Package ‘biogrowth’

November 8, 2024

Type Package

Title Modelling of Population Growth

Version 1.0.5

Description Modelling of population growth under static and dynamic environmental conditions. Includes functions for model fitting and making prediction under isothermal and dynamic conditions. The methods (algorithms & models) are based on predictive microbiology (See Perez-Rodriguez and Valero (2012, ISBN:978-1-4614-5519-6)).

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports deSolve (>= 1.28), tibble (>= 3.0.3), dplyr (>= 0.8.5), FME (>= 1.3.6), MASS (>= 7.3), rlang (>= 0.4.7), purrr (>= 0.3.4), ggplot2 (>= 3.3.2), cowplot (>= 1.0.0), lamW (>= 1.3.0), tidyr (>= 1.0.2), formula.tools (>= 1.7.1), lifecycle

Suggests knitr, rmarkdown, tidyverse (>= 1.3.0)

VignetteBuilder knitr

Depends R (>= 2.10)

NeedsCompilation no

Author Alberto Garre [aut, cre] (<https://orcid.org/0000-0002-4404-3550>),
Jeroen Koomen [aut],
Heidy den Besten [aut],
Marcel Zwietering [aut]

Maintainer Alberto Garre <`garre.alberto@gmail.com`>

Repository CRAN

Date/Publication 2024-11-08 17:50:02 UTC

Contents

`approx_env` *Generates functions for linear interpolation of environmental conditions*

Description

Generates functions for linear interpolation of environmental conditions

Usage

```
approx_env(env_conditions)
```

Arguments

`env_conditions`	A tibble describing the variation of the environmental conditions through the storage time. Must contain a column named `time` and as many additional columns as environmental factors.

Value

A list of functions that return the value of each environmental condition for some storage time

---

`arabian_tractors` *Number of tractors in the Arab World according to the World Bank*

Description

A dataset showing the increase in tractors in the Arab World. It was retrieved from https://data.worldbank.org/indicator/AG.A

Usage

```
arabian_tractors
```

Format

A tibble with 40 rows (each corresponding to one year) and 7 columns:

year Year for the recording

tractors Number of tractors

`Aryani_model` *Secondary Aryani model*

Description

Secondary model as defined by Aryani et al. (2015).

Usage

```
Aryani_model(x, xmin, xhalf)
```

Arguments

`x`	Value of the environmental factor.
`xmin`	Minimum value for growth.
`xhalf`	Value where gamma = 0.5

Value

The corresponding gamma factor.

`bilinear_lag` *Bilinear model with lag phase*

Description

Bilinear model with lag phase

Usage

```
bilinear_lag(times, logN0, mu, lambda)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration

`bilinear_stationary` *Bilinear model with stationary phase*

Description

Bilinear model with stationary phase

Usage

```
bilinear_stationary(times, logN0, mu, logNmax)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`logNmax`	Maximum log microbial count

`calculate_gammas` *Calculates every gamma factor*

Description

A helper function for `predict_dynamic_growth()` that calculates the value of every gamma factor corresponding to some storage time.

Usage

```
calculate_gammas(this_t, env_func, sec_models)
```

Arguments

`this_t`	Storage time
`env_func`	A list of functions (generated using `approxfun`) that give the value of each environmental function for some storage time.
`sec_models`	A nested list describing the secondary models.

Value

A vector of gamma factors (one per environmental factor).

`calculate_gammas_secondary`
Gamma factors for fitting secondary models

Description

A helper for fitting the secondary gamma models. Calculates the gamma factors corresponding to the models defined and the experimental conditions. In order for it to work, the environmental factors must be named identically in the 3 arguments.

Usage

```
calculate_gammas_secondary(sec_model_names, my_data, secondary_models)
```

Arguments

`sec_model_names`	named character vector defining the type of secondary model. Its names correspond to the environmental conditions and its values define the corresponding type of secondary model.
`my_data`	Tibble of experimental conditions.
`secondary_models`	A list defining the parameters of the secondary models.

Value

a numeric vector of length `nrow(my_data)` with the gamma factor for each experimental condition.

`check_growth_guess` *Visual check of an initial guess of the model parameters*

Description

[Stable]

Generates a plot comparing a set of data points against the model prediction corresponding to an initial guess of the model parameters

Usage

```
check_growth_guess(
  fit_data,
  model_keys,
  guess,
  environment = "constant",
  env_conditions = NULL,
```

```
  approach = "single",
  logbase_mu = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	Tibble (or data.frame) of data for the fit. It must have two columns, one with the elapsed time (`time` by default) and another one with the decimal logarithm of the populatoin size (`logN` by default). Different column names can be defined using the `formula` argument.
`model_keys`	Named the equations of the secondary model as in `fit_growth()`
`guess`	Named vector with the initial guess of the model parameters as in `fit_growth()`
`environment`	type of environment. Either "constant" (default) or "dynamic" (see below for details on the calculations for each condition)
`env_conditions`	Tibble describing the variation of the environmental conditions for dynamic experiments. See `fit_growth()`. Ignored when `environment = "constant"`
`approach`	whether "single" (default) or "global". Please see `fit_growth()` for details.“
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.

Value

A `ggplot2::ggplot()` comparing the model prediction against the data

Examples

```
## Examples under constant environmental conditions --------------------------

## We need some data

my_data <- data.frame(time = 0:9,
                      logN = c(2, 2.1, 1.8, 2.5, 3.1, 3.4, 4, 4.5, 4.8, 4.7)
                      )

## We can directly plot the comparison for some values

check_growth_guess(my_data, list(primary = "modGompertz"),
                   c(logN0 = 1.5, mu = .8, lambda = 4, C = 3)
                   )

## Ot it can be combined with the automatic initial guess

check_growth_guess(my_data, list(primary = "modGompertz"),
                   make_guess_primary(my_data, "modGompertz")
                   )
```

```
## Examples under dynamic environmental conditions --------------------------

## We will use the datasets included in the package

data("example_dynamic_growth")
data("example_env_conditions")

## Model equations are assigned as in fit_growth

sec_models <- list(temperature = "CPM", aw = "CPM")

## Guesses of model parameters are also defined as in fit_growth

guess <- list(Nmax = 1e4,
              N0 = 1e0, Q0 = 1e-3,
              mu_opt = 4,
              temperature_n = 1,
              aw_xmax = 1, aw_xmin = .9, aw_n = 1,
              temperature_xmin = 25, temperature_xopt = 35,
              temperature_xmax = 40, aw_xopt = .95
              )

## We can now check our initial guess

check_growth_guess(example_dynamic_growth, sec_models, guess,
                   "dynamic",
                   example_env_conditions)
```

`check_primary_pars` *Basic check of parameters for primary models*

Description

Checks that: the model name is correct, the right number of model parameters have been defined and that the parameters have the right names

Usage

```
check_primary_pars(model_name, pars)
```

Arguments

`model_name`	Model identifier
`pars`	A named list of model parameters

Value

If there is no error, the model function.

`check_secondary_pars` *Basic checks of secondary parameters*

Description

Checks that the model names are correct, that no parameter is defined twice, that every parameter is defined and that no unknown parameter has been defined. Raises an error if any of these conditions is not met.

Usage

```
check_secondary_pars(
  starting_point,
  known_pars,
  sec_model_names,
  primary_pars = "mu_opt"
)
```

Arguments

`starting_point`	Named vector with initial values for the model parameters to estimate from the data. The growth rate under optimum conditions must be named `mu_opt`. The rest must be called 'env_factor'+'_'+'parameter'. For instance, the minimum pH for growth is 'pH_xmin'.
`known_pars`	Named vector of fixed model parameters. Must be named using the same convention as `starting_point`.
`sec_model_names`	Named character vector defining the secondary model for each environmental factor.
`primary_pars`	Character vector with the parameter names of the primary model.

`check_stochastic_pars` *Model definition checks for predict_stochastic_growth*

Description

Does several checks of the model parameters. Besides those by check_primary_pars, it checks that corr_matrix is square, that pars and corr_matrix have compatible dimensions, and that pars has the correct names.

Usage

```
check_stochastic_pars(model_name, pars, corr_matrix)
```

Arguments

`model_name`	Character describing the primary growth model.
`pars`	A tibble describing the parameter uncertainty (see details).
`corr_matrix`	Correlation matrix of the model parameters. Defined in the same order as in `pars`. An identity matrix by default (uncorrelated parameters).

`compare_growth_fits` *Model comparison and selection for growth models*

Description

[Experimental]

This function is a constructor for GrowthComparison or GlobalGrowthComparison, a class that provides several functions for model comparison and model selection for growth models fitted using `fit_growth()`. Please see the help pages for GrowthComparison or GlobalGrowthComparison for further details.

Although it is not necessary, we recommend passing the models as a named list, as these names will later be kept in plots and tables.

Usage

```
compare_growth_fits(models)
```

Arguments

`models`	a (we recommend named) list of models fitted using `fit_growth()`. Every model should be of the same class. Otherwise, some functions may give unexpected results.

Examples

```
## Example 1 - Fitting under static environmental conditions ----------------

## We will use the data on growth of Salmonella included in the package

data("growth_salmonella")

## We will fit 3 different models to the data

fit1 <- fit_growth(growth_salmonella,
                   list(primary = "Baranyi"),
                   start = c(lambda = 0, logNmax = 8, mu = .1, logN0 = 2),
                   known = c(),
                   environment = "constant",
                   )

fit2 <- fit_growth(growth_salmonella,
```

```
                   list(primary = "Baranyi"),
                   start = c(logNmax = 8, mu = .1, logN0 = 2),
                   known = c(lambda = 0),
                   environment = "constant",
                   )

fit3 <- fit_growth(growth_salmonella,
                   list(primary = "modGompertz"),
                   start = c(C = 8, mu = .1, logN0 = 2),
                   known = c(lambda = 0),
                   environment = "constant",
                   )

## We can now put them in a (preferably named) list

my_models <- list(`Baranyi` = fit1,
                  `Baranyi no lag` = fit2,
                  `Gompertz no lag` = fit3)

## And pass them to compare_growth_fits

model_comparison <- compare_growth_fits(my_models)

##  The instance of GrowthComparison has useful S3 methods

print(model_comparison)
plot(model_comparison)
plot(model_comparison, type = 2)
plot(model_comparison, type = 3)

## The statistical indexes can be accessed through summary and coef

summary(model_comparison)
coef(model_comparison)

## Example 2 - Fitting under dynamic environmental conditions ---------------


## We will use one of the example datasets

data("example_dynamic_growth")
data("example_env_conditions")

## First model fitted

sec_models <- list(temperature = "CPM", aw = "CPM")

known_pars <- list(Nmax = 1e4,
                   N0 = 1e0, Q0 = 1e-3,
                   mu_opt = 4,
                   temperature_n = 1,
                   aw_xmax = 1, aw_xmin = .9, aw_n = 1
                   )
```

```
my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
                 temperature_xmax = 40, aw_xopt = .95)

dynamic_fit <- fit_growth(example_dynamic_growth,
                          sec_models,
                          my_start, known_pars,
                          environment = "dynamic",
                          env_conditions = example_env_conditions
                          )

## Second model (different secondary model for temperature)

sec_models <- list(temperature = "Zwietering", aw = "CPM")

known_pars <- list(Nmax = 1e4,
                   N0 = 1e0, Q0 = 1e-3,
                   mu_opt = 4,
                   temperature_n = 1,
                   aw_xmax = 1, aw_xmin = .9, aw_n = 1
                   )

my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
                 aw_xopt = .95)

dynamic_fit2 <- fit_growth(example_dynamic_growth,
                           sec_models,
                           my_start, known_pars,
                           environment = "dynamic",
                           env_conditions = example_env_conditions
                           )

## Once both models have been fitted, we can call the function

dynamic_comparison <- compare_growth_fits(list(m1 = dynamic_fit, m2 = dynamic_fit2))

## Which also returns an instance of GrowthComparison with the same S3 methods

print(dynamic_comparison)
plot(dynamic_comparison)
plot(dynamic_comparison, type = 2)
plot(dynamic_comparison, type = 3)

## The statistical indexes can be accessed through summary and coef

summary(dynamic_comparison)
coef(dynamic_comparison)


## Example 3 - Global fitting -----------------------------------------------


## We use the example data
```

```
data("multiple_counts")
data("multiple_conditions")

## We need to fit (at least) two models

sec_models <- list(temperature = "CPM", pH = "CPM")

known_pars <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
                   temperature_n = 2, temperature_xmin = 20,
                   temperature_xmax = 35,
                   pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

my_start <- list(mu_opt = .8, temperature_xopt = 30)

global_fit <- fit_growth(multiple_counts,
                         sec_models,
                         my_start,
                         known_pars,
                         environment = "dynamic",
                         algorithm = "regression",
                         approach = "global",
                         env_conditions = multiple_conditions
                         )

sec_models <- list(temperature = "CPM", pH = "CPM")

known_pars <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
                   temperature_n = 1, temperature_xmin = 20,
                   temperature_xmax = 35,
                   pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

my_start <- list(mu_opt = .8, temperature_xopt = 30)

global_fit2 <- fit_growth(multiple_counts,
                          sec_models,
                          my_start,
                          known_pars,
                          environment = "dynamic",
                          algorithm = "regression",
                          approach = "global",
                          env_conditions = multiple_conditions
                          )

## We can now pass both models to the function as a (named) list

global_comparison <- compare_growth_fits(list(`n=2` = global_fit,
                                              `n=1` = global_fit2)
                                         )

## The residuals and model fits plots are divided by experiments

plot(global_comparison)
```

```
plot(global_comparison, type = 3)

## The remaining S3 methods are the same as before

print(global_comparison)
plot(global_comparison, type = 2)
summary(global_comparison)
coef(global_comparison)
```

`compare_secondary_fits` *Model comparison and selection for secondary growth models*

Description

[Experimental]

This function is a constructor for SecondaryComparison a class that provides several functions for model comparison and model selection for growth models fitted using `fit_secondary_growth()`. Please see the help pages for SecondaryComparison for further details.

Although it is not necessary, we recommend passing the models as a named list, as these names will later be kept in plots and tables.

Usage

```
compare_secondary_fits(models)
```

Arguments

`models`	a (we recommend named) list of models fitted using `fit_secondary_growth()`.

Examples

```
## We first need to fit some models

data("example_cardinal")

sec_model_names <- c(temperature = "Zwietering", pH = "CPM")

known_pars <- list(mu_opt = 1.2, temperature_n = 1,
                   pH_n = 2, pH_xmax = 6.8, pH_xmin = 5.2)

my_start <- list(temperature_xmin = 5, temperature_xopt = 35,
                 pH_xopt = 6.5)

fit1 <- fit_secondary_growth(example_cardinal, my_start, known_pars, sec_model_names)
```

```
known_pars <- list(mu_opt = 1.2, temperature_n = 2,
                   pH_n = 2, pH_xmax = 6.8, pH_xmin = 5.2)

 fit2 <- fit_secondary_growth(example_cardinal, my_start, known_pars, sec_model_names)

 ## We can now pass the models to the constructor

 comparison <- compare_secondary_fits(list(`n=1` = fit1,
                                           `n=2` = fit2))

 ## The function includes several S3 methods for model selection and comparison

 print(comparison)

 plot(comparison)
 plot(comparison, type = 2)

 ## The numerical indexes can be accessed using coef and summary

 coef(comparison)
 summary(comparison)
```

`conditions_pH_temperature` *Conditions during a dynamic growth experiment*

Description

A dataset to demonstrate the use of fit_dynamic_growth. The observations environmental conditions are described in conditions_pH_temperature.

Usage

```
conditions_pH_temperature
```

Format

A tibble with 4 rows and 3 columns:

time elapsed time

temperature temperature

pH pH

`CPM_model` *Secondary Cardinal Parameter (CPM) model*

Description

Secondary cardinal parameter model as defined by Rosso et al. (1995).

Usage

```
CPM_model(x, xmin, xopt, xmax, n)
```

Arguments

`x`	Value of the environmental factor.
`xmin`	Minimum value for growth.
`xopt`	Optimum value for growth.
`xmax`	Maximum value for growth.
`n`	Order of the CPM model.

Value

The corresponding gamma factor.

`dBaranyi` *Baranyi growth model*

Description

Microbial growth model as defined in Baranyi and Roberts (1994). It has been implemented according to the requirements of `deSolve::ode()`. For consistency in the function for isothermal growth, calculations are done assuming the user input for mu is in log10 scale. In other words, the input is multiplied by ln(10).

Usage

