`, concentration response of photosynthesis (-)
- `nout` >= `16`
  - `C_int_unb`, unbound internal concentration (mass per volume)
  - `C_ext`, external concentration (mass per volume)

  - `Tmp`, temperature (deg C)
  - `Irr`, irradiance (kJ m-2 d-1)
  - `Phs`, Phosphorus concentration (mg P L-1)
  - `Ntr`, Nitrogen concentration (mg N L-1)
- `nout >= 18`
  - `dBM`, biomass derivative (g dw m-2 d-1)
  - `dM_int`, mass of toxicant in plants derivative (mass per m2 d-1)

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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

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, uploaded on 22. Sept. 2021. 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`, `Myrio()`, `Myrio_log()`

`lik_profile` *Likelihood profiling*

Description

The aim of the function is 2-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 (using [profile_par())]. 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 algoritm takes something small).

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).

[profile_par())]: R:profile_par()) [log_lik()]: R:log_lik()

Usage

```
lik_profile(
  x,
  par,
  output,
  data = NULL,
  bounds = NULL,
  refit = TRUE,
  type = c("coarse", "fine"),
  break_prof = FALSE,
  ...
)
```

Arguments

`x`	either a single scenario or a list of CalibrationSet 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
`break_prof`	if 'TRUE' (default), stop the profiling if a better optimum is located
`...`	additional parameters passed on to `stats::optim()` and `calibrate()`

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)

# exposure - control run
exp <- Schmitt2013 %>%
  filter(ID == "T0") %>%
  select(time=t, conc)

# observations - control run
obs <- Schmitt2013 %>%
  filter(ID == "T0") %>%
   select(t, BM=obs)

# update metsulfuron
myscenario <- metsulfuron %>%
  set_param(c(k_phot_fix = TRUE,Emax = 1)) %>%
  set_init(c(BM = 12)) %>%
  set_exposure(exp)

fit <- calibrate(
  x = myscenario,
  par = c(k_phot_max = 1),
  data = obs,
  output = "BM",
  lower=0,
  upper=1,
  method="Brent"
```

```
)

# Likelihood profiling

res <- lik_profile(
  x = myscenario,
  data = obs,
  output = "BM",
  par = fit$par,
  pars_bound = list(
    k_phot_max = list(0, 30)
  ),
  refit = FALSE,
  type = "fine",
  method = "Brent"
)
# plot
plot_lik_profile(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 (considering a normal distribution around the prediction for each datapoint)

Usage

```
log_lik(npars, obs, pred)
```

Arguments

`npars`	named numeric vector of parameters that the model was calibrated on
`obs`	numeric vector of observed values
`pred`	numeric vector of predicted values

Value

the log likelihood value

Examples

```
# 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(
  npars = length(pars),
  obs = obs,
  pred = pred
)
```

`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
- Myriophyllum

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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

See Also

Scenarios

Other macrophyte models: `Lemna_SETAC()`, `Lemna_Schmitt()`, `Myrio()`, `Myrio_log()`

Other scenarios: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Myriophyllum-models`, `Scenarios`, `Transferable`

`metsulfuron` *Lemna data published by Schmitt (2013)*

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 calibrating the model using substance specific effect data of metsulfuron-methyl.

Usage

```
metsulfuron
```

Format

An object of class `LemnaSchmittScenario` of length 1.

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

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

Description

The scenario consists of a parameterized GUTS-RED-IT model and a constant exposure series. Model parameters were derived from an acute fish toxicity study of the fathead minnow and chlorpyrifos by Geiger *et al.* (1988). The dataset is also referred to as *GUTS Ring-test dataset C* by EFSA (2018). Fitted parameters were estimated using the morse package.

Usage

```
minnow_it
```

Format

An object of class `GutsRedIt` of length 1.

Details

The background mortality rate (`hb`) was set to zero.

