Package 'MicrobialGrowth'

July 21, 2025

Type Package

Title Estimates Growth Parameters from Models and Plots the Curve

Version 1.0.0

Maintainer Florentin L'Homme <florentin.lhomme@gmail.com>

Description Fit growth curves to various known microbial growth models automatically to estimate growth parameters. Growth curves can be plotted with their uncertainty band. Growth models are: modified Gompertz model (Zwietering et al. (1990) <doi:10.1128/aem.56.6.1875-1881.1990>), Baranyi model (Baranyi and Roberts (1994) <doi:10.1016/0168-1605%2894%2990157-0>), Rosso model (Rosso et al. (1993) <doi:10.1006/jtbi.1993.1099>) and linear model (Dantigny (2005) <doi:10.1016/j.ijfoodmicro.2004.10.013>).

License GPL (>= 3)

Encoding UTF-8

Imports graphics, grDevices, methods, nlstools, stats, utils

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

LazyData true

Collate 'utils.R' 'base.R' 'acide.R' 'baranyi.R' 'data.R' 'gompertz.R' 'linear.R' 'rosso.R'

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Florentin L'Homme [aut, cre] (ORCID: <https://orcid.org/0000-0001-9684-3085>), Clarisse Breard [aut] (ORCID: <https://orcid.org/0009-0007-5310-7599>)

Repository CRAN

Date/Publication 2025-03-10 16:40:08 UTC

Contents

`.baranyi.formula` *Baranyi equation.*

Description

Baranyi equation.

Usage

```
.baranyi.formula(N0, Nmax, mu, lambda, base = exp(1))
```

Arguments

`N0`	initial population.
`Nmax`	final/maximum population.
`mu`	growth rate.
`lambda`	latency time.
`base`	the logarithm base used for plot y-scaling. By default, the natural logarithm is used. Set `NULL` to not scale.

Details

The output result is by default in the form `ln(N_t/N0)` (with `N_t` the population at time `t`). The base used can be modified by specifying the desired base in the `base` argument. For example, specifying `base=10` corresponds to output in the form `log_{10}(N_t/N0)`. It is possible to specify `base = NULL` to retrieve the normal `N_t` output.

Value

a function taking as input x (the time) and outputting the value of the Baranyi equation.

Examples

```
f <- .baranyi.formula(0.1, 2, 0.2, 5)
f(4)
## [1] 0.3498583
f(20)
## [1] 2.344923
```

`.checkMicrobialGrowthArgs` *Check MicrobialGrowth arguments (regression function)*

Description

Check the arguments passed to the regression function. Tests are generic for all models. For example, the same length for x and y, the type of the different arguments, the values inserted in `start`, `lower` and `upper`, etc. are tested

Usage

```
.checkMicrobialGrowthArgs(
  x,
  y,
  clip,
  start,
  lower,
  upper,
  nls.args,
  callbackError
)
```

Arguments

`x`	index series or time series.
`y`	values or list of values to regress (should not be logged).
`clip`	a pair of values indicating in which interval to clip the data `y`. When `clip` is missing, default values are used.
`start`	a named list of starting estimates. When `start` is missing, default values are used.
`lower`	a named list of lower bounds. When `lower` is missing, default values are used.
`upper`	a named list of upper bounds. When `upper` is missing, default values are used.
`nls.args`	additional parameters to use when calling nls.
`callbackError`	function to call on error during regression.

Details

During the check, the `clip` value is also updated. If the lower bound of `clip` is `-Inf` (default value), then this value is replaced by the smallest value greater than zero found in `y`.

Value

the modified `clip` value and raises an error if something is wrong.

.checkMicrobialGrowthCreateArgs *Check MicrobialGrowth arguments (create function)*

Description

Check the arguments passed to the create function. Tests are generic for all models. For example, the type and value `xlim`, the values of N0, Nmax, etc. are tested.

Usage

```
.checkMicrobialGrowthCreateArgs(N0, Nmax, mu, lambda, xlim, n)
```

Arguments

`N0`	initial population.
`Nmax`	final/maximum population.
`mu`	growth rate.
`lambda`	latency time.
`xlim`	range of values to simulate `x` and `y` (and hence plotting, *etc.*)
`n`	number of points to simulate in the interval `xlim`.

Value

raise an error if something is wrong.

.checkModelIntegrity *Model Integrity Checker*

Description

Function to check the integrity of a given model. Used only for development.

Usage

```
.checkModelIntegrity(model, verbose = TRUE)
```

Arguments

`model`	the model to check.
`verbose`	boolean indicating if the function is verbose, i.e. it indicates the different steps that it validates. If `FALSE`, only warnings and errors will be reported.

Value

No return value, called to check the integrity of a (new) model for the package (raises an error if the model is invalid).