```
dBaranyi(time, state, pars, env_func, sec_models)
```

Arguments

`time`	numeric vector (length 1) of storage time
`state`	named numeric vector with two components: Q and N
`pars`	named numeric vector of model parameters (Nmax and mu_opt)
`env_func`	named list of functions returning the values of the environmental conditions for time (t)
`sec_models`	named list of parameters of the secondary model

Value

A numeric vector of two components according to the requirements of `deSolve::ode()`.

`distribution_to_logcount` *Distribution of times to reach a certain microbial count*

Description

[Superseded]

The function `distribution_to_logcount()` has been superseded by function `time_to_size()`, which provides more general interface.

Returns the probability distribution of the storage time required for the microbial count to reach `log_count` according to the predictions of a stochastic `model`. Calculations are done using linear interpolation of the individual model predictions.

Usage

```
distribution_to_logcount(model, log_count)
```

Arguments

`model`	An instance of `StochasticGrowth` or `MCMCgrowth`.
`log_count`	The target microbial count.

Value

An instance of `TimeDistribution()`.

Examples

```
## We need an instance of StochasticGrowth

my_model <- "modGompertz"
my_times <- seq(0, 30, length = 100)
n_sims <- 3000

library(tibble)

pars <- tribble(
    ~par, ~mean, ~sd, ~scale,
    "logN0", 0, .2, "original",
    "mu", 2, .3, "sqrt",
    "lambda", 4, .4, "sqrt",
    "C", 6, .5, "original"
)
```

```
stoc_growth <- predict_stochastic_growth(my_model, my_times, n_sims, pars)
```

DynamicGrowth *DynamicGrowth class*

Description

[Superseded]

The class DynamicGrowth has been superseded by the top-level class GrowthPrediction, which provides a unified approach for growth modelling.

Still, it is returned if the superseded `predict_dynamic_growth()` is called.

A subclass of list with items:

- simulation: A tibble with the model prediction
- gammas: A tibble with the value of each gamma factor for each value of `times`.
- env_conditions: A list of functions interpolating the environmental conditions.
- primary_pars: A list with the model parameters of the primary model.
- sec_models: A nested list defining the secondary models.

Usage

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

## S3 method for class 'DynamicGrowth'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed",
  label_x = "time"
)

## S3 method for class 'DynamicGrowth'
coef(object, ...)
```

Arguments

`x`	The object of class `DynamicGrowth` to plot.
`...`	ignored
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`label_x`	Label of the x-axis.
`object`	an instance of DynamicGrowth

Methods (by generic)

- `print(DynamicGrowth)`: print of the model
- `plot(DynamicGrowth)`: predicted growth curve under dynamic conditions.
- `coef(DynamicGrowth)`: coefficients of the model

`example_cardinal` *Growth rates obtained for several growth experiments*

Description

An example dataset illustrating the requirements of the `fit_secondary_growth()` function.

Usage

```
example_cardinal
```

Format

A data frame with 64 rows and 3 variables:

temperature storage temperature (ºC)

pH pH of the media

mu specific growth rate (log10 CFU/h)

`example_dynamic_growth`

Microbial growth under dynamic conditions

Description

An example dataset illustrating the requirements of the `fit_dynamic_growth()` function.

Usage

```
example_dynamic_growth
```

Format

A data frame with 30 rows and 2 variables:

time elapsed time (h)

logN log population size (log10 CFU)

`example_env_conditions`

Environmental conditions during a dynamic experiment

Description

An example dataset illustrating the requirements of the `fit_dynamic_growth()` function.

Usage

```
example_env_conditions
```

Format

A data frame with 3 rows and 3 variables:

time elapsed time (h)

temperature storage temperature (ºC)

aw water activity

`extract_primary_pars` *A helper to build the primary models*

Description

Most of the functions for fitting mix in the vectors parameters for the primary and secondary models, but the functions for making predictions need that they are separated. This one extracts the parameters of the primary model.

Usage

```
extract_primary_pars(this_p, known_pars)
```

Arguments

`this_p`	A named vector of model parameters (usually, the ones fitted).
`known_pars`	Another named vector of model parameters (usually the known ones).

Value

A list with the parameters of the primary model

`extract_secondary_pars` *A helper to build the secondary models*

Description

Most of the functions for fitting mix in the vectors parameters for the primary and secondary models, but the functions for making predictions need that they are separated. This one extracts the parameters of the secondary model.

Usage

```
extract_secondary_pars(this_p, known_pars, sec_model_names)
```

Arguments

`this_p`	A named vector of model parameters (usually, the ones fitted).
`known_pars`	Another named vector of model parameters (usually the known ones).
`sec_model_names`	A named character vector defining for each environmental factor (vector names) the type of secondary model (vector values).

Value

A nested list defining the secondary models.

`FitDynamicGrowth` *FitDynamicGrowth class*

Description

[Superseded]

The class FitDynamicGrowth has been superseded by the top-level class GrowthFit, which provides a unified approach for growth modelling.

Still, it is still returned if the superseded `fit_dynamic_growth()` is called.

It is a subclass of list with the items:

- fit_results: the object returned by `modFit`.
- best_prediction: the model prediction for the fitted parameters.
- env_conditions: environmental conditions for the fit.
- data: data used for the fit.
- starting: starting values for model fitting
- known: parameter values set as known.
- sec_models: a named vector with the secondary model for each environmental factor

Usage

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

## S3 method for class 'FitDynamicGrowth'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = 1,
  point_col = "black",
  point_size = 3,
  point_shape = 16,
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed"
)
```

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

## S3 method for class 'FitDynamicGrowth'
residuals(object, ...)

## S3 method for class 'FitDynamicGrowth'
coef(object, ...)

## S3 method for class 'FitDynamicGrowth'
vcov(object, ...)

## S3 method for class 'FitDynamicGrowth'
deviance(object, ...)

## S3 method for class 'FitDynamicGrowth'
fitted(object, ...)

## S3 method for class 'FitDynamicGrowth'
predict(object, times = NULL, newdata = NULL, ...)

## S3 method for class 'FitDynamicGrowth'
logLik(object, ...)

## S3 method for class 'FitDynamicGrowth'
AIC(object, ..., k = 2)
```

Arguments

`x`	The object of class `FitDynamicGrowth` to plot.
`...`	ignored
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`point_col`	Aesthetic parameter to change the colour of the point geom, see: `ggplot2::geom_point()`
`point_size`	Aesthetic parameter to change the size of the point geom, see: `ggplot2::geom_point()`

`point_shape`	Aesthetic parameter to change the shape of the point geom, see: ggplot2::`ggplot2::geom_point()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`object`	an instance of FitDynamicGrowth
`times`	A numeric vector with the time points for the simulations. `NULL` by default (using the same time points as those for the simulation).
`newdata`	a tibble describing the environmental conditions (as `env_conditions`) in `predict_dynamic_growth()`. If `NULL` (default), uses the same conditions as those for fitting.
`k`	penalty for the parameters (k=2 by default)

Methods (by generic)

- `print(FitDynamicGrowth)`: comparison between the fitted model and the data.
- `plot(FitDynamicGrowth)`: comparison between the fitted model and the data.
- `summary(FitDynamicGrowth)`: statistical summary of the fit.
- `residuals(FitDynamicGrowth)`: residuals of the model.
- `coef(FitDynamicGrowth)`: vector of fitted parameters.
- `vcov(FitDynamicGrowth)`: (unscaled) variance-covariance matrix of the model, calculated as 1/(0.5*Hessian)
- `deviance(FitDynamicGrowth)`: deviance of the model.
- `fitted(FitDynamicGrowth)`: fitted values.
- `predict(FitDynamicGrowth)`: model predictions.
- `logLik(FitDynamicGrowth)`: loglikelihood of the model
- `AIC(FitDynamicGrowth)`: Akaike Information Criterion

`FitDynamicGrowthMCMC` *FitDynamicGrowthMCMC class*

Description

[Superseded]

The class FitDynamicGrowthMCMC has been superseded by the top-level class GrowthFit, which provides a unified approach for growth modelling.

Still, it is returned if the superseded `fit_MCMC_growth()` is called.

It is a subclass of list with the items:

- fit_results: the object returned by `modMCMC`.
- best_prediction: the model prediction for the fitted parameters.
- env_conditions: environmental conditions for the fit.
- data: data used for the fit.
- starting: starting values for model fitting
- known: parameter values set as known.
- sec_models: a named vector with the secondary model for each environmental factor

Usage

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

## S3 method for class 'FitDynamicGrowthMCMC'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = 1,
  point_col = "black",
  point_size = 3,
  point_shape = 16,
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed"
)

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

## S3 method for class 'FitDynamicGrowthMCMC'
residuals(object, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
coef(object, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
vcov(object, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
deviance(object, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
fitted(object, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
predict(object, times = NULL, newdata = NULL, ...)

## S3 method for class 'FitDynamicGrowthMCMC'
logLik(object, ...)
```

```
## S3 method for class 'FitDynamicGrowthMCMC'
AIC(object, ..., k = 2)

## S3 method for class 'FitDynamicGrowthMCMC'
predictMCMC(
  model,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`x`	The object of class `FitDynamicGrowthMCMC` to plot.
`...`	ignored
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmenta factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`point_col`	Aesthetic parameter to change the colour of the point geom, see: `ggplot2::geom_point()`
`point_size`	Aesthetic parameter to change the size of the point geom, see: `ggplot2::geom_point()`
`point_shape`	Aesthetic parameter to change the shape of the point geom, see: `ggplot2::geom_point()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`object`	an instance of FitDynamicGrowthMCMC
`times`	Numeric vector of storage times for the predictions.
`newdata`	a tibble describing the environmental conditions (as `env_conditions`) in `predict_dynamic_growth()`. If `NULL` (default), uses the same conditions as those for fitting.
`k`	penalty for the parameters (k=2 by default)
`model`	An instance of FitDynamicGrowthMCMC

`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column named 'time' with the storage time and as many columns as required with the environmental conditions.
`niter`	Number of iterations.
`newpars`	A named list defining new values for the some model parameters. The name must be the identifier of a model already included in the model. These parameters do not include variation, so defining a new value for a fitted parameters "fixes" it. `NULL` by default (no new parameters).
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

Value

An instance of `MCMCgrowth()`.

Methods (by generic)

- `print(FitDynamicGrowthMCMC)`: print of the model
- `plot(FitDynamicGrowthMCMC)`: compares the model fitted against the data.
- `summary(FitDynamicGrowthMCMC)`: statistical summary of the fit.
- `residuals(FitDynamicGrowthMCMC)`: model residuals.
- `coef(FitDynamicGrowthMCMC)`: vector of fitted model parameters.
- `vcov(FitDynamicGrowthMCMC)`: variance-covariance matrix of the model, estimated as the variance of the samples from the Markov chain.
- `deviance(FitDynamicGrowthMCMC)`: deviance of the model, calculated as the sum of squared residuals for the parameter values resulting in the best fit.
- `fitted(FitDynamicGrowthMCMC)`: vector of fitted values.
- `predict(FitDynamicGrowthMCMC)`: vector of model predictions.
- `logLik(FitDynamicGrowthMCMC)`: loglikelihood of the model
- `AIC(FitDynamicGrowthMCMC)`: Akaike Information Criterion
- `predictMCMC(FitDynamicGrowthMCMC)`: prediction including parameter uncertainty

`FitIsoGrowth` *FitIsoGrowth class*

Description

[Superseded]

The class FitIsoGrowth has been superseded by the top-level class GrowthFit, which provides a unified approach for growth modelling.

Still, it is still returned if the superseded `fit_isothermal_growth()` is called.

It is a subclass of list with the items:

- data: data used for model fitting
- model: name of the primary inactivation model
- starting_point: initial value of the model parameters
- known: fixed model parameters
- fit: object returned by `FME::modFit()`
- best_prediction: model prediction for the model fitted.

Usage

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

## S3 method for class 'FitIsoGrowth'
plot(
  x,
  y = NULL,
  ...,
  line_col = "black",
  line_size = 1,
  line_type = 1,
  point_col = "black",
  point_size = 3,
  point_shape = 16
)

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

## S3 method for class 'FitIsoGrowth'
residuals(object, ...)

## S3 method for class 'FitIsoGrowth'
coef(object, ...)

## S3 method for class 'FitIsoGrowth'
vcov(object, ...)

## S3 method for class 'FitIsoGrowth'
deviance(object, ...)

## S3 method for class 'FitIsoGrowth'
fitted(object, ...)

## S3 method for class 'FitIsoGrowth'
predict(object, times = NULL, ...)

## S3 method for class 'FitIsoGrowth'
```

```
logLik(object, ...)

## S3 method for class 'FitIsoGrowth'
AIC(object, ..., k = 2)
```

Arguments

`x`	The object of class `FitIsoGrowth` to plot.
`...`	ignored
`y`	ignored
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`point_col`	Aesthetic parameter to change the colour of the point geom, see: `ggplot2::geom_point()`
`point_size`	Aesthetic parameter to change the size of the point geom, see: `ggplot2::geom_point()`
`point_shape`	Aesthetic parameter to change the shape of the point geom, see: `ggplot2::geom_point()`
`object`	an instance of FitIsoGrowth
`times`	numeric vector describing the time points for the prediction. If `NULL` (default), uses the same points as those used for fitting.
`k`	penalty for the parameters (k=2 by default)

Methods (by generic)

- `print(FitIsoGrowth)`: print of the model
- `plot(FitIsoGrowth)`: compares the fitted model against the data.
- `summary(FitIsoGrowth)`: statistical summary of the fit.
- `residuals(FitIsoGrowth)`: vector of model residuals.
- `coef(FitIsoGrowth)`: vector of fitted model parameters.
- `vcov(FitIsoGrowth)`: variance-covariance matrix of the model, estimated as 1/(0.5*Hessian)
- `deviance(FitIsoGrowth)`: deviance of the model.
- `fitted(FitIsoGrowth)`: vector of fitted values.
- `predict(FitIsoGrowth)`: vector of model predictions.
- `logLik(FitIsoGrowth)`: loglikelihood of the model
- `AIC(FitIsoGrowth)`: Akaike Information Criterion

`FitMultipleDynamicGrowth`

FitMultipleDynamicGrowth class

Description

[Superseded]

The class FitMultipleDynamicGrowth has been superseded by the top-level class GlobalGrowthFit, which provides a unified approach for growth modelling.

Still, it is still returned if the superseded `fit_multiple_growth()` is called.

It is a subclass of list with the items:

- fit_results: the object returned by `modFit`.
- best_prediction: a list with the models predictions for each condition.
- data: a list with the data used for the fit.
- starting: starting values for model fitting
- known: parameter values set as known.
- sec_models: a named vector with the secondary model for each environmental factor.

Usage

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

## S3 method for class 'FitMultipleDynamicGrowth'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_x = "time",
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed",
  point_size = 3,
  point_shape = 16,
  subplot_labels = "AUTO"
)
```

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

## S3 method for class 'FitMultipleDynamicGrowth'
residuals(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
coef(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
vcov(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
deviance(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
fitted(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
predict(object, env_conditions, times = NULL, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
logLik(object, ...)