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

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

Description

The scenario consists of a parameterized GUTS-RED-SD model and a constant exposure series. Model parameters were derived from an acute fish toxicity study of the fathead minnow and chlorpyrifos by Geiger *et al.* (1988). The dataset is also referred to as *GUTS Ring-test dataset C* by EFSA (2018). Fitted parameters were estimated using the morse package.

Usage

```
minnow_sd
```

Format

An object of class `GutsRedSd` of length 1.

Details

The background mortality rate (`hb`) was set to zero.

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

morse *Import* `morse` *model parameters*

Description

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

Usage

```
morse(
  file,
  find.SD = TRUE,
  find.IT = TRUE,
  reset.hb = TRUE,
  params = c("estim", "all"),
  mcmc.size
)
```

Arguments

`file`	Path to .RData file
`find.SD`	a logical value. If `TRUE`, it will try to find fitted parameters of the GUTS_SD model
`find.IT`	a logical value. If `TRUE`, it will try to find fitted parameters of the GUTS_IT model
`reset.hb`	a logical value. If `TRUE`, the background hazard rate `hb` is set to zero
`params`	`character`, if `"estim"` is selected then only the fitted parameter set for each model is returned, else for `"auto"` all parameter sets in the MCM chains are returned
`mcmc.size`	optional `integer`, sets the maximum number of imported parameter sets per MCMC, all parameters are imported by default

Value

vector of `parameter_set` objects

Examples

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

# import parameters for a specific model
try(morse("path/to/morse_fit.RData", find.IT=TRUE, find.SD=FALSE))

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

`Myrio` *Myriophyllum model with exponential growth*

Description

The *Myriophyllum* model is derived from the *Lemna* TKTD model by Klein *et al.* (2021). The Myriophyllum model is mathematically equivalent to the Tier 2C version of the *Lemna* model by Klein *et al.* (2021), cf. `Lemna_SETAC()`. Recommended settings for Tier 2C are `k_photo_fixed=TRUE` and `k_resp=0` (Klein *et al.* 2021). 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).

Usage

```
Myrio()
```

Value

an S4 object of type MyrioExpScenario

State variables

The model has two state variables:

- `BM`, Biomass (g dw m-2 for field studies or mg dw for lab)
- `M_int`, Mass of toxicant in plant population (mass per m2, e.g. ug m-2)

Model parameters

- Growth model
  - `k_photo_max`, Maximum photosynthesis rate (d-1), default: `0.47`
- 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 (field) or mg (lab) 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 two additional columns besides state variables:

- `C_int`, internal concentration of toxicant (mass per volume)
- `TSL`, total shoot length (?)

The available output levels are as follows:

- `nout >= 1`
  - `C_int`, internal concentration (mass per volume)
- `nout >= 2`
  - `TSL`, total shoot length (?)
- `nout >= 3`
  - `f_photo`, photosynthesis dependency function (-)
- `nout >= 5`, growth and TK/TD
  - `C_int_unb`, unbound internal concentration (mass per volume)
  - `C_ext`, external concentration (mass per volume)
- `nout >= 7`, environmental factors
  - `dBM`, biomass derivative (g dw m-2 d-1)
  - `dM_int`, mass of toxicant in plants derivative (mass per m2 d-1)

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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

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, uploaded on 22. Sept. 2021. https://www.setac.org/group/effect-modeling.html

See Also

Macrophyte-models, Transferable, Scenarios

Other Myriophyllum models: `Myrio_log`(), `Myriophyllum-models`

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

`Myriophyllum-models` *Myriophyllum models*

Description

Supported models:

- `Myrio()`, with exponential growth
- `Myrio_log()`, with logistic growth

See Also

Lemna-models, Transferable

Other Myriophyllum models: `Myrio`(), `Myrio_log`()

Other scenarios: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`, `Scenarios`, `Transferable`

`Myrio_log` *Myriophyllum model with logistic growth*

Description

The *Myriophyllum* model is derived from the *Lemna* TKTD model by Klein *et al.* (2021). `Myrio_log()` modifies the `Myrio()` model to feature logistic growth, i.e. control growth is described by the differential equation `d BM/dt = k_photo_max*BM*(1 - BM/BM_L)` where `BM_L` is the carrying capacity.