## S3 method for class 'FitMultipleDynamicGrowth'
AIC(object, ..., k = 2)
```

Arguments

`x`	an instance of FitMultipleDynamicGrowth.
`...`	ignored
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_x`	label of the x-axis
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`

`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`point_size`	Size of the data points
`point_shape`	shape of the data points
`subplot_labels`	labels of the subplots according to `plot_grid`.
`object`	an instance of FitMultipleDynamicGrowth
`env_conditions`	a tibble describing the environmental conditions (as in `fit_multiple_growth()`.
`times`	A numeric vector with the time points for the simulations. `NULL` by default (using the same time points as the ones defined in `env_conditions`).
`k`	penalty for the parameters (k=2 by default)

Methods (by generic)

- `print(FitMultipleDynamicGrowth)`: print of the model
- `plot(FitMultipleDynamicGrowth)`: comparison between the fitted model and the experimental data.
- `summary(FitMultipleDynamicGrowth)`: statistical summary of the fit.
- `residuals(FitMultipleDynamicGrowth)`: calculates the model residuals. Returns a tibble with 4 columns: time (storage time), logN (observed count), exp (name of the experiment) and res (residual).
- `coef(FitMultipleDynamicGrowth)`: vector of fitted parameters.
- `vcov(FitMultipleDynamicGrowth)`: (unscaled) variance-covariance matrix, estimated as 1/(0.5*Hessian).
- `deviance(FitMultipleDynamicGrowth)`: deviance of the model.
- `fitted(FitMultipleDynamicGrowth)`: fitted values. They are returned as a tibble with 3 columns: time (storage time), exp (experiment identifier) and fitted (fitted value).
- `predict(FitMultipleDynamicGrowth)`: vector of model predictions
- `logLik(FitMultipleDynamicGrowth)`: loglikelihood of the model
- `AIC(FitMultipleDynamicGrowth)`: Akaike Information Criterion

`FitMultipleGrowthMCMC` *FitMultipleGrowthMCMC class*

Description

[Superseded]

The class FitMultipleGrowthMCMC has been superseded by the top-level class GlobalGrowthFit, which provides a unified approach for growth modelling.

Still, it is still returned if the superseded `fit_multiple_growth_MCMC()` is called.

It is a subclass of list with the items:

- fit_results: the object returned by `modFit`.
- best_prediction: a list with the models predictions for each condition.
- data: a list with the data used for the fit.
- starting: starting values for model fitting
- known: parameter values set as known.
- sec_models: a named vector with the secondary model for each environmental factor.

Usage

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

## S3 method for class 'FitMultipleGrowthMCMC'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_x = "time",
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed",
  point_size = 3,
  point_shape = 16,
  subplot_labels = "AUTO"
)

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

## S3 method for class 'FitMultipleGrowthMCMC'
residuals(object, ...)
```

```
## S3 method for class 'FitMultipleGrowthMCMC'
coef(object, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
vcov(object, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
deviance(object, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
fitted(object, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
predict(object, env_conditions, times = NULL, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
logLik(object, ...)

## S3 method for class 'FitMultipleGrowthMCMC'
AIC(object, ..., k = 2)

## S3 method for class 'FitMultipleGrowthMCMC'
predictMCMC(
  model,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`x`	an instance of FitMultipleGrowthMCMC.
`...`	ignored
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_x`	label of the x-axis
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`

`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`point_size`	Size of the data points
`point_shape`	shape of the data points
`subplot_labels`	labels of the subplots according to `plot_grid`.
`object`	an instance of FitMultipleGrowthMCMC
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column named 'time' with the storage time and as many columns as required with the environmental conditions.
`times`	Numeric vector of storage times for the predictions.
`k`	penalty for the parameters (k=2 by default)
`model`	An instance of FitMultipleGrowthMCMC
`niter`	Number of iterations.
`newpars`	A named list defining new values for the some model parameters. The name must be the identifier of a model already included in the model. These parameters do not include variation, so defining a new value for a fitted parameters "fixes" it. `NULL` by default (no new parameters).
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

Value

An instance of `MCMCgrowth()`.

Methods (by generic)

- `print(FitMultipleGrowthMCMC)`: print of the model
- `plot(FitMultipleGrowthMCMC)`: comparison between the model fitted and the data.
- `summary(FitMultipleGrowthMCMC)`: statistical summary of the fit.
- `residuals(FitMultipleGrowthMCMC)`: model residuals. They are returned as a tibble with 4 columns: time (storage time), logN (observed count), exp (name of the experiment) and res (residual).
- `coef(FitMultipleGrowthMCMC)`: vector of fitted model parameters.
- `vcov(FitMultipleGrowthMCMC)`: variance-covariance matrix of the model, estimated as the variance of the samples from the Markov chain.
- `deviance(FitMultipleGrowthMCMC)`: deviance of the model, calculated as the sum of squared residuals of the prediction with the lowest standard error.
- `fitted(FitMultipleGrowthMCMC)`: fitted values of the model. They are returned as a tibble with 3 columns: time (storage time), exp (experiment identifier) and fitted (fitted value).

- `predict(FitMultipleGrowthMCMC)`: model predictions. They are returned as a tibble with 3 columns: time (storage time), logN (observed count), and exp (name of the experiment).
- `logLik(FitMultipleGrowthMCMC)`: loglikelihood of the model
- `AIC(FitMultipleGrowthMCMC)`: Akaike Information Criterion
- `predictMCMC(FitMultipleGrowthMCMC)`: prediction including parameter uncertainty

`FitSecondaryGrowth` *FitSecondaryGrowth class*

Description

The `FitSecondaryGrowth` class contains a model fitted to a set of growth rates gathered under a variety of static conditions. Its constructor is `fit_secondary_growth()`.

It is a subclass of list with the items:

- fit_results: object returned by `FME::modFit()`.
- secondary_model: secondary model fitted to the data.
- mu_opt_fit: estimated growth rate under optimum conditions.
- data: data used for the fit.
- transformation: type of transformation of `mu` for the fit.

Usage

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

## S3 method for class 'FitSecondaryGrowth'
plot(x, y = NULL, ..., which = 1, add_trend = FALSE, add_segment = FALSE)

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

## S3 method for class 'FitSecondaryGrowth'
residuals(object, ...)

## S3 method for class 'FitSecondaryGrowth'
coef(object, ...)

## S3 method for class 'FitSecondaryGrowth'
vcov(object, ...)

## S3 method for class 'FitSecondaryGrowth'
deviance(object, ...)

## S3 method for class 'FitSecondaryGrowth'
```

```
fitted(object, ...)

## S3 method for class 'FitSecondaryGrowth'
predict(object, newdata = NULL, ...)

## S3 method for class 'FitSecondaryGrowth'
logLik(object, ...)

## S3 method for class 'FitSecondaryGrowth'
AIC(object, ..., k = 2)
```

Arguments

`x`	An instance of FitSecondaryGrowth.
`...`	ignored
`y`	ignored.
`which`	A numeric with the type of plot. 1 for obs versus predicted (default), 2 for gamma curve
`add_trend`	Whether to add a trend line (only for which=2)
`add_segment`	Whether to join the observed and fitted points (only for which=2)
`object`	an instance of FitSecondaryGrowth
`newdata`	A tibble describing the environmental conditions as in `fit_secondary_growth()`. If `NULL`, it uses the same conditions as for model fitting (default).
`k`	penalty for the parameters (k=2 by default)

Methods (by generic)

- `print(FitSecondaryGrowth)`: print of the model
- `plot(FitSecondaryGrowth)`: plots to evaluate the goodness of the fit.
- `summary(FitSecondaryGrowth)`: statistical summary of the fit.
- `residuals(FitSecondaryGrowth)`: vector of model residuals.
- `coef(FitSecondaryGrowth)`: vector of fitted model parameters.
- `vcov(FitSecondaryGrowth)`: variance-covariance matrix of the model, estimated as 1/(0.5*Hessian)
- `deviance(FitSecondaryGrowth)`: deviance of the model.
- `fitted(FitSecondaryGrowth)`: vector of fitted values.

  The fitted values are returned in the same scale as the one used for the fitting (sqrt, log or none).
- `predict(FitSecondaryGrowth)`: vector of model predictions.
- `logLik(FitSecondaryGrowth)`: loglikelihood of the model
- `AIC(FitSecondaryGrowth)`: Akaike Information Criterion

`fit_dynamic_growth` *Fit dynamic growth models*

Description

[Superseded]

The function `fit_dynamic_growth()` has been superseded by the top-level function `fit_growth()`, which provides a unified approach for growth modelling.

Nonetheless, it can still fit a growth model to data obtained under dynamic conditions using the one-step approach (non-linear regression).

Usage

```
fit_dynamic_growth(
  fit_data,
  env_conditions,
  starting_point,
  known_pars,
  sec_model_names,
  ...,
  check = TRUE,
  logbase_mu = logbase_logN,
  logbase_logN = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	Tibble with the data to use for model fit. It must contain a column with the elapsed time (named "time" by default) and another one with the decimal logarithm of the observed population size (named "logN" by default). Different column names can be specified using the "formula" argument.
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column with the elapsed time (named "time" by default) and as many columns as required with the environmental conditions. A different column name can be specified using the "formula" argument, although it must be the same one as in "fit_data". Note that only those defined in "sec_model_names" will be considered for the model fit.
`starting_point`	A named vector of starting values for the model parameters. Parameters for the primary model must be named in the usual way. Parameters for the secondary model are named as `env_factor`+'_'+`parameter`. For instance, the maximum growth temperature shall be named 'temperature_xmax'.
`known_pars`	A named vector of known model parameters (i.e. not fitted). They must be named using the same convention as for `starting_point`.

`sec_model_names`	A named character vector defining the secondary model for each environmental factor. The names define the factor and the value the type of model. Names must match columns in `fit_data` and `env_conditions`.
`...`	Additional arguments passed to modFit.
`check`	Whether to check model parameters (TRUE by default).
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.

Value

An instance of `FitDynamicGrowth()`.

Examples

```
## We use the datasets included in the package

data("example_dynamic_growth")
data("example_env_conditions")

## Define the secondary models

sec_model_names <- c(temperature = "CPM", aw= "CPM")

## Any model parameter can be fixed

known_pars <- list(Nmax = 1e4,  # Primary model
    N0 = 1e0, Q0 = 1e-3,  # Initial values of the primary model
    mu_opt = 4, # mu_opt of the gamma model
    temperature_n = 1,  # Secondary model for temperature
    aw_xmax = 1, aw_xmin = .9, aw_n = 1  # Secondary model for water activity
    )

## The remaining parameters need initial values

my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
    temperature_xmax = 40, aw_xopt = .95)

## We can now call the fitting function

my_dyna_fit <- fit_dynamic_growth(example_dynamic_growth, example_env_conditions,
    my_start, known_pars, sec_model_names)

summary(my_dyna_fit)

## We can compare the data and the fitted curve
```

```
plot(my_dyna_fit)

## We can plot any environmental condition using add_factor

plot(my_dyna_fit, add_factor = "aw",
    label_y1 = "Log count (log CFU/ml)",
    label_y2 = "Water activity")
```

`fit_growth` *Fitting microbial growth*

Description

[Stable]

This function provides a top-level interface for fitting growth models to data describing the variation of the population size through time, either under constant or dynamic environment conditions. See below for details on the calculations.

Usage

```
fit_growth(
  fit_data,
  model_keys,
  start,
  known,
  environment = "constant",
  algorithm = "regression",
  approach = "single",
  env_conditions = NULL,
  niter = NULL,
  ...,
  check = TRUE,
  logbase_mu = logbase_logN,
  logbase_logN = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	observed microbial growth. The format varies depending on the type of model fit. See the relevant sections (and examples) below for details.
`model_keys`	a named list assigning equations for the primary and secondary models. See the relevant sections (and examples) below for details.

`start`	a named numeric vector assigning initial guesses to the model parameters to estimate from the data. See relevant section (and examples) below for details.
`known`	named numeric vector of fixed model parameters, using the same conventions as for "start".
`environment`	type of environment. Either "constant" (default) or "dynamic" (see below for details on the calculations for each condition)
`algorithm`	either "regression" (default; Levenberg-Marquard algorithm) or "MCMC" (Adaptive Monte Carlo algorithm).
`approach`	approach for model fitting. Either "single" (the model is fitted to a unique experiment) or "global" (the model is fitted to several dynamic experiments).
`env_conditions`	Tibble describing the variation of the environmental conditions for dynamic experiments. See the relevant sections (and examples) below for details. Ignored for environment="constant".
`niter`	number of iterations of the MCMC algorithm. Ignored when algorithm!="MCMC".
`...`	Additional arguments for `FME::modFit()`.
`check`	Whether to check the validity of the models. TRUE by default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	An object of class "formula" defining the names of the x and y variables in the data. `logN ~ time` as a default.

Value

If `approach="single`, an instance of GrowthFit. If `approach="multiple"`, an instance of GlobalGrowthFit

Please check the help pages of each class for additional information.

Fitting under constant conditions

When environment="constant", the functions fits a primary growth model to the population size observed during an experiment. In this case, the data has to be a tibble (or data.frame) with two columns:

- time: the elapsed time
- logN: the logarithm of the observed population size Nonetheless, the names of the columns can be modified with the formula argument.

The model equation is defined through the model_keys argument. It must include an entry named "primary" assigned to a model. Valid model keys can be retrieved calling `primary_model_data()`.

The model is fitted by non-linear regression (using `FME::modFit()`). This algorithm needs initial guesses for every model parameter. This are defined as a named numeric vector. The names must be valid model keys, which can be retrieved using `primary_model_data()` (see example below). Apart from that, any model parameter can be fixed using the "known" argument. This is a named numeric vector, with the same convenctions as "start".

Fitting under dynamic conditions to a single experiment

When environment="constant" and approach="single", a dynamic growth model combining the Baranyi primary growth model with the gamma approach for the effect of the environmental conditions on the growth rate is fitted to an experiment gathered under dynamic conditions. In this case, the data is similar to fitting under constant conditions: a tibble (or data.frame) with two columns:

- time: the elapsed time
- logN: the logarithm of the observed population size Note that these default names can be changed using the formula argument.

The values of the experimental conditions during the experiment are defined using the "env_conditions" argument. It is a tibble (or data.frame) with one column named ("time") defining the elapsed time. Note that this default name can be modified using the formula argument of the function. The tibble needs to have as many additional columns as environmental conditions included in the model, providing the values of the environmental conditions.

The model equations are defined through the model_keys argument. It must be a named list where the names match the column names of "env_conditions" and the values are model keys. These can be retrieved using `secondary_model_data()`.

The model can be fitted using regression (`FME::modFit()`) or an adaptive Monte Carlo algorithm (`FME::modMCMC()`). Both algorithms require initial guesses for every model parameter to fit. These are defined through the named numeric vector "start". Each parameter must be named as *factor*+"_"+*parameter*, where *factor* is the name of the environmental factor defined in "model_keys". The *parameter* is a valid key that can be retrieved from `secondary_model_data()`. For instance, parameter Xmin for the factor temperature would be defined as "temperature_xmin".

Note that the argument ... allows passing additional arguments to the fitting functions.

Fitting under dynamic conditions to multiple experiments (global fitting)

When environment="constant" and approach="global", fit_growth tries to find the vector of model parameters that best describe the observations of several growth experiments.

The input requirements are very similar to the case when approach="single". The models (equations, initial guesses, known parameters, algorithms...) are identical. The only difference is that "fit_data" must be a list, where each element describes the results of an experiment (using the same conventions as when approach="single"). In a similar fashion, "env_conditions" must be a list describing the values of the environmental factors during each experiment. Although it is not mandatory, it is recommended that the elements of both lists are named. Otherwise, the function assigns automatically-generated names, and matches them by order.#'

Examples

```
## Example 1 - Fitting a primary model --------------------------------------

## A dummy dataset describing the variation of the population size

my_data <- data.frame(time = c(0, 25, 50, 75, 100),
                      logN = c(2, 2.5, 7, 8, 8))

## A list of model keys can be gathered from
```

```
primary_model_data()

## The primary model is defined as a list

models <- list(primary = "Baranyi")

## The keys of the model parameters can also be gathered from primary_model_data

primary_model_data("Baranyi")$pars

## Any model parameter can be fixed

known <- c(mu = .2)

## The remaining parameters need initial guesses

start <- c(logNmax = 8, lambda = 25, logN0 = 2)

primary_fit <- fit_growth(my_data, models, start, known,
                          environment = "constant",
                          )

## The instance of FitIsoGrowth includes several useful methods

print(primary_fit)
plot(primary_fit)
coef(primary_fit)
summary(primary_fit)

## time_to_size can be used to calculate the time for some concentration

time_to_size(primary_fit, 4)

## Example 2 - Fitting under dynamic conditions------------------------------

## We will use the example data included in the package

data("example_dynamic_growth")

## And the example environmental conditoins (temperature & aw)

data("example_env_conditions")

## Valid keys for secondary models can be retrived from

secondary_model_data()

## We need to assign a model equation (secondary model) to each environmental factor

sec_models <- list(temperature = "CPM", aw = "CPM")

## The keys of the model parameters can be gathered from the same function
```

```
secondary_model_data("CPM")$pars

## Any model parameter (of the primary or secondary models) can be fixed

known_pars <- list(Nmax = 1e4,  # Primary model
                   N0 = 1e0, Q0 = 1e-3,  # Initial values of the primary model
                   mu_opt = 4, # mu_opt of the gamma model
                   temperature_n = 1,  # Secondary model for temperature
                 aw_xmax = 1, aw_xmin = .9, aw_n = 1  # Secondary model for water activity
                   )

## The rest, need initial guesses (you know, regression)

my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
                 temperature_xmax = 40, aw_xopt = .95)

## We can now fit the model


dynamic_fit <- fit_growth(example_dynamic_growth,
                          sec_models,
                          my_start, known_pars,
                          environment = "dynamic",
                          env_conditions = example_env_conditions
                          )

## The instance of FitDynamicGrowth has several S3 methods

plot(dynamic_fit, add_factor = "temperature")
summary(dynamic_fit)

## We can use time_to_size to calculate the time required to reach a given size

time_to_size(dynamic_fit, 3)



## Example 3- Fitting under dynamic conditions using MCMC -------------------

## We can reuse most of the arguments from the previous example
## We just need to define the algorithm and the number of iterations


set.seed(12421)
MCMC_fit <- fit_growth(example_dynamic_growth,
                       sec_models,
                       my_start, known_pars,
                       environment = "dynamic",
                       env_conditions = example_env_conditions,
                       algorithm = "MCMC",
                       niter = 1000
                       )
```

```
## The instance of FitDynamicGrowthMCMC has several S3 methods

plot(MCMC_fit, add_factor = "aw")
summary(MCMC_fit)

## We can use time_to_size to calculate the time required to reach a given size

time_to_size(MCMC_fit, 3)

## It can also make growth predictions including uncertainty

uncertain_growth <- predictMCMC(MCMC_fit,
                                seq(0, 10, length = 1000),
                                example_env_conditions,
                                niter = 1000)

## The instance of MCMCgrowth includes several nice S3 methods

plot(uncertain_growth)
print(uncertain_growth)

## time_to_size can calculate the time to reach some count

time_to_size(uncertain_growth, 2)
time_to_size(uncertain_growth, 2, type = "distribution")



## Example 4 - Fitting a unique model to several dynamic experiments --------

## We will use the data included in the package

data("multiple_counts")
data("multiple_conditions")

## We need to assign a model equation for each environmental factor

sec_models <- list(temperature = "CPM", pH = "CPM")

## Any model parameter (of the primary or secondary models) can be fixed

known_pars <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
                   temperature_n = 2, temperature_xmin = 20,
                   temperature_xmax = 35,
                   pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

## The rest, need initial guesses

my_start <- list(mu_opt = .8, temperature_xopt = 30)

## We can now fit the model
```

```
global_fit <- fit_growth(multiple_counts,
                         sec_models,
                         my_start,
                         known_pars,
                         environment = "dynamic",
                         algorithm = "regression",
                         approach = "global",
                         env_conditions = multiple_conditions
                         )

## The instance of FitMultipleDynamicGrowth has nice S3 methods

plot(global_fit)
summary(global_fit)
print(global_fit)

## We can use time_to_size to calculate the time to reach a given size

time_to_size(global_fit, 4.5)



## Example 5 - MCMC fitting a unique model to several dynamic experiments ---

## Again, we can re-use all the arguments from the previous example
## We just need to define the right algorithm and the number of iterations
## On top of that, we will also pass upper and lower bounds to modMCMC


set.seed(12421)
global_MCMC <- fit_growth(multiple_counts,
                         sec_models,
                         my_start,
                         known_pars,
                         environment = "dynamic",
                         algorithm = "MCMC",
                         approach = "global",
                         env_conditions = multiple_conditions,
                         niter = 1000,
                         lower = c(.2, 29),  # lower limits of the model parameters
                         upper = c(.8, 34)  # upper limits of the model parameters
                         )

## The instance of FitMultipleDynamicGrowthMCMC has nice S3 methods

plot(global_MCMC)
summary(global_MCMC)
print(global_MCMC)

## We can use time_to_size to calculate the time to reach a given size

time_to_size(global_MCMC, 3)
```

```
## It can also be used to make model predictions with parameter uncertainty

uncertain_prediction <- predictMCMC(global_MCMC,
                                    seq(0, 50, length = 1000),
                                    multiple_conditions[[1]],
                                    niter = 100
                                    )

## The instance of MCMCgrowth includes several nice S3 methods

plot(uncertain_growth)
print(uncertain_growth)

## time_to_size can calculate the time to reach some count

time_to_size(uncertain_growth, 2)
time_to_size(uncertain_growth, 2, type = "distribution")
```

`fit_isothermal_growth` *Fit primary growth models*

Description

[Superseded]

The function `fit_isothermal_growth()` has been superseded by the top-level function `fit_growth()`, which provides a unified approach for growth modelling.

Nonetheless, it can still fit a primary growth model to data obtained under static environmental conditions.

Usage

```
fit_isothermal_growth(
  fit_data,
  model_name,
  starting_point,
  known_pars,
  ...,
  check = TRUE,
  formula = logN ~ time,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`fit_data`	Tibble of data for the fit. It must have two columns, one with the elapsed time (`time` by default) and another one with the decimal logarithm of the populatoin size (`logN` by default). Different column names can be defined using the `formula` argument.
`model_name`	Character defining the primary growth model
`starting_point`	Named vector of initial values for the model parameters.
`known_pars`	Named vector of known model parameters (not fitted).
`...`	Additional arguments passed to `FME::modFit()`.
`check`	Whether to do some basic checks (TRUE by default).
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `FitIsoGrowth()`.

Examples

```
## Some dummy data

library(tibble)

my_data <- tibble(time = c(0, 25, 50, 75, 100),
    logN = c(2, 2.5, 7, 8, 8))

## Choose the model

my_model <- "Baranyi"

## Initial values for the model parameters

start = c(logNmax = 8, lambda = 25, logN0 = 2)

## Any model parameter can be fixed

known <- c(mu = .2)

## Now, we can call the function

static_fit <- fit_isothermal_growth(my_data, my_model, start, known)

summary(static_fit)
```

```
## We can plot the fitted model against the observations

plot(static_fit)
```

`fit_MCMC_growth` *Fit growth models using MCMC*

Description

[Superseded]

The function `fit_MCMC_growth()` has been superseded by the top-level function `fit_growth()`, which provides a unified approach for growth modelling.

But, it can still fit a growth model to a data obtained under dynamic conditions using the one-step approach (MCMC algorithm).

Usage

```
fit_MCMC_growth(
  fit_data,
  env_conditions,
  starting_point,
  known_pars,
  sec_model_names,
  niter,
  ...,
  check = TRUE,
  formula = logN ~ time,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`fit_data`	Tibble with the data to use for model fit. It must contain a column with the elapsed time (named "time" by default) and another one with the decimal logarithm of the observed population size (named "logN" by default). Different column names can be specified using the "formula" argument.
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column with the elapsed time (named "time" by default) and as many columns as required with the environmental conditions. A different column name can be specified using the "formula" argument, although it must be the same one as in "fit_data". Note that only those defined in "sec_model_names" will be considered for the model fit.

`starting_point`	A named vector of starting values for the model parameters. Parameters for the primary model must be named in the usual way. Parameters for the secondary model are named as `env_factor`+'_'+`parameter`. For instance, the maximum growth temperature shall be named 'temperature_xmax'.
`known_pars`	A named vector of known model parameters (i.e. not fitted). They must be named using the same convention as for `starting_point`.
`sec_model_names`	A named character vector defining the secondary model for each environmental factor. The names define the factor and the value the type of model. Names must match columns in `fit_data` and `env_conditions`.
`niter`	number of iterations of the MCMC algorithm.
`...`	Additional arguments passed to modFit.
`check`	Whether to check model parameters (TRUE by default).
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `FitDynamicGrowthMCMC()`.

Examples

```
## We use the example data included in the package

data("example_dynamic_growth")
data("example_env_conditions")

## Definition of the secondary models
sec_model_names <- c(temperature = "CPM", aw= "CPM")

## Any model parameter can be fixed
known_pars <- list(Nmax = 1e4,  # Primary model
    N0 = 1e0, Q0 = 1e-3,  # Initial values of the primary model
    mu_opt = 4, # mu_opt of the gamma model
    temperature_n = 1,  # Secondary model for temperature
    aw_xmax = 1, aw_xmin = .9, aw_n = 1  # Secondary model for water activity
    )

## We need starting values for the remaining parameters

my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
    temperature_xmax = 40,
    aw_xopt = .95)

## We can now call the fitting function
```

```
set.seed(12124) # Setting seed for repeatability

my_MCMC_fit <- fit_MCMC_growth(example_dynamic_growth, example_env_conditions,
    my_start, known_pars, sec_model_names, niter = 3000)

## Always check the MCMC chain!!

plot(my_MCMC_fit$fit_results)

## We can compare data against fitted curve

plot(my_MCMC_fit)

## Any environmental factor can be included using add_factor

plot(my_MCMC_fit, add_factor = "temperature",
    label_y1 = "Count (log CFU/ml)", label_y2 = "Temperature (C)")
```

`fit_multiple_growth` *Fitting growth models to multiple dynamic experiments*

Description

[Superseded]

The function `fit_multiple_growth()` has been superseded by the top-level function `fit_growth()`, which provides a unified approach for growth modelling.

But, if you so wish, this function still enables fitting a growth model using a dataset comprised of several experiments with potentially different dynamic experimental conditions. Note that the definition of secondary models must comply with the `secondary_model_data` function.

Usage

```
fit_multiple_growth(
  starting_point,
  experiment_data,
  known_pars,
  sec_model_names,
  ...,
  check = TRUE,
  formula = logN ~ time,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`starting_point`	a named vector of starting values for the model parameters.
`experiment_data`	a nested list with the experimental data. Each entry describes one experiment as a list with two elements: data and conditions. `data` is a tibble with a column giving the elapsed time (named "time" by default) and another one with the decimal logarithm of the population size (named "logN" by default). `conditions` is a tibble with one column giving the elapsed time (using the same name as `data`) and as many additional columns as environmental factors. The default column names can be changed with the formula argument.
`known_pars`	named vector of known model parameters
`sec_model_names`	named character vector with names of the environmental conditions and values of the secondary model (see secondary_model_data).
`...`	additional arguments for `FME::modFit()`.
`check`	Whether to check the validity of the models. `TRUE` by default.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `FitMultipleDynamicGrowth()`.

Examples

```
## We will use the multiple_experiments data set

data("multiple_experiments")

## For each environmental factor, we need to defined a model

sec_names <- c(temperature = "CPM", pH = "CPM")

## Any model parameter can be fixed

known <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
    temperature_n = 2, temperature_xmin = 20, temperature_xmax = 35,
    pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

## The rest require starting values for model fitting

start <- list(mu_opt = .8, temperature_xopt = 30)

## We can now call the fitting function
```

```
global_fit <- fit_multiple_growth(start, multiple_experiments, known, sec_names)

## Parameter estimates can be retrieved with summary

summary(global_fit)

## We can compare fitted model against observations

plot(global_fit)

## Any single environmental factor can be added to the plot using add_factor

plot(global_fit, add_factor = "temperature")
```

`fit_multiple_growth_MCMC`

Fitting growth models to multiple dynamic experiments using MCMC

Description

[Superseded]

The function `fit_multiple_growth_MCMC()` has been superseded by the top-level function `fit_growth()`, which provides a unified approach for growth modelling.

However, this functions can still be used to fit a growth model using a dataset comprised of several experiments with potentially different dynamic experimental conditions.

Usage

```
fit_multiple_growth_MCMC(
  starting_point,
  experiment_data,
  known_pars,
  sec_model_names,
  niter,
  ...,
  check = TRUE,
  formula = logN ~ time,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`starting_point`	a named vector of starting values for the model parameters.

`experiment_data`	a nested list with the experimental data. Each entry describes one experiment as a list with two elements: data and conditions. `data` is a tibble with a column giving the elapsed time (named "time" by default) and another one with the decimal logarithm of the population size (named "logN" by default). `conditions` is a tibble with one column giving the elapsed time (using the same name as `data`) and as many additional columns as environmental factors. The default column names can be changed with the formula argument.
`known_pars`	named vector of known model parameters
`sec_model_names`	named character vector with names of the environmental conditions and values of the secondary model (see secondary_model_data).
`niter`	number of samples of the MCMC algorithm.
`...`	additional arguments for FME::modMCMC (e.g. upper and lower bounds).
`check`	Whether to check the validity of the models. `TRUE` by default.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `FitMultipleGrowthMCMC()`.

Examples

```
## We will use the multiple_experiments data set

data("multiple_experiments")

## For each environmental factor, we need to defined a model

sec_names <- c(temperature = "CPM", pH = "CPM")

## Any model parameter can be fixed

known <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
    temperature_n = 2, temperature_xmin = 20, temperature_xmax = 35,
    pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

## The rest require starting values for model fitting

start <- list(mu_opt = .8, temperature_xopt = 30)

## We can now call the fitting function

set.seed(12412)
```

```
global_MCMC <- fit_multiple_growth_MCMC(start, multiple_experiments, known, sec_names, niter = 1000,
   lower = c(.2, 29),  # lower limits of the model parameters
   upper = c(.8, 34))  # upper limits of the model parameters

## Parameter estimates can be retrieved with summary

summary(global_MCMC)

## We can compare fitted model against observations

plot(global_MCMC)

## Any single environmental factor can be added to the plot using add_factor

plot(global_MCMC, add_factor = "temperature")
```

`fit_secondary_growth` *Fit secondary growth models*

Description

[Stable]

Fits a secondary growth model to a set of growth rates obtained experimentally. Modelling is done according to the gamma concept proposed by Zwietering (1992) and cardinal parameter models.

Usage

```
fit_secondary_growth(
  fit_data,
  starting_point,
  known_pars,
  sec_model_names,
  transformation = "sq",
  ...,
  check = TRUE,
  formula = mu ~ .
)
```

Arguments

`fit_data`	Tibble with the data used for the fit. It must have one column with the observed growth rate (named `mu` by default; can be changed using the "formula" argument) and as many columns as needed with the environmental factors.
`starting_point`	Named vector with initial values for the model parameters to estimate from the data. The growth rate under optimum conditions must be named `mu_opt`. The rest must be called 'env_factor'+'_'+'parameter'. For instance, the minimum pH for growth is 'pH_xmin'.

`known_pars`	Named vector of fixed model parameters. Must be named using the same convention as `starting_point`.
`sec_model_names`	Named character vector defining the secondary model for each environmental factor.
`transformation`	Character defining the transformation of `mu` for model fitting. One of `sq` (square root; default), `log` (log-transform) or `none` (no transformation).
`...`	Additional arguments passed to `FME::modFit()`.
`check`	Whether to do some basic checks (TRUE by default).
`formula`	an object of class "formula" describing the y variable. The right hand side must be ".". By default `mu ~ .`.

Value

An instance of `FitSecondaryGrowth()`.

Examples

```
## We use the data included in the package

data("example_cardinal")

## Define the models to fit

sec_model_names <- c(temperature = "Zwietering", pH = "CPM")

## Any model parameter can be fixed

known_pars <- list(mu_opt = 1.2, temperature_n = 1,
    pH_n = 2, pH_xmax = 6.8, pH_xmin = 5.2)

## Initial values must be given for every other parameter

my_start <- list(temperature_xmin = 5, temperature_xopt = 35,
    pH_xopt = 6.5)

## We can now call the fitting function

fit_cardinal <- fit_secondary_growth(example_cardinal, my_start, known_pars, sec_model_names)

## With summary, we can look at the parameter estimates

summary(fit_cardinal)

## The plot function compares predictions against observations

plot(fit_cardinal)

## Passing which = 2, generates a different kind of plot
```

```
plot(fit_cardinal, which = 2)
plot(fit_cardinal, which = 2, add_trend = TRUE)
plot(fit_cardinal, which = 2, add_segment = TRUE)
```

`full_Ratkowski` *Full Ratkowsky model*

Description

Gamma model adapted from the one by Ratkowsky et al. (1983).

Usage

```
full_Ratkowski(x, xmin, xmax, c)
```

Arguments

`x`	Value of the environmental factor.
`xmin`	Minimum value for growth
`xmax`	Maximum value for growth
`c`	Parameter defining the speed of the decline

`get_all_predictions` *A helper for making the plots*

Description

A helper for making the plots

Usage

```
get_all_predictions(model)
```

Arguments

`model`	An instance of FitMultipleDynamicGrowth

`get_dyna_residuals` *Residuals of dynamic prediction*

Description

Function for calculating residuals of a dynamic prediction according to the requirements of `FME::modFit()`.