Usage

```
Myrio_log()
```

Value

an S4 object of type MyrioLogScenario

Model parameters

- Growth model
  - `k_photo_max`, Maximum photosynthesis rate (d-1), default: `0.47`
  - `BM_L`, Carrying capacity (g dw m-2)
- 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 (?)
  - `K_pw`, Partitioning coefficient plant:water (-), default: `1`
  - `k_met`, Metabolisation rate (d-1), default: `0`

State variables

The model has two state variables:

- `BM`, Biomass (g dw m-2 for field studies or mg dw for lab)
- `M_int`, Mass of toxicant in plant population (mass per m2, e.g. ug m-2)

Environmental factors

None.

Simulation output

Simulation results will contain two additional columns besides state variables:

- `C_int`, internal concentration of toxicant (mass per volume)
- `TSL`, total shoot length (?)

The available output levels are as follows:

- `nout` >= 1
  - `C_int`, internal concentration (mass per volume)
- `nout` >= 2
  - `TSL`, total shoot length (?)
- `nout` >= 3
  - `f_photo`, photosynthesis dependency function (-)
- `nout` >= 5, growth and TK/TD
  - `C_int_unb`, unbound internal concentration (mass per volume)
  - `C_ext`, external concentration (mass per volume)
- `nout` >= 7, environmental factors
  - `dBM`, biomass derivative (g dw m-2 d-1)
  - `dM_int`, mass of toxicant in plants derivative (mass per m2 d-1)

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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

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()`.

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, uploaded on 22. Sept. 2021. https://www.setac.org/group/effect-modeling.html

See Also

Transferable, Scenarios

Other Myriophyllum models: `Myrio()`, `Myriophyllum-models`

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

`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

```
# this will reset the output times of the sample scenario,
# simulate() will quit with an error
try(
  minnow_it %>%
    set_exposure(no_exposure()) %>%
    simulate()
)

# set zero exposure, but keep original output times
minnow_it %>%
  set_exposure(no_exposure(), reset_times=FALSE) %>%
  simulate()
```

`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 parameter_set

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_epx` *Plot EPx values*

Description

Plot EPx values

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 't' 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_lik_profile` *Plot likelihood profiles or all profiled parameters*

Description

The function provides a combined plot of the likelihood profiles of all parameters profiled.

Usage

```
plot_lik_profile(x)
```

Arguments

`x`	object of class lik_profile

Value

plots

`plot_param_space` *Plot likelihood profiles or all profiled parameters*

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

```
plot_param_space(x)
```

Arguments

`x`	object of class param_space

Value

plots

`plot_ppc` *Creates a PPC plot for a single dataset*

Description

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.

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

Value

a ggplot2 plot object

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

Description

The function expects a data.frame with four 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 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).

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

Value

a ggplot2 plot object

`plot_scenario` *Creates a prediction plot for one effect scenario*

Description

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)

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)

Value

plot of the results for one `effect scenario`

Examples

```
plot_scenario(metsulfuron)
```

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

Description

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)

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

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()
```

`Rsubcapitata` *An algae scenario*

Description

Data are from Weber 2012 publication.

Usage

```
Rsubcapitata
```

Format

An object of class `AlgaeTKTDScenario` of length 1.

References

Weber D, Schaeffer 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 subspicatus and Pseudokirchneriella subcapitata. Environmental Toxicology and Chemistry, 31, 899-908. doi:10.1002/etc.1765

See Also

Algae_TKTD

`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. However, when conducting a large number of repeated simulations, e.g. when calculating effect profiles (EPx values) or simulating moving exposure windows, it may be computational efficient to run control simulations only once and cache their results within the scenario. Please refer to `cache_controls()` for details.

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 scenarios: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`, `Myriophyllum-models`, `Transferable`

`Schmitt2013` *A Lemna data set with multiple treatment levels*

Description

Data are from Schmitt 2013 publication.

Usage