Usage

```
get_dyna_residuals(
  this_p,
  fit_data,
  env_conditions,
  known_pars,
  sec_model_names,
  cost = NULL,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`this_p`	named vector of model parameters
`fit_data`	tibble with the data for the fit
`env_conditions`	tibble with the environmental conditions
`known_pars`	named vector of known model parameters
`sec_model_names`	named character vector with names the environmental conditions and values the secondary model (e.g. 'CPM').
`cost`	an instance of modCost to be combined (to fit multiple models).
`logbase_mu`	Base of the logarithm of the growthrate. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `FME::modCost()`.

`get_iso_residuals` *Residuals of isothermal prediction*

Description

Residuals of isothermal prediction

Usage

```
get_iso_residuals(
  this_p,
  fit_data,
  model_name,
  known_pars,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`this_p`	named vector of model parameters to fit
`fit_data`	tibble with the data for the fit
`model_name`	character defining the primary growth model
`known_pars`	named vector of fixed model parameters
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `modCost`.

`get_multi_dyna_residuals` *Residuals of multiple dynamic predictions*

Description

Function for calculating residuals of dynamic predictions under different conditions for the same model parameters according to the requirements of `FME::modFit()`.

Usage

```
get_multi_dyna_residuals(
  this_p,
  experiment_data,
  known_pars,
  sec_model_names,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`this_p`	named vector of model parameters
`experiment_data`	a nested list with the experimental data. Each entry describes one experiment as a list with two elements: data and conditions. `data` is a tibble with two columns: time and logN. `conditions` is a tibble with one column named time and as many additional columns as environmental factors.
`known_pars`	named vector of known model parameters
`sec_model_names`	named character vector with names of the environmental conditions and values of the secondary model (see secondary_model_data).
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

an instance of `modCost`.

`get_secondary_residuals`

Residuals of secondary models

Description

Residual function for `fit_secondary_growth()`.

Usage

```
get_secondary_residuals(
  this_p,
  my_data,
  known_pars,
  sec_model_names,
  transformation
)
```

Arguments

`this_p`	Named vector of model parameter values.
`my_data`	Tibble with the data used for the fit.
`known_pars`	Named vector of fixed model paramaters.
`sec_model_names`	Named character vector defining the secondary model for each environmental factor.
`transformation`	Character defining the tranformation of `mu` for model fitting. One of `sq` (square root), `log` (log-transform) or `none` (no transformation).

Value

A numeric vector of residuals.

`GlobalGrowthComparison`

GlobalGrowthComparison class

Description

The `GlobalGrowthComparison` class contains several functions for model comparison and model selection of growth models. It should not be instanced directly. Instead, it should be constructed using `compare_growth_fits()`. It is similar to GrowthComparison, although with specific tools to deal with several experiments.

It includes two type of tools for model selection and comparison: statistical indexes and visual analyses. Please check the sections below for details.

Note that all these tools use the names defined in `compare_growth_fits()`, so we recommend passing a named list to that function.

Usage

```
## S3 method for class 'GlobalGrowthComparison'
coef(object, ...)

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

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

## S3 method for class 'GlobalGrowthComparison'
plot(x, y, ..., type = 1, add_trend = TRUE)
```

Arguments

`object`	an instance of GlobalGrowthComparison
`...`	ignored
`x`	an instance of GlobalGrowthComparison
`y`	ignored
`type`	if type==1, the plot compares the model predictions. If type ==2, the plot compares the parameter estimates. If type==3, the plot shows the residuals
`add_trend`	should a trend line of the residuals be added for type==3? `TRUE` by default

Methods (by generic)

- `coef(GlobalGrowthComparison)`: table of parameter estimates
- `summary(GlobalGrowthComparison)`: summary table for the comparison
- `print(GlobalGrowthComparison)`: print of the model comparison
- `plot(GlobalGrowthComparison)`: illustrations comparing the fitted models

Statistical indexes

`GlobalGrowthComparison` implements two S3 methods to obtain numerical values to facilitate model comparison and selection.

- the `coef` method returns a tibble with the values of the parameter estimates and their corresponding standard errors for each model.
- the `summary` returns a tibble with the AIC, number of degrees of freedom, mean error and root mean squared error for each model.

Visual analyses

The S3 plot method can generate three types of plots:

- when `type = 1`, the plot compares the fitted growth curves against the experimental data used to fit the model.
- when `type = 2`, the plot compares the parameter estimates using error bars, where the limits of the error bars are the expected value +/- one standard error. In case one model does not has some model parameter (i.e. either because it is not defined or because it was fixed), the parameter is not included in the plot.
- when `type=3`, the plot shows the tendency of the residuals for each model. This plot can be used to detect deviations from independence.

These plots are divided by facets for each experiment.

`GlobalGrowthFit` *GlobalGrowthFit class*

Description

[Stable]

The `GlobalGrowthFit` class contains a growth model fitted to data using a global approach. Its constructor is `fit_growth()`.

It is a subclass of list with the items:

- algorithm: type of algorithm as in `fit_growth()`
- data: data used for model fitting
- start: initial guess of the model parameters
- known: fixed model parameters
- primary_model: a character describing the primary model
- fit_results: an instance of modFit or modMCMC with the results of the fit
- best_prediction: Instance of GrowthPrediction with the best growth fit
- sec_models: a named vector with the secondary models assigned for each environmental factor. `NULL` for `environment="constant"`
- env_conditions: a list with the environmental conditions used for model fitting. `NULL` for `environment="constant"`
- niter: number of iterations of the Markov chain. `NULL` if `algorithm != "MCMC"`
- logbase_mu: base of the logarithm for the definition of parameter mu (check the relevant vignette)
- logbase_logN: base of the logarithm for the definition of the population size (check the relevant vignette)
- environment: "dynamic". Always

Usage

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

## S3 method for class 'GlobalGrowthFit'
coef(object, ...)

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

## S3 method for class 'GlobalGrowthFit'
predict(object, env_conditions, times = NULL, ...)

## S3 method for class 'GlobalGrowthFit'
```

```
residuals(object, ...)

## S3 method for class 'GlobalGrowthFit'
vcov(object, ...)

## S3 method for class 'GlobalGrowthFit'
deviance(object, ...)

## S3 method for class 'GlobalGrowthFit'
fitted(object, ...)

## S3 method for class 'GlobalGrowthFit'
logLik(object, ...)

## S3 method for class 'GlobalGrowthFit'
AIC(object, ..., k = 2)

## S3 method for class 'GlobalGrowthFit'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_x = "time",
  label_y1 = NULL,
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed",
  point_size = 3,
  point_shape = 16,
  subplot_labels = "AUTO"
)

## S3 method for class 'GlobalGrowthFit'
predictMCMC(
  model,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`x`	an instance of GlobalGrowthFit
`...`	ignored
`object`	an instance of GlobalGrowthFit
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column named 'time' with the storage time and as many columns as required with the environmental conditions.
`times`	Numeric vector of storage times for the predictions.
`k`	penalty for the parameters (k=2 by default)
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_x`	label of the x-axis
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`point_size`	Size of the data points
`point_shape`	shape of the data points
`subplot_labels`	labels of the subplots according to `plot_grid`.
`model`	An instance of GlobalGrowthFit
`niter`	Number of iterations.
`newpars`	A named list defining new values for the some model parameters. The name must be the identifier of a model already included in the model. These parameters do not include variation, so defining a new value for a fitted parameters "fixes" it. `NULL` by default (no new parameters).
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

Value

An instance of MCMCgrowth.

Methods (by generic)

- `print(GlobalGrowthFit)`: print of the model
- `coef(GlobalGrowthFit)`: vector of fitted model parameters.
- `summary(GlobalGrowthFit)`: statistical summary of the fit.
- `predict(GlobalGrowthFit)`: vector of model predictions
- `residuals(GlobalGrowthFit)`: model residuals. They are returned as a tibble with 4 columns: time (storage time), logN (observed count), exp (name of the experiment) and res (residual).
- `vcov(GlobalGrowthFit)`: variance-covariance matrix of the model, estimated as 1/(0.5*Hessian) for regression and as the variance-covariance of the draws for MCMC
- `deviance(GlobalGrowthFit)`: deviance of the model.
- `fitted(GlobalGrowthFit)`: fitted values. They are returned as a tibble with 3 columns: time (storage time), exp (experiment identifier) and fitted (fitted value).
- `logLik(GlobalGrowthFit)`: loglikelihood of the model
- `AIC(GlobalGrowthFit)`: Akaike Information Criterion
- `plot(GlobalGrowthFit)`: comparison between the fitted model and the experimental data.
- `predictMCMC(GlobalGrowthFit)`: prediction including parameter uncertainty

`greek_tractors` *Number of tractors in Greece according to the World Bank*

Description

A dataset showing the increase in tractors in Greece. It was retrieved from https://data.worldbank.org/indicator/AG.AGR.TRA

Usage

```
greek_tractors
```

Format

A tibble with 46 rows (each corresponding to one year) and 7 columns:

year Year for the recording

tractors Number of tractors

`GrowthComparison` *GrowthComparison class*

Description

The `GrowthComparison` class contains several functions for model comparison and model selection of growth models. It should not be instanced directly. Instead, it should be constructed using `compare_growth_fits()`.

It includes two type of tools for model selection and comparison: statistical indexes and visual analyses. Please check the sections below for details.

Note that all these tools use the names defined in `compare_growth_fits()`, so we recommend passing a named list to that function.

Usage

```
## S3 method for class 'GrowthComparison'
plot(x, y, ..., type = 1, add_trend = TRUE)

## S3 method for class 'GrowthComparison'
coef(object, ...)

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

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

Arguments

`x`	an instance of GrowthComparison
`y`	ignored
`...`	ignored
`type`	if type==1, the plot compares the model predictions. If type ==2, the plot compares the parameter estimates. If type==3, the plot shows the residuals
`add_trend`	should a trend line of the residuals be added for type==3? `TRUE` by default
`object`	an instance of GrowthComparison

Methods (by generic)

- `plot(GrowthComparison)`: illustrations comparing the fitted models
- `coef(GrowthComparison)`: table of parameter estimates
- `print(GrowthComparison)`: print of the model comparison
- `summary(GrowthComparison)`: summary table for the comparison

Statistical indexes

`GrowthComparison` implements two S3 methods to obtain numerical values to facilitate model comparison and selection.

- the `coef` method returns a tibble with the values of the parameter estimates and their corresponding standard errors for each model.
- the `summary` returns a tibble with the AIC, number of degrees of freedom, mean error and root mean squared error for each model.

Visual analyses

The S3 plot method can generate three types of plots:

- when `type = 1`, the plot compares the fitted growth curves against the experimental data used to fit the model.
- when `type = 2`, the plot compares the parameter estimates using error bars, where the limits of the error bars are the expected value +/- one standard error. In case one model does not have some model parameter (i.e. either because it is not defined or because it was fixed), the parameter is not included in the plot.
- when `type=3`, the plot shows the tendency of the residuals for each model. This plot can be used to detect deviations from independence.

`GrowthFit` *GrowthFit class*

Description

[Stable]

The `GrowthFit` class contains a growth model fitted to data under static or dynamic conditions. Its constructor is `fit_growth()`.

It is a subclass of list with the items:

- environment: type of environment as in `fit_growth()`
- algorithm: type of algorithm as in `fit_growth()`
- data: data used for model fitting
- start: initial guess of the model parameters
- known: fixed model parameters
- primary_model: a character describing the primary model
- fit_results: an instance of modFit or modMCMC with the results of the fit
- best_prediction: Instance of GrowthPrediction with the best growth fit
- sec_models: a named vector with the secondary models assigned for each environmental factor. `NULL` for `environment="constant"`
- env_conditions: a tibble with the environmental conditions used for model fitting. `NULL` for `environment="constant"`

- niter: number of iterations of the Markov chain. `NULL` if `algorithm != "MCMC"`
- logbase_mu: base of the logarithm for the definition of parameter mu (check the relevant vignette)
- logbase_logN: base of the logarithm for the definition of the population size (check the relevant vignette)

Usage

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

## S3 method for class 'GrowthFit'
coef(object, ...)

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

## S3 method for class 'GrowthFit'
predict(object, times = NULL, env_conditions = NULL, ...)

## S3 method for class 'GrowthFit'
residuals(object, ...)

## S3 method for class 'GrowthFit'
vcov(object, ...)

## S3 method for class 'GrowthFit'
deviance(object, ...)

## S3 method for class 'GrowthFit'
fitted(object, ...)

## S3 method for class 'GrowthFit'
logLik(object, ...)

## S3 method for class 'GrowthFit'
AIC(object, ..., k = 2)

## S3 method for class 'GrowthFit'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  line_col = "black",
  line_size = 1,
  line_type = 1,
  point_col = "black",
```

```
  point_size = 3,
  point_shape = 16,
  ylims = NULL,
  label_y1 = NULL,
  label_y2 = add_factor,
  label_x = "time",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed"
)

## S3 method for class 'GrowthFit'
predictMCMC(
  model,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`x`	The object of class GrowthFit to plot.
`...`	ignored.
`object`	an instance of GrowthFit
`times`	Numeric vector of storage times for the predictions.
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column named 'time' with the storage time and as many columns as required with the environmental conditions.
`k`	penalty for the parameters (k=2 by default)
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis. Ignored if `environment="constant"`
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`point_col`	Aesthetic parameter to change the colour of the point geom, see: `ggplot2::geom_point()`
`point_size`	Aesthetic parameter to change the size of the point geom, see: `ggplot2::geom_point()`
`point_shape`	Aesthetic parameter to change the shape of the point geom, see: `ggplot2::geom_point()`

`ylims`	A two dimensional vector with the limits of the primary y-axis. `NULL` by default
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis. Ignored if `environment="constant"`
`label_x`	Label of the x-axis
`line_col2`	Same as lin_col, but for the environmental factor. Ignored if `environment="constant"`
`line_size2`	Same as line_size, but for the environmental factor. Ignored if `environment="constant"`
`line_type2`	Same as lin_type, but for the environmental factor. Ignored if `environment="constant"`
`model`	An instance of GrowthFit
`niter`	Number of iterations.
`newpars`	A named list defining new values for the some model parameters. The name must be the identifier of a model already included in the model. These parameters do not include variation, so defining a new value for a fitted parameters "fixes" it. `NULL` by default (no new parameters).
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

Value

An instance of MCMCgrowth.

Methods (by generic)

- `print(GrowthFit)`: print of the model
- `coef(GrowthFit)`: vector of fitted model parameters.
- `summary(GrowthFit)`: statistical summary of the fit.
- `predict(GrowthFit)`: vector of model predictions.
- `residuals(GrowthFit)`: vector of model residuals.
- `vcov(GrowthFit)`: variance-covariance matrix of the model, estimated as 1/(0.5*Hessian) for regression and as the variance-covariance of the draws for MCMC
- `deviance(GrowthFit)`: deviance of the model.
- `fitted(GrowthFit)`: vector of fitted values.
- `logLik(GrowthFit)`: loglikelihood of the model
- `AIC(GrowthFit)`: Akaike Information Criterion
- `plot(GrowthFit)`: compares the fitted model against the data.
- `predictMCMC(GrowthFit)`: prediction including parameter uncertainty

`GrowthPrediction` *GrowthPrediction class*

Description

[Stable]

The `GrowthPrediction` class contains the results of a growth prediction. Its constructor is `predict_growth()`.

It is a subclass of list with the items:

- simulation: a tibble with the model simulation
- primary model: a list describing the primary model as in `predict_growth()`
- environment: a character describing the type of environmental conditions as in `predict_growth()`
- env_conditions: a named list with the functions used to approximate the (dynamic) environmental conditions. `NULL` if `environment="constant"`.
- sec_models: a named list describing the secondary models as in `predict_growth()`. `NULL` if `environment="constant"`.
- gammas: a tibble describing the variation of the gamma factors through the experiment. `NUll` if `environment="constant"`.
- logbase_mu: the log-base for the definition of parameter mu (see the relevant vignette)
- logbase_logN: the log-base for the definition of the logarithm of the population size

Usage

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

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

## S3 method for class 'GrowthPrediction'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  ylims = NULL,
  label_y1 = NULL,
  label_y2 = add_factor,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  line_col2 = "black",
  line_size2 = 1,
  line_type2 = "dashed",
```

```
  label_x = "time"
)

## S3 method for class 'GrowthPrediction'
coef(object, ...)
```

Arguments

`x`	The object of class `GrowthPrediction` to plot.
`...`	ignored
`object`	an instance of GrowthPrediction
`y`	ignored
`add_factor`	whether to plot also one environmental factor. If `NULL` (default), no environmental factor is plotted. If set to one character string that matches one entry of x$env_conditions, that condition is plotted in the secondary axis. Ignored for `environment="constant"`.
`ylims`	A two dimensional vector with the limits of the primary y-axis.
`label_y1`	Label of the primary y-axis.
`label_y2`	Label of the secondary y-axis.
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`line_col2`	Same as lin_col, but for the environmental factor.
`line_size2`	Same as line_size, but for the environmental factor.
`line_type2`	Same as lin_type, but for the environmental factor.
`label_x`	Label of the x-axis.

Methods (by generic)

- `print(GrowthPrediction)`: print of the model
- `summary(GrowthPrediction)`: summary of the model
- `plot(GrowthPrediction)`: predicted growth curve.
- `coef(GrowthPrediction)`: coefficients of the model

`GrowthUncertainty` *GrowthUncertainty class*

Description

[Stable]

The `GrowthUncertainty` class contains the results of a growth prediction under isothermal conditions considering parameter uncertainty. Its constructor is `predict_growth_uncertainty()`.

It is a subclass of list with the items:

- sample: parameter sample used for the calculations.
- simulations: growth curves predicted for each parameter.
- quantiles: limits of the credible intervals (5%, 10%, 50%, 90%, 95%) for each time point.
- model: Model used for the calculations.
- mus: Mean parameter values used for the simulations.
- sigma: Variance-covariance matrix used for the simulations.
- logbase_mu: base of the logarithm for the definition of parameter mu (check the relevant vignette)
- logbase_logN: base of the logarithm for the definition of the population size (check the relevant vignette)

Usage

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

## S3 method for class 'GrowthUncertainty'
plot(
  x,
  y = NULL,
  ...,
  line_col = "black",
  line_size = 0.5,
  line_type = "solid",
  ribbon80_fill = "grey",
  ribbon90_fill = "grey",
  alpha80 = 0.5,
  alpha90 = 0.4
)
```

Arguments

`x`	The object of class `GrowthUncertainty` to plot.
`...`	ignored.

`y`	ignored
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`ribbon80_fill`	fill colour for the space between the 10th and 90th quantile, see: `ggplot2::geom_ribbon()`
`ribbon90_fill`	fill colour for the space between the 5th and 95th quantile, see: `ggplot2::geom_ribbon()`
`alpha80`	transparency of the ribbon aesthetic for the space between the 10th and 90th quantile. Takes a value between 0 (fully transparant) and 1 (fully opaque)
`alpha90`	transparency of the ribbon aesthetic for the space between the 5th and 95th quantile. Takes a value between 0 (fully transparant) and 1 (fully opaque).

Methods (by generic)

- `print(GrowthUncertainty)`: print of the model
- `plot(GrowthUncertainty)`: Growth prediction (prediction band) considering parameter uncertainty.

`growth_pH_temperature` *Example of dynamic growth*

Description

A dataset to demonstrate the use of fit_dynamic_growth. The values of the environmental conditions are described in conditions_pH_temperature.

Usage

```
growth_pH_temperature
```

Format

A tibble with 20 rows and 2 columns:

time elapsed time

logN decimal logarithm of the population size

`growth_salmonella` *Growth of Salmonella spp in broth*

Description

An example dataset to illustrate `fit_isothermal_growth()`. It describes the growth of Salmonella spp. in broth. It was retrieved from ComBase (ID: B092_10).

Usage

```
growth_salmonella
```

Format

A tibble with 21 rows and 2 columns:

time elapsed time in hours.

logN observed population size (log CFU/g).

`inhibitory_model` *Secondary model for inhibitory compounds*

Description

Secondary model for the effect of inhibitory compounds.

Usage

```
inhibitory_model(x, MIC, alpha)
```

Arguments

`x`	Value of the environmental factor (in principle, concentration of compound).
`MIC`	Minimum Inhibitory Concentration
`alpha`	shape factor of the miodel

Value

The corresponding gamma factor.

`is.DynamicGrowth` *Test of DynamicGrowth object*

Description

Tests if an object is of class `DynamicGrowth`.

Usage

```
is.DynamicGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `DynamicGrowth`

`is.FitDynamicGrowth` *Test of FitDynamicGrowth object*

Description

Tests if an object is of class `FitDynamicGrowth`.

Usage

```
is.FitDynamicGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitDynamicGrowth`

`is.FitDynamicGrowthMCMC` *Test of FitDynamicGrowthMCMC object*

Description

Tests if an object is of class `FitDynamicGrowthMCMC`.

Usage

```
is.FitDynamicGrowthMCMC(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitDynamicGrowthMCMC`

`is.FitIsoGrowth` *Test of FitIsoGrowth object*

Description

Tests if an object is of class `FitIsoGrowth`.

Usage

```
is.FitIsoGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitIsoGrowth`

`is.FitMultipleDynamicGrowth`

Test of FitMultipleDynamicGrowth object

Description

Tests if an object is of class `FitMultipleDynamicGrowth`.

Usage

```
is.FitMultipleDynamicGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitMultipleDynamicGrowth`

`is.FitMultipleDynamicGrowthMCMC`

Test of FitMultipleDynamicGrowthMCMC object

Description

Tests if an object is of class `FitMultipleDynamicGrowthMCMC`.

Usage

```
is.FitMultipleDynamicGrowthMCMC(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitMultipleDynamicGrowthMCMC`

`is.FitSecondaryGrowth` *Test of FitSecondaryGrowth object*

Description

Tests if an object is of class `FitSecondaryGrowth`.

Usage

```
is.FitSecondaryGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `FitSecondaryGrowth`

`is.GlobalGrowthFit` *Test of GlobalGrowthFit object*

Description

Tests if an object is of class GlobalGrowthFit

Usage

```
is.GlobalGrowthFit(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class GlobalGrowthFit

`is.GrowthFit` *Test of GrowthFit object*

Description

Tests if an object is of class GrowthFit

Usage

```
is.GrowthFit(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class GrowthFit

`is.GrowthPrediction` *Test of GrowthPrediction object*

Description

Tests if an object is of class GrowthPrediction

Usage

```
is.GrowthPrediction(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class GrowthPrediction

---

`is.GrowthUncertainty` *Test of GrowthUncertainty object*

Description

Tests if an object is of class GrowthUncertainty

Usage

```
is.GrowthUncertainty(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class GrowthUncertainty

---

`is.IsothermalGrowth` *Test of IsothermalGrowth object*

Description

Tests if an object is of class `IsothermalGrowth`.

Usage

```
is.IsothermalGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `IsothermalGrowth`

`is.MCMCgrowth` *Test of MCMCgrowth object*

Description

Tests if an object is of class `MCMCgrowth`.

Usage

```
is.MCMCgrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `MCMCgrowth`

`is.StochasticGrowth` *Test of StochasticGrowth object*

Description

Tests if an object is of class `StochasticGrowth`.

Usage

```
is.StochasticGrowth(x)
```

Arguments

`x`	object to be checked.

Value

A boolean specifying whether `x` is of class `StochasticGrowth`

`IsothermalGrowth` *IsothermalGrowth class*

Description

[Superseded]

The class IsothermalGrowth has been superseded by the top-level class GrowthPrediction, which provides a unified approach for growth modelling.

Still, it is still returned if the superseded `predict_isothermal_growth()` is called.

It is a subclass of list with the items:

- simulation: A tibble with the model simulation.
- model: The name of the model used for the predictions.
- pars: A list with the values of the model parameters.

Usage

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

## S3 method for class 'IsothermalGrowth'
plot(
  x,
  y = NULL,
  ...,
  line_col = "black",
  line_size = 1,
  line_type = "solid",
  ylims = NULL,
  label_y = NULL,
  label_x = "time"
)

## S3 method for class 'IsothermalGrowth'
coef(object, ...)
```

Arguments

`x`	The object of class IsothermalGrowth to plot.
`...`	ignored
`y`	ignored
`line_col`	Aesthetic parameter to change the colour of the line, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line, see: `ggplot2::geom_line()`
`line_type`	Aesthetic parameter to change the type of the line, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`

`ylims`	Two-dimensional numeric vector with the limits of the y-axis (or `NULL`, which is the default)
`label_y`	Title of the y-axis
`label_x`	Title of the x-axis
`object`	an instance of IsothermalGrowth

Methods (by generic)

- `print(IsothermalGrowth)`: print of the model
- `plot(IsothermalGrowth)`: plot of the predicted growth curve.
- `coef(IsothermalGrowth)`: coefficients of the model

`iso_Baranyi` *Isothermal Baranyi model*

Description

Baranyi growth model as defined by Baranyi and Roberts (1994). We use the solution calculated by Poschet et al. (2005, doi: https://doi.org/10.1016/j.ijfoodmicro.2004.10.008) after log-transformation according to MONTE CARLO ANALYSIS FOR MICROBIAL GROWTH CURVES, by Oksuz and Buzrul.

Usage

```
iso_Baranyi(times, logN0, mu, lambda, logNmax)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration
`logNmax`	Maximum log microbial count

Value

Numeric vector with the predicted microbial count.

`iso_Baranyi_noLag` *Isothermal Baranyi model without lag phase*

Description

Baranyi growth model as defined by Baranyi and Roberts (1994). We use the solution calculated by Poschet et al. (2005, doi: https://doi.org/10.1016/j.ijfoodmicro.2004.10.008) after log-transformation according to MONTE CARLO ANALYSIS FOR MICROBIAL GROWTH CURVES, by Oksuz and Buzrul.

Usage

```
iso_Baranyi_noLag(times, logN0, mu, logNmax)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`logNmax`	Maximum log microbial count

Value

Numeric vector with the predicted microbial count.

`iso_Baranyi_noStat` *Isothermal Baranyi model without stationary phase*

Description

Baranyi growth model as defined by Baranyi and Roberts (1994). We use the solution calculated by Poschet et al. (2005, doi: https://doi.org/10.1016/j.ijfoodmicro.2004.10.008) after log-transformation according to MONTE CARLO ANALYSIS FOR MICROBIAL GROWTH CURVES, by Oksuz and Buzrul.

Usage

```
iso_Baranyi_noStat(times, logN0, mu, lambda)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration

Value

Numeric vector with the predicted microbial count.

`iso_repGompertz` *Reparameterized Gompertz model*

Description

Reparameterized Gompertz growth model defined by Zwietering et al. (1990).

Usage

```
iso_repGompertz(times, logN0, C, mu, lambda)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`C`	Difference between `logN0` and the maximum log-count.
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration

Value

Numeric vector with the predicted microbial count.

`lambda_to_Q0` *Q0 from lag phase duration*

Description

[Stable]

Convenience function to calculate the value of Q0 for the Baranyi model from the duration of the lag phase

Usage

```
lambda_to_Q0(lambda, mu, logbase_mu = 10)
```

Arguments

`lambda`	Duration of the lag phase.
`mu`	Specific growth rate in the exponential phase.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.

`logistic_model` *Logistic growth model*

Description

Logistic growth model

Usage

```
logistic_model(times, logN0, mu, lambda, C)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration
`C`	Difference between `logN0` and the maximum log-count.

Value

Numeric vector with the predicted microbial count

`loglinear_model` *Loglinear model*

Description

Loglinear model

Usage

```
loglinear_model(times, logN0, mu)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)

`make_guess_factor` *Initial guesses for the secondary model of one factor*

Description

Initial guesses for the secondary model of one factor

Usage

```
make_guess_factor(fit_data, sec_model, factor)
```

Arguments

`fit_data`	Tibble with the data used for the fit. It must have one column with the observed growth rate (named `mu` by default; can be changed using the "formula" argument) and as many columns as needed with the environmental factors.
`sec_model`	character defining the secondary model equation according to `secondary_model_data()`
`factor`	character defining the environmental factor

`make_guess_primary` *Initial guesses for fitting primary growth models*

Description

[Experimental]

The function uses some heuristics to provide initial guesses for the parameters of the growth model selected that can be used with `fit_growth()`.

Usage

```
make_guess_primary(
  fit_data,
  primary_model,
  logbase_mu = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	the experimental data. A tibble (or data.frame) with a column named `time` with the elapsed time and one called `logN` with the logarithm of the population size
`primary_model`	a string defining the equation of the primary model, as defined in `primary_model_data()`
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.

Value

A named numeric vector of initial guesses for the model parameters

Examples

```
## An example of experimental data

my_data <- data.frame(time = 0:9,
                      logN = c(2, 2.1, 1.8, 2.5, 3.1, 3.4, 4, 4.5, 4.8, 4.7))

## We just need to pass the data and the model equation

make_guess_primary(my_data, "Logistic")

## We can use this together with fit_growth()

fit_growth(my_data,
           list(primary = "Logistic"),
           make_guess_primary(my_data, "Logistic"),
           c()
           )

## The parameters returned by the function are adapted to the model

make_guess_primary(my_data, "Baranyi")

## It can express mu in other logbases

make_guess_primary(my_data, "Baranyi", logbase_mu = exp(1))  # natural
make_guess_primary(my_data, "Baranyi", logbase_mu = 2)  # base2
```

`make_guess_secondary` *Initial guesses for the parameters of a secondary model*

Description

[Experimental]

Uses some heuristic rules to generate an initial guess of the model parameters of secondary growth models that can be used for model fitting with `fit_secondary_growth()`.

Usage

```
make_guess_secondary(fit_data, sec_model_names)
```

Arguments

`fit_data`	Tibble with the data used for the fit. It must have one column with the observed growth rate (named `mu` by default; can be changed using the "formula" argument) and as many columns as needed with the environmental factors.
`sec_model_names`	Named character vector defining the secondary model for each environmental factor.

Examples

```
## We can use the example dataset included in the package

data("example_cardinal")

## We assign model equations to factors as usual

sec_model_names <- c(temperature = "Zwietering", pH = "fullRatkowsky")

## We can then calculate the initial guesses

make_guess_secondary(example_cardinal, sec_model_names)

## We can pass these parameters directly to fit_secondary_growth

fit_secondary_growth(example_cardinal,
                     make_guess_secondary(example_cardinal, sec_model_names),
                     c(),
                     sec_model_names)
```

`MCMCgrowth` *MCMCgrowth class*

Description

[Stable]

The `MCMCgrowth` class contains the results of a growth prediction consider parameter variability based on a model fitted using an MCMC algorithm.

It is a subclass of list with items:

- sample: Parameter sample used for the calculations.
- simulations: Individual growth curves calculated based on the parameter sample.
- quantiles: Tibble with the limits of the credible intervals (5%, 10%, 50%, 90% and 95%) for each time point.
- model: Instance of `FitDynamicGrowthMCMC` used for predictions.
- env_conditions: A tibble with the environmental conditions of the simulation.

Usage

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

## S3 method for class 'MCMCgrowth'
plot(
  x,
  y = NULL,
  ...,
  add_factor = NULL,
  alpha_80 = 0.5,
  fill_80 = "grey",
  alpha_90 = 0.5,
  fill_90 = "grey",
  label_y1 = "logN",
  label_y2 = add_factor,
  line_col = "black",
  line_type = 1,
  line_size = 1,
  line_type2 = 2,
  line_col2 = "black",
  line_size2 = 1,
  ylims = NULL
)
```

Arguments

`x`	The object of class `MCMCgrowth` to plot.
`...`	ignored.
`y`	ignored
`add_factor`	Includes the variation of one environmental factor in the plot. It must be one of the column names in x$env_conditions.
`alpha_80`	transparency of the ribbon for the 80th posterior. .5 by default.
`fill_80`	fill colour of the ribbon for the 80th posterior. "grey" by default.
`alpha_90`	transparency of the ribbon for the 90th posterior. .5 by default.
`fill_90`	fill colour of the ribbon for the 90th posterior. "grey" by default.
`label_y1`	label of the primary y axis. "logN" by default.
`label_y2`	label of the secondary y axis. The name of the environmental factor by default.
`line_col`	colour of the line representing the median. "black" by default.
`line_type`	linetype for the line representing the median. solid by default.
`line_size`	size of the line representing the median. 1 by default.
`line_type2`	linetype for the line representing the environmental condition. Dashed by default.
`line_col2`	colour of the line representing the environmental condition. "black" by default.

`line_size2`	size of the line representing the environmental condition. 1 by default.
`ylims`	limits of the primary y-axis. `NULL` by default (let ggplot choose).

Methods (by generic)

- `print(MCMCgrowth)`: print of the model
- `plot(MCMCgrowth)`: plot of predicted growth (prediction band).

`multiple_conditions` *Environmental conditions during several dynamic experiments*

Description

This dataset is paired with multiple_counts to illustrate the global fitting of `fit_growth()`.