```
Schmitt2013
```

Format

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

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

`sequence` *Sequence of scenarios*

Description

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. Scenario sequences can be used to e.g. implement changes in model parameters over time.

Details

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

Limitations:

Only simulation of sequences are supported, at the moment. Effects and effect profiles (EPx values) cannot be derived, yet.

Value

an S4 object of type ScenarioSequence

Examples

```
# create two scenarios that need to be simulated one after the other
scen1 <- minnow_it %>% set_times(0:3)
scen2 <- minnow_it %>% set_times(3:6) %>% set_param(c(kd=0))

# create a sequence and assign scenarios
sq <- sequence(list(scen1, scen2))

# simulate the sequence

simulate(sq)
```

`set_bounds` *Set boundaries of model parameters*

Description

Modifies the boundaries of model parameters for one or more scenario or CalibrationSet 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)
```

Arguments

`x`	vector of scenario or CalibrationSet 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 CalibrationSet 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,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, ...)
```

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
Lemna_Schmitt() %>% set_exposure(data.frame(time=c(0, 1, 2, 3), conc=c(1, 1, 0, 0)))

# set one ExposureSeries
es1 <- ExposureSeries(data.frame(time=0, conc=0))
Lemna_Schmitt() %>% set_exposure(es1)

# set two ExposureSeries to create two scenarios
es2 <- ExposureSeries(data.frame(time=5:10, conc=1))
Lemna_Schmitt() %>% set_exposure(c(es1, es2))

# set one ExposureSeries without resetting existing output times
Lemna_Schmitt() %>%
  set_times(0:5) %>%
  set_exposure(es1, reset_times=FALSE)
```

`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 'vector'
set_init(x, init)

## S4 method for signature 'EffectScenario'
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 `EffectScenario` objects
`code`	a code for a mode of action

Value

modified `EffectScenario` objects

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 model parameters*

Description

Modifies the parameters of one or more `EffectScenario` objects.

Usage

```
set_param(x, param)

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

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

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

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

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

## S4 method for signature 'ScenarioSequence,parameter_set'
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 of the same model type.

Usage

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

Arguments

`x`	(vector of) `EffectScenario` objects
`tag`	vector of `character`

Value

(vector of) modified `EffectScenario` objects

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 `ExposureScenario` objects

See Also

`simulate()`

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 (*BM*) is moved to a new medium after a period of time. 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)
```

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

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

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 `EffectScenario` objects

See Also

`Lemna_Schmitt()`

Examples

```
# Simulate biomass transfer of 50 *g dw/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()
```

`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)
```

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

Examples

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

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 generally define the simulated period. Function argument `times` overrides settings of the scenario, i.e. time points set in `scenario@times`. However, the simulation can be limited to a subset of time points by enabling a moving exposure window, see `set_window()`.

Results will be returned for each 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.

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

```
minnow_sd %>% simulate() # tidy syntax
simulate(minnow_sd) # base R syntax

# Set new output times
minnow_sd %>% simulate(times=c(0,4))

# Modify simulated time frame
minnow_sd %>% simulate(times=c(0,10))

# Use an alternative exposure profile than defined in the scenario
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 %>% simulate(times=seq(0,1,0.001)) %>% tail()

# Defining only two output times allows for larger solver time steps
minnow_it %>% simulate(times=c(0,1))

# The difference between results can be reduced by limiting the solver's
# maximum step length
minnow_it %>% simulate(times=c(0,1), hmax=0.001)

# The same numerical precision can be achieved by switching to
# the Radau scheme
minnow_it %>% simulate(times=c(0,1), method="radau")

# A very small step length may reduce the difference even further but may
# also exceed the allowed number of steps per output interval. The following
# simulation will be aborted with a solver error:
try(
  minnow_it %>% simulate(times=c(0,1), hmax=0.0001)
)

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

```
minnow_it %>% simulate(times=c(0,1), hmax=0.0001, maxsteps=10^5)
```

`simulate_batch` *Batch simulation of several effect scenarios*

Description

An effect scenario contains only one exposure level. Consequently, the simulation of an effect scenario (e.g. metsulfuron %>% simulate will return the results for one exposure level only). However, in a laboratory experiment examining the effects of different 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').