Usage

```
multiple_conditions
```

Format

A nested list with two elements, each one corresponding to one experiment. Each element is a data.frame with three columns:

- time: elapsed time
- temperature: observed temperature
- pH: observed pH

`multiple_counts` *Population growth observed in several dynamic experiments*

Description

This dataset is paired with multiple_conditions to illustrate the global fitting of `fit_growth()`.

Usage

```
multiple_counts
```

Format

A nested list with two elements, each one corresponding to one experiment. Each element is a data.frame with two columns:

- time: elapsed time
- logN: log10 of the microbial concentration

`multiple_experiments` *A set of growth experiments under dynamic conditions*

Description

An example dataset illustrating the requirements of `fit_multiple_growth()` and `fit_multiple_growth_MCMC()`.

Usage

```
multiple_experiments
```

Format

A nested list with two elements. Each element corresponds to one experiment and is described by a list with two data frames:

- **data** a tibble describing the microbial counts. It has 2 columns: time (elapsed time) and logN (logarithm of the microbial count).
- **conditions** a tibble describing the environmental conditions. It has 3 columns: time (elapsed time), temperature (storage temperature) and pH (pH of the media).

`predictMCMC` *Generic for calculating predictions with uncertainty from fits*

Description

Generic for calculating predictions with uncertainty from fits

Usage

```
predictMCMC(
  model,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`model`	Fit object
`times`	see specific methods for each class
`env_conditions`	see specific methods for each class
`niter`	see specific methods for each class
`newpars`	see specific methods for each class
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

`predict_dynamic_growth`

Growth under dynamic conditions

Description

[Superseded]

The function `predict_dynamic_growth()` has been superseded by the top-level function `predict_growth()`, which provides a unified approach for growth modelling.

Regardless on that, it can still predict population growth under dynamic conditions based on the Baranyi model (Baranyi and Roberts, 1994) and secondary models based on the gamma concept (Zwietering et al. 1992).

Model predictions are done by linear interpolation of the environmental conditions defined in `env_conditions`.

Usage

```
predict_dynamic_growth(
  times,
  env_conditions,
  primary_pars,
  secondary_models,
  ...,
  check = TRUE,
  logbase_logN = 10,
  logbase_mu = logbase_logN,
  formula = . ~ time
)
```

Arguments

`times`	Numeric vector of storage times to make the predictions
`env_conditions`	Tibble (or data.frame) describing the variation of the environmental conditions during storage. It must have with the elapsed time (named `time` by default; can be changed with the "formula" argument), and as many additional columns as environmental factors.

`primary_pars`	A named list defining the parameters of the primary model and the initial values of the model variables. That is, with names `mu_opt`, `Nmax`, `N0`, `Q0`.
`secondary_models`	A nested list describing the secondary models.
`...`	Additional arguments for `deSolve::ode()`.
`check`	Whether to check the validity of the models. `TRUE` by default.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`formula`	An object of class "formula" describing the x variable. `. ~ time` as a default.

Value

An instance of `DynamicGrowth()`.

Examples

```
## Definition of the environmental conditions

library(tibble)

my_conditions <- tibble(time = c(0, 5, 40),
    temperature = c(20, 30, 35),
    pH = c(7, 6.5, 5)
    )

## Definition of the model parameters

my_primary <- list(mu_opt = 2,
    Nmax = 1e8,N0 = 1e0,
    Q0 = 1e-3)

sec_temperature <- list(model = "Zwietering",
    xmin = 25, xopt = 35, n = 1)

sec_pH = list(model = "CPM",
    xmin = 5.5, xopt = 6.5,
    xmax = 7.5, n = 2)

my_secondary <- list(
    temperature = sec_temperature,
    pH = sec_pH
    )

my_times <- seq(0, 50, length = 1000)

## Do the simulation

dynamic_prediction <- predict_dynamic_growth(my_times,
```

```
    my_conditions, my_primary,
    my_secondary)

## Plot the results

plot(dynamic_prediction)

## We can plot some environmental factor with add_factor

plot(dynamic_prediction, add_factor = "temperature", ylims= c(0, 8),
    label_y1 = "Microbial count (log CFU/ml)",
    label_y2 = "Storage temperature (C)")
```

`predict_growth` *Prediction of microbial growth*

Description

[Stable]

This function provides a top-level interface for predicting population growth. Predictions can be made either under constant or dynamic environmental conditions. See below for details on the calculations.

Usage

```
predict_growth(
  times,
  primary_model,
  environment = "constant",
  secondary_models = NULL,
  env_conditions = NULL,
  ...,
  check = TRUE,
  logbase_mu = logbase_logN,
  logbase_logN = 10,
  formula = . ~ time
)
```

Arguments

`times`	numeric vector of time points for making the predictions
`primary_model`	named list defining the values of the parameters of the primary growth model
`environment`	type of environment. Either "constant" (default) or "dynamic" (see below for details on the calculations for each condition)
`secondary_models`	a nested list describing the secondary models. See below for details

`env_conditions`	Tibble describing the variation of the environmental conditions for dynamic experiments. It must have with the elapsed time (named `time` by default; can be changed with the "formula" argument), and as many additional columns as environmental factors. Ignored for "constant" environments.
`...`	Additional arguments for `deSolve::ode()`.
`check`	Whether to check the validity of the models. `TRUE` by default.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	An object of class "formula" describing the x variable for predictions under dynamic conditions. `. ~ time` as a default.

Details

To ease data input, the functions can convert between parameters defined in different scales. Namely, for predictions in constant environments (environment="constant"):

- "logN0" can be defined as "N0". The function automatically calculates the log-transformation.
- "logNmax" can be defined as "Nmax". The function automatically calculates the log-transformation.
- "mu" can be defined as "mu_opt". The function assumes the prediction is under optimal growth conditions.
- "lambda" can be defined by "Q0". The duration of the lag phase is calculated using `Q0_to_lambda()`.

And, for predictions in dynamic environments (environment="dynamic"):

- "N0" can be defined as "N0". The function automatically calculates the antilog-transformation.
- "Nmax" can be defined as "logNmax". The function automatically calculates the antilog-transformation.
- "mu" can be defined as "mu_opt". The function assumes mu was calculated under optimal growth conditions.
- "Q0" can be defined by the value of "lambda" under dynamic conditions. Then, the value of Q0 is calculated using `lambda_to_Q0()`.

Value

An instance of GrowthPrediction.

Predictions in constant environments

Predictions under constant environments are calculated using only primary models. Consequently, the arguments "secondary_models" and "env_conditions" are ignored. If these were passed, the function would return a warning. In this case, predictions are calculated using the algebraic form of the primary model (see vignette for details).

The growth model is defined through the "primary_model" argument using a named list. One of the list elements must be named "model" and must take take one of the valid keys returned by `primary_model_data()`. The remaining entries of the list define the values of the parameters of the selected model. A list of valid keys can be retrieved using `primary_model_data()` (see example below). Note that the functions can do some operations to facilitate the compatibility between constant and dynamic environments (see Details).

Predictions in dynamic environments

Predictions under dynamic environments are calculated by solving numerically the differential equation of the Baranyi growth model. The effect of changes in the environmental conditions in the growth rate are calculated according to the gamma approach. Therefore, one must define both primary and secondary models.

The dynamic environmental conditions are defined using a tibble (or data.frame) through the "env_conditions" argument. It must include one column named "time" stating the elapsed time and as many additional columns as environmental conditions included in the prediction. For values of time not included in the tibble, the values of the environmental conditions are calculated by linear interpolation.

Primary models are defined as a named list through the "primary_model" argument. It must include the following elements:

- N0: initial population size
- Nmax: maximum population size in the stationary growth phase
- mu_opt: growth rate under optimal growth conditions
- Q0: value defining the duration of the lag phase Additional details on these parameters can be found in the package vignettes.

Secondary models are defined as a nested list through the "secondary_models" argument. The list must have one entry per environmental condition, whose name must match those used in the "env_conditions" argument. Each of these entries must be a named list defining the secondary model for each environmental condition. The model equation is defined in an entry named "model" (valid keys can be retrieved from `secondary_model_data()`). Then, additional entries defined the values of each model parameters (valid keys can be retrieved from `secondary_model_data()`)

For additional details on how to define the secondary models, please see the package vignettes (and examples below).

Examples

```
## Example 1 - Growth under constant conditions ------------------------------

## Valid model keys can be retrieved calling primary_model_data()

primary_model_data()

my_model <- "modGompertz"  # we will use the modified-Gompertz

## The keys of the model parameters can also be obtained from primary_model_data()

primary_model_data(my_model)$pars

## We define the primary model as a list
```

```
my_model <- list(model = "modGompertz", logN0 = 0, C = 6, mu = .2, lambda = 20)

## We can now make the predictions

my_time <- seq(0, 100, length = 1000)  # Vector of time points for the calculations

my_prediction <- predict_growth(my_time, my_model, environment = "constant")

## The instance of IsothermalGrowth includes several S3 methods

print(my_prediction)
plot(my_prediction)
coef(my_prediction)

## Example 2 - Growth under dynamic conditions ------------------------------

## We will consider the effect of two factors: temperature and pH

my_conditions <- data.frame(time = c(0, 5, 40),
                            temperature = c(20, 30, 35),
                            pH = c(7, 6.5, 5)
                            )

## The primary model is defined as a named list

my_primary <- list(mu = 2, Nmax = 1e7, N0 = 1, Q0 = 1e-3)

## The secondary model is defined independently for each factor

sec_temperature <- list(model = "Zwietering",
    xmin = 25, xopt = 35, n = 1)

sec_pH = list(model = "CPM",
    xmin = 5.5, xopt = 6.5,
    xmax = 7.5, n = 2)

## Then, they are assigned to each factor using a named list

my_secondary <- list(
    temperature = sec_temperature,
    pH = sec_pH
    )

## We can call the function now

my_times <- seq(0, 50, length = 1000)  # Where the output is calculated

dynamic_prediction <- predict_growth(environment = "dynamic",
                                     my_times, my_primary, my_secondary,
                                     my_conditions
                                     )

## The instance of DynamicGrowth includes several useful S3 methods
```

```
print(dynamic_prediction)
plot(dynamic_prediction)
plot(dynamic_prediction, add_factor = "pH")
coef(dynamic_prediction)

## The time_to_size function can predict the time to reach a population size

time_to_size(my_prediction, 3)
```

`predict_growth_uncertainty`

Isothermal growth with parameter uncertainty

Description

[Stable]

Simulation of microbial growth considering uncertianty in the model parameters. Calculations are based on Monte Carlo simulations, considering the parameters follow a multivariate normal distribution.

Usage

```
predict_growth_uncertainty(
  model_name,
  times,
  n_sims,
  pars,
  corr_matrix = diag(nrow(pars)),
  check = TRUE,
  logbase_mu = logbase_logN,
  logbase_logN = 10
)
```

Arguments

`model_name`	Character describing the primary growth model.
`times`	Numeric vector of storage times for the simulations.
`n_sims`	Number of simulations.
`pars`	A tibble describing the parameter uncertainty (see details).
`corr_matrix`	Correlation matrix of the model parameters. Defined in the same order as in `pars`. An identity matrix by default (uncorrelated parameters).
`check`	Whether to do some tests. `FALSE` by default.

`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Details

The distributions of the model parameters are defined in the `pars` argument using a tibble with 4 columns:

- par: identifier of the model parameter (according to `primary_model_data()`),
- mean: mean value of the model parameter.,
- sd: standard deviation of the model parameter.,
- scale: scale at which the model parameter is defined. Valid values are 'original' (no transformation), 'sqrt' square root or 'log' log-scale. The parameter sample is generated considering the parameter follows a marginal normal distribution at this scale, and is later converted to the original scale for calculations.

Value

An instance of `GrowthUncertainty()`.

Examples

```
## Definition of the simulation settings

my_model <- "Baranyi"
my_times <- seq(0, 30, length = 100)
n_sims <- 3000

library(tibble)

pars <- tribble(
    ~par, ~mean, ~sd, ~scale,
    "logN0", 0, .2, "original",
    "mu", 2, .3, "sqrt",
    "lambda", 4, .4, "sqrt",
    "logNmax", 6, .5, "original"
)

## Calling the function

stoc_growth <- predict_growth_uncertainty(my_model, my_times, n_sims, pars)

## We can plot the results

plot(stoc_growth)

## Adding parameter correlation

my_cor <- matrix(c(1,   0,   0, 0,
```

```
    0,   1, 0.7, 0,
    0, 0.7,   1, 0,
    0,   0,   0, 1),
    nrow = 4)

stoc_growth2 <- predict_growth_uncertainty(my_model, my_times, n_sims, pars, my_cor)

plot(stoc_growth2)

## The time_to_size function can calculate the median growth curve to reach a size

time_to_size(stoc_growth, 4)

## Or the distribution of times

dist <- time_to_size(stoc_growth, 4, type = "distribution")
plot(dist)
```

`predict_isothermal_growth` *Isothermal microbial growth*

Description

[Superseded]

The function `predict_isothermal_growth()` has been superseded by the top-level function `predict_growth()`, which provides a unified approach for growth modelling.

Regardless of that, it can still be used to predict population growth under static environmental conditions (i.e. using primary models).

Usage

```
predict_isothermal_growth(
  model_name,
  times,
  model_pars,
  check = TRUE,
  logbase_mu = 10,
  logbase_logN = 10
)
```

Arguments

`model_name`	Character defining the growth model.
`times`	Numeric vector of storage times for the predictions.
`model_pars`	Named vector or list defining the values of the model parameters.
`check`	Whether to do basic checks (TRUE by default).
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, the same as logbase_logN. See vignette about units for details.
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Value

An instance of `IsothermalGrowth()`.

Examples

```
## Define the simulations parameters

my_model <- "modGompertz"
my_pars <- list(logN0 = 2, C = 6, mu = .2, lambda = 25)
my_time <- seq(0, 100, length = 1000)

## Do the simulation

static_prediction <- predict_isothermal_growth(my_model, my_time, my_pars)

## Plot the results

plot(static_prediction)
```

`predict_MCMC_growth` *Stochastic growth of MCMC fit*

Description

[Superseded]

The function `predict_MCMC_growth()` has been superseded by `predictMCMC()` S3 methods of the relevant classes.

Nonetheless, it can still make a prediction of microbial growth including parameter uncertainty based on a growth model fitted using `fit_MCMC_growth()` or `fit_multiple_growth_MCMC()`. This function predicts growth curves for `niter` samples (with replacement) of the samples of the MCMC algorithm. Then, credible intervals are calculated based on the quantiles of the model predictions at each time point.

Usage

```
predict_MCMC_growth(
  MCMCfit,
  times,
  env_conditions,
  niter,
  newpars = NULL,
  formula = . ~ time
)
```

Arguments

`MCMCfit`	An instance of `FitDynamicGrowthMCMC` or `FitMultipleGrowthMCMC`.
`times`	Numeric vector of storage times for the predictions.
`env_conditions`	Tibble with the (dynamic) environmental conditions during the experiment. It must have one column named 'time' with the storage time and as many columns as required with the environmental conditions.
`niter`	Number of iterations.
`newpars`	A named list defining new values for the some model parameters. The name must be the identifier of a model already included in the model. These parameters do not include variation, so defining a new value for a fitted parameters "fixes" it. `NULL` by default (no new parameters).
`formula`	A formula stating the column named defining the elapsed time in `env_conditions`. By default, . ~ time.

Value

An instance of `MCMCgrowth()`.

Examples

```
## We need a FitDynamicGrowthMCMC object

data("example_dynamic_growth")
data("example_env_conditions")

sec_model_names <- c(temperature = "CPM", aw= "CPM")

known_pars <- list(Nmax = 1e4,  # Primary model
    N0 = 1e0, Q0 = 1e-3,  # Initial values of the primary model
    mu_opt = 4, # mu_opt of the gamma model
    temperature_n = 1,  # Secondary model for temperature
    aw_xmax = 1, aw_xmin = .9, aw_n = 1  # Secondary model for water activity
    )

my_start <- list(temperature_xmin = 25, temperature_xopt = 35,
    temperature_xmax = 40,
    aw_xopt = .95)
```

```
set.seed(12124) # Setting seed for repeatability

my_MCMC_fit <- fit_MCMC_growth(example_dynamic_growth, example_env_conditions,
    my_start, known_pars, sec_model_names, niter = 3000)

## Define the conditions for the simulation

my_times <- seq(0, 15, length = 50)
niter <- 2000

newpars <- list(N0 = 1e-1,  # A parameter that was fixed
                temperature_xmax = 120  # A parameter that was fitted
                )