Usage

```
simulate_batch(model_base, treatments, param_sample = NULL)
```

Arguments

`model_base`	effect scenario object with mean parameters
`treatments`	treatments exposure levels as data frame (time, conc, trial)
`param_sample`	data.frame with parameter sample

Value

a `data.frame` with simulation results

Examples

```
exposure <- data.frame(time = Schmitt2013$t,
            conc = Schmitt2013$conc,
           trial = Schmitt2013$ID)

sim_result <- simulate_batch(model_base = metsulfuron,
                            treatments = exposure)
```

`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, times, ...)

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

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

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

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

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

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

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

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

## S4 method for signature 'DebDaphnia'
solver(
  scenario,
  times,
  approx = c("linear", "constant"),
  f = 1,
  method = "ode45",
  ...
)

## S4 method for signature 'AlgaeWeberScenario'
```

```
solver(
  scenario,
  times,
  approx = c("linear", "constant"),
  f = 1,
  method = "lsoda",
  hmax = 0.1,
  ...
)

## S4 method for signature 'AlgaeTKTDScenario'
solver(
  scenario,
  times,
  approx = c("linear", "constant"),
  f = 1,
  method = "lsoda",
  hmax = 0.1,
  ...
)

## S4 method for signature 'AlgaeSimpleScenario'
solver(
  scenario,
  times,
  approx = c("linear", "constant"),
  f = 1,
  method = "ode45",
  hmax = 0.01,
  ...
)
```

Arguments

`scenario`	scenario object
`times`	numeric vector of output times, overrides any scenario setting
`...`	additional parameters passed on to `deSolve::ode()`
`approx`	string, interpolation method of exposure series, see `stats::approxfun()`
`f`	if approx=`"constant"`, a number between 0 and 1 inclusive, see `stats::approxfun()`
`method`	string, numerical solver used by `deSolve::ode()`
`hmax`	numeric, maximum step length in time, see `deSolve::ode()`

Value

`data.frame` with simulation results

Methods (by class)

- `solver(ANY)`: Default solver, raises an error
- `solver(GutsRedSd)`: Numerically integrates GUTS-RED-SD models
- `solver(GutsRedIt)`: Numerically integrates GUTS-RED-IT models
- `solver(LemnaSchmittScenario)`: Numerically integrates Lemna_Schmitt models
- `solver(LemnaSetacScenario)`: Numerically integrates Lemna_SETAC models
- `solver(MyrioExpScenario)`: Numerically integrates Myrio models
- `solver(MyrioLogScenario)`: Numerically integrates Myrio_log models
- `solver(DebAbj)`: Numerically integrates DEB_abj models
- `solver(DebDaphnia)`: Numerically integrates DEB_Daphnia models
- `solver(AlgaeWeberScenario)`: numerically integrates Algae_Weber models
- `solver(AlgaeTKTDScenario)`: numerically integrates Algae_TKTD models
- `solver(AlgaeSimpleScenario)`: numerically integrates Algae_Simple models

`survival` *Survival rate*

Description

Deprecated function. Derives the survival rate of individuals for *Reduced GUTS* models. Was replaced by `simulate()`.

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")
```

`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 time points at which transfers occur, e.g. `c(7,10,14)`
- `transfer.interval` `numeric`, interval length until frond transfer to new medium
- `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()`.

Any transfer time point must also be an output time point. If a transfer occurs, simulation results of that time point will report the model state **before** the transfer. Be aware that in order to use transfers at regular intervals, the simulation must start at time point zero.

See Also

`set_transfer()`

Other scenarios: `Algae-models`, `DEB-models`, `GUTS-RED-models`, `Lemna-models`, `Macrophyte-models`, `Myriophyllum-models`, `Scenarios`

Index