## Make the simulations

my_MCMC_prediction <- predict_MCMC_growth(my_MCMC_fit,
    my_times,
    example_env_conditions, # It could be different from the one used for fitting
    niter,
    newpars)

## We can plot the prediction interval

plot(my_MCMC_prediction)

## We can also get the quantiles at each time point

print(my_MCMC_prediction$quantiles)
```

`predict_stochastic_growth` *Deprecated isothermal growth with parameter uncertainty*

Description

[Deprecated]

`predict_stochastic_growth()` was renamed `predict_growth_uncertainty()` because the original function name may be misleading, as this is not a stochastic differential equation

Usage

```
predict_stochastic_growth(
  model_name,
  times,
  n_sims,
```

```
  pars,
  corr_matrix = diag(nrow(pars)),
  check = TRUE
)
```

Arguments

`model_name`	Character describing the primary growth model.
`times`	Numeric vector of storage times for the simulations.
`n_sims`	Number of simulations.
`pars`	A tibble describing the parameter uncertainty (see details).
`corr_matrix`	Correlation matrix of the model parameters. Defined in the same order as in `pars`. An identity matrix by default (uncorrelated parameters).
`check`	Whether to do some tests. `FALSE` by default.

`primary_model_data` *Metainformation of primary growth models*

Description

[Stable]

Provides different types of meta-data about the primary growth models included in biogrowth. This information is the basis of the automatic checks, and can also help in the definition of models for `predict_growth()` and `fit_growth()`.

Usage

```
primary_model_data(model_name = NULL)
```

Arguments

`model_name`	The name of the model or `NULL` (default).

Value

If model_name is `NULL`, returns a character string with the available models. If is a valid identifier, it returns a list with metainformation about the model. If model_name name is not a valid identifier, raises an error.

`Q0_to_lambda` *Lag phase duration from Q0*

Description

[Stable]

Convenience function to calculate the lag phase duration (lambda) of the Baranyi model from the maximum specific growth rate and the initial value of the variable Q.

Note that this function uses the unit system of biogrowth (i.e. log10). Care must be taken when using parameters obtained from other sources.

Usage

```
Q0_to_lambda(q0, mu, logbase_mu = 10)
```

Arguments

`q0`	Initial value of the variable Q.
`mu`	Specific growth rate in the exponential phase.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.

`refrigeratorSpain` *Temperature recorded in refrigerators*

Description

This dataset includes the temperature recorded in refrigerators in households of the Catalonia region. The data was published as part of Jofre et al. (2019) Domestic refrigerator temperatures in Spain: Assessment of its impact on the safety and shelf-life of cooked meat products. Food Research International, 126, 108578. And was kindly provided by the original authors of the study.

Usage

```
refrigeratorSpain
```

Format

A tibble with three columns:

- time: elapsed time in hours
- A1: temperature observed in refrigerator "1"
- A2: temperature observed in refrigerator "2"

`richards_model` *Richards growth model*

Description

Richards growth model

Usage

```
richards_model(times, logN0, mu, lambda, C, nu)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration
`C`	Difference between `logN0` and the maximum log-count.
`nu`	Parameter describing the transition between growth phases

---

`Rossoaw_model` *Secondary Rosso model for water activity*

Description

Secondary model for water activity as defined by Aryani et al. (2001).

Usage

```
Rossoaw_model(x, xmin)
```

Arguments

`x`	Value of the environmental factor (in principle, aw).
`xmin`	Minimum value for growth (in principle, aw).

Value

The corresponding gamma factor.

`SecondaryComparison` *SecondaryComparison class*

Description

The `SecondaryComparison` class contains several functions for model comparison and model selection of growth models. It should not be instanced directly. Instead, it should be constructed using `compare_secondary_fits()`.

It includes two type of tools for model selection and comparison: statistical indexes and visual analyses. Please check the sections below for details.

Note that all these tools use the names defined in `compare_secondary_fits()`, so we recommend passing a named list to that function.

Usage

```
## S3 method for class 'SecondaryComparison'
coef(object, ...)

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

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

## S3 method for class 'SecondaryComparison'
plot(x, y, ..., type = 1, add_trend = TRUE)
```

Arguments

`object`	an instance of SecondaryComparison
`...`	ignored
`x`	an instance of SecondaryComparison
`y`	ignored
`type`	if type==1, the plot compares the model predictions. If type ==2, the plot compares the parameter estimates.
`add_trend`	should a trend line of the residuals be added for type==3? `TRUE` by default

Methods (by generic)

- `coef(SecondaryComparison)`: table of parameter estimates
- `summary(SecondaryComparison)`: summary table for the comparison
- `print(SecondaryComparison)`: print of the model comparison
- `plot(SecondaryComparison)`: illustrations comparing the fitted models

Statistical indexes

`SecondaryComparison` implements two S3 methods to obtain numerical values to facilitate model comparison and selection.

- the `coef` method returns a tibble with the values of the parameter estimates and their corresponding standard errors for each model.
- the `summary` returns a tibble with the AIC, number of degrees of freedom, mean error and root mean squared error for each model.

Visual analyses

The S3 plot method can generate three types of plots:

- when `type = 1`, the plot compares the observations against the model predictions for each model. The plot includes a linear model fitted to the residuals. In the case of a perfect fit, the line would have slope=1 and intercept=0 (shown as a black, dashed line).
- when `type = 2`, the plot compares the parameter estimates using error bars, where the limits of the error bars are the expected value +/- one standard error. In case one model does not has some model parameter (i.e. either because it is not defined or because it was fixed), the parameter is not included in the plot.

`secondary_model_data` *Metainformation of secondary growth models*

Description

[Stable]

Provides different types of meta-data about the secondary growth models included in biogrowth. This information is the basis of the automatic checks, and can also help in the definition of models for `predict_growth()` and `fit_growth()`.

Usage

```
secondary_model_data(model_name = NULL)
```

Arguments

`model_name`	The name of the model or `NULL` (default).

Value

If model_name is `NULL`, returns a character string with the available models. If is a valid identifier, it returns a list with metainformation about the model. If model_name name is not a valid identifier, raises an error.

`show_guess_dynamic` *Plot of the initial guess for growth under dynamic environmental conditions*

Description

Compares the prediction corresponding to a guess of the parameters of the model against experimental data

Usage

```
show_guess_dynamic(
  fit_data,
  model_keys,
  guess,
  env_conditions,
  logbase_mu = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	Tibble (or data.frame) of data for the fit. It must have two columns, one with the elapsed time (`time` by default) and another one with the decimal logarithm of the populatoin size (`logN` by default). Different column names can be defined using the `formula` argument.
`model_keys`	Named the equations of the secondary model as in `fit_growth()`
`guess`	Named vector with the initial guess of the model parameters as in `fit_growth()`
`env_conditions`	Tibble describing the variation of the environmental conditions for dynamic experiments. See `fit_growth()`.
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.

Value

A `ggplot2::ggplot()` comparing the model prediction against the data

`show_guess_primary` *Plot of the initial guess for growth under constant environmental conditions*

Description

Compares the prediction corresponding to a guess of the parameters of the primary model against experimental data

Usage

```
show_guess_primary(
  fit_data,
  model_name,
  guess,
  logbase_mu = 10,
  formula = logN ~ time
)
```

Arguments

`fit_data`	Tibble (or data.frame) of data for the fit. It must have two columns, one with the elapsed time (`time` by default) and another one with the decimal logarithm of the populatoin size (`logN` by default). Different column names can be defined using the `formula` argument.
`model_name`	Character defining the primary growth model as per `primary_model_data()`
`guess`	Named vector with the initial guess of the model parameters
`logbase_mu`	Base of the logarithm the growth rate is referred to. By default, 10 (i.e. log10). See vignette about units for details.
`formula`	an object of class "formula" describing the x and y variables. `logN ~ time` as a default.

Value

A `ggplot2::ggplot()` comparing the model prediction against the data

`StochasticGrowth` *StochasticGrowth class*

Description

[Deprecated]

The class StochasticGrowth has been deprecated by class GrowthUncertainty, which provides less misleading name.

Still, it is still returned if the deprecated `predict_stochastic_growth()` is called.

The `StochasticGrowth` class contains the results of a growth prediction under isothermal conditions considering parameter unceratinty. Its constructor is `predict_stochastic_growth()`.

It is a subclass of list with the items:

- sample: parameter sample used for the calculations.
- simulations: growth curves predicted for each parameter.
- quantiles: limits of the credible intervals (5%, 10%, 50%, 90%, 95%) for each time point.
- model: Model used for the calculations.
- mus: Mean parameter values used for the simulations.
- sigma: Variance-covariance matrix used for the simulations.

Usage

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

## S3 method for class 'StochasticGrowth'
plot(
  x,
  y = NULL,
  ...,
  line_col = "black",
  line_size = 0.5,
  line_type = "solid",
  ribbon80_fill = "grey",
  ribbon90_fill = "grey",
  alpha80 = 0.5,
  alpha90 = 0.4
)
```

Arguments

`x`	The object of class `StochasticGrowth` to plot.
`...`	ignored.
`y`	ignored
`line_col`	Aesthetic parameter to change the colour of the line geom in the plot, see: `ggplot2::geom_line()`
`line_size`	Aesthetic parameter to change the thickness of the line geom in the plot, see: `ggplot2::geom_line()`

`line_type`	Aesthetic parameter to change the type of the line geom in the plot, takes numbers (1-6) or strings ("solid") see: `ggplot2::geom_line()`
`ribbon80_fill`	fill colour for the space between the 10th and 90th quantile, see: `ggplot2::geom_ribbon()`
`ribbon90_fill`	fill colour for the space between the 5th and 95th quantile, see: `ggplot2::geom_ribbon()`
`alpha80`	transparency of the ribbon aesthetic for the space between the 10th and 90th quantile. Takes a value between 0 (fully transparant) and 1 (fully opaque)
`alpha90`	transparency of the ribbon aesthetic for the space between the 5th and 95th quantile. Takes a value between 0 (fully transparant) and 1 (fully opaque).

Details

FitIsoGrowth class

Methods (by generic)

- `print(StochasticGrowth)`: print of the model
- `plot(StochasticGrowth)`: Growth prediction (prediction band) considering parameter uncertainty.

`TimeDistribution` *TimeDistribution class*

Description

The `TimeDistribution` class contains an estimate of the probability distribution of the time to reach a given microbial count. Its constructor is `distribution_to_logcount()`.

It is a subclass of list with the items:

- distribution Sample of the distribution of times to reach `log_count`.
- summary Summary statistics of distribution (mean, sd, median, q10 and q90).

Usage

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

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

## S3 method for class 'TimeDistribution'
plot(x, y = NULL, ..., bin_width = NULL)
```

Arguments

`x`	The object of class `TimeDistribution` to plot.
`...`	ignored.
`object`	An instance of `TimeDistribution`.
`y`	ignored.
`bin_width`	A number that specifies the width of a bin in the histogram, see: `ggplot2::geom_histogram()`. `NULL` by default.

Methods (by generic)

- `print(TimeDistribution)`: print of the model
- `summary(TimeDistribution)`: summary of the model
- `plot(TimeDistribution)`: plot of the distribution of the time to reach a microbial count.

`time_to_logcount` *Time to reach a given microbial count*

Description

[Superseded]

The function `time_to_logcount()` has been superseded by function `time_to_size()`, which provides a more general interface.

But it still returns the storage time required for the microbial count to reach `log_count` according to the predictions of `model`. Calculations are done using linear interpolation of the model predictions.

Usage

```
time_to_logcount(model, log_count)
```

Arguments

`model`	An instance of `IsothermalGrowth` or `DynamicGrowth`.
`log_count`	The target log microbial count.

Value

The predicted time to reach `log_count`.

Examples

```
## First of all, we will get an IsothermalGrowth object

my_model <- "modGompertz"
my_pars <- list(logN0 = 2, C = 6, mu = .2, lambda = 25)
my_time <- seq(0, 100, length = 1000)

static_prediction <- predict_isothermal_growth(my_model, my_time, my_pars)
plot(static_prediction)

## And now we calculate the time to reach a microbial count

time_to_logcount(static_prediction, 2.5)

## If log_count is outside the range of the predicted values, NA is returned

time_to_logcount(static_prediction, 20)
```

`time_to_size` *Time for the population to reach a given size*

Description

[Experimental]

Calculates the elapsed time required for the population to reach a given size (in log scale)

Usage

```
time_to_size(model, size, type = "discrete", logbase_logN = NULL)
```

Arguments

`model`	An instance of GrowthPrediction, GrowthFit, GlobalGrowthFit, GrowthUncertainty or MCMCgrowth.
`size`	Target population size (in log scale)
`type`	Tye of calculation, either "discrete" (default) or "distribution"
`logbase_logN`	Base of the logarithm for the population size. By default, 10 (i.e. log10). See vignette about units for details.

Details

The calculation method differs depending on the value of `type`. If `type="discrete"` (default), the function calculates by linear interpolation a discrete time to reach the target population size. If `type="distribution"`, this calculation is repeated several times, generating a distribution of the time. Note that this is only possible for instances of GrowthUncertainty or MCMCgrowth.

Value

If `type="discrete"`, a number. If `type="distribution"`, an instance of TimeDistribution.

Examples

```
## Example 1 - Growth predictions -------------------------------------------

## The model is defined as usual with predict_growth

my_model <- list(model = "modGompertz", logN0 = 0, C = 6, mu = .2, lambda = 20)

my_time <- seq(0, 100, length = 1000)  # Vector of time points for the calculations

my_prediction <- predict_growth(my_time, my_model, environment = "constant")

plot(my_prediction)

## We just have to pass the model and the size (in log10)

time_to_size(my_prediction, 3)

## If the size is not reached, it returns NA

time_to_size(my_prediction, 8)

## By default, it considers the population size is defined in the same log-base
## as the prediction. But that can be changed using logbase_logN

time_to_size(my_prediction, 3)
time_to_size(my_prediction, 3, logbase_logN = 10)
time_to_size(my_prediction, log(100), logbase_logN = exp(1))

## Example 2 - Model fit ----------------------------------------------------

my_data <- data.frame(time = c(0, 25, 50, 75, 100),
                      logN = c(2, 2.5, 7, 8, 8))

models <- list(primary = "Baranyi")

known <- c(mu = .2)

start <- c(logNmax = 8, lambda = 25, logN0 = 2)

primary_fit <- fit_growth(my_data, models, start, known,
                          environment = "constant",
                          )

plot(primary_fit)

time_to_size(primary_fit, 4)

## Example 3 - Global fitting -----------------------------------------------
```

```
## We need a model first

data("multiple_counts")
data("multiple_conditions")

sec_models <- list(temperature = "CPM", pH = "CPM")

known_pars <- list(Nmax = 1e8, N0 = 1e0, Q0 = 1e-3,
                   temperature_n = 2, temperature_xmin = 20,
                   temperature_xmax = 35,
                   temperature_xopt = 30,
                   pH_n = 2, pH_xmin = 5.5, pH_xmax = 7.5, pH_xopt = 6.5)

my_start <- list(mu_opt = .8)

global_fit <- fit_growth(multiple_counts,
                         sec_models,
                         my_start,
                         known_pars,
                         environment = "dynamic",
                         algorithm = "regression",
                         approach = "global",
                         env_conditions = multiple_conditions
                         )

plot(global_fit)

## The function calculates the time for each experiment

time_to_size(global_fit, 3)

## It returns NA for the particular experiment if the size is not reached

time_to_size(global_fit, 4.5)
```

`trilinear_model` *Trilinear growth model*

Description

Trilinear growth model defined by Buchanan et al. (1997).

Usage

```
trilinear_model(times, logN0, mu, lambda, logNmax)
```

Arguments

`times`	Numeric vector of storage times
`logN0`	Initial log microbial count
`mu`	Maximum specific growth rate (in ln CFU/t)
`lambda`	Lag phase duration
`logNmax`	Maximum log microbial count

Value

Numeric vector with the predicted microbial count.

`zwietering_gamma` *Zwietering gamma model*

Description

Gamma model as defined by Zwietering et al. (1992). To avoid unreasonable predictions, it has been modified setting gamma=0 for values of x outside (xmin, xopt)

Usage

```
zwietering_gamma(x, xmin, xopt, n)
```

Arguments

`x`	Value of the environmental factor.
`xmin`	Minimum value of the environmental factor for growth.
`xopt`	Maximum value for growth
`n`	Exponent of the secondary model

Value

The corresponding gamma factor.

Index