Examples

```
# Auto-run on package build
models = listAvailableModels()
for (model in models) {
  .checkModelIntegrity(model)
}
```

.getDefaultNlsValues *Default NLS values*

Description

Gives default values for NLS regression from `x` and `y` values. The method of calculating default values differs depending on the amount of data available in `y`. Default values can be pre-set by providing them in the `start`, `lower` and `upper` arguments.

Usage

```
.getDefaultNlsValues(x, y, start = list(), lower = list(), upper = list())
```

Arguments

`x`	index series or time series.
`y`	values or list of values to regress (should not be logged, must be strictly greater than zero).
`start`	a named list of starting estimates. The coefficients specified in this list will not be calculated.
`lower`	a named list of lower bounds. The coefficients specified in this list will not be calculated.
`upper`	a named list of upper bounds. The coefficients specified in this list will not be calculated.

Details

default values are calculated as follows:

- `start`
  - `N0`: the minimum value of `y`
  - `Nmax`: the maximum value of `y`
  - `mu`:

    - if length(y) <= THRESHOLD_FEW_DATA: the greatest slope between two contiguous points (on logged y values)
    - else: the linear regression on data positioned in the middle ±25% of the amplitude on logged y
  - lambda: the highest value of x which is within the lowest 5% of amplitude of y
- lower
  - N0: the smallest value greater than zero calculated with 1/.Machine$double.xmax
  - Nmax: the mean value of y
  - mu: the amplitude on y divided by the amplitude on x
  - lambda:the minimum value of x
- upper
  - N0: the mean value of y
  - Nmax: twice the max value of y
  - mu:
    - if length(y) <= THRESHOLD_FEW_DATA: the amplitude on logged y divided by the smallest step between two contiguous x values
    - else: the greatest slope between two contiguous points (on logged y values)
  - lambda: the maximum value of x

Note that it is possible, particularly when there is little data, that linear regression for start$mu is not possible, hence the presence of condition with THRESHOLD_FEW_DATA.

Value

the default values of start, lower and upper for NLS regression.

Examples

```
# Example data
x = c(0.00, 5.26, 10.53, 15.79, 21.05, 26.32, 31.58, 36.84, 42.11, 47.37, 52.63,
      57.89, 63.16, 68.42, 73.68, 78.95, 84.21, 89.47, 94.74, 100.00)
y = c(0.15, 0.15, 0.15, 0.16, 0.19, 0.26, 0.38, 0.58, 0.85, 1.18, 1.53, 1.86,
      2.15, 2.38, 2.55, 2.66, 2.78, 2.85, 2.89, 2.93)

# Simple example
values = .getDefaultNlsValues(x, y)
cat("N0=", values$start$N0, " with limits [", values$lower$N0, ", ", values$upper$N0,"]", sep="")
## N0=0.15 with limits [5.562685e-309, 1.4315]

# Example with specifying a starting value (which will therefore not be calculated)
values = .getDefaultNlsValues(x, y, start=list(N0=0.1))
cat("N0=", values$start$N0, " with limits [", values$lower$N0, ", ", values$upper$N0,"]", sep="")
## N0=0.1 with limits [5.562685e-309, 1.4315]
```

`.gompertz.formula` *Gompertz equation.*

Description

Gompertz equation.

Usage

```
.gompertz.formula(N0, Nmax, mu, lambda, base = exp(1))
```

Arguments

`N0`	initial population.
`Nmax`	final/maximum population.
`mu`	growth rate.
`lambda`	latency time.
`base`	the logarithm base used for plot y-scaling. By default, the natural logarithm is used. Set `NULL` to not scale.

Details

The output result is by default in the form `ln(N_t/N0)` (with `N_t` the population at time `t`). The base used can be modified by specifying the desired base in the `base` argument. For example, specifying `base=10` corresponds to output in the form `log_{10}(N_t/N0)`. It is possible to specify `base = NULL` to retrieve the normal `N_t` output.

Value

a function taking as input x (the time) and outputting the value of the Gompertz equation.

Examples

```
f <- .gompertz.formula(0.1, 2, 0.2, 5)
f(4)
## [1] 0.1150952
f(20)
## [1] 2.505549
```

.linear.formula *Linear equation.*

Description

Linear equation.

Usage

```
.linear.formula(N0, Nmax, mu, lambda, base = NULL)
```

Arguments

N0	initial radius.
Nmax	final/maximum radius.
mu	growth rate.
lambda	latency time.
base	decimal base used for plot y-scaling.

Value

a function taking as input x (the time) and outputting the value of the linear equation.

Examples

```
f <- .linear.formula(0.1, 2, 0.2, 5)
f(4)
## [1] 0
f(20)
## [1] 3
```

.MicrobialGrowth.baranyi *Baranyi regression function*

Description

Regression function for Baranyi's model

Usage

```
.MicrobialGrowth.baranyi(
  x,
  y,
  clip = c(-Inf, Inf),
  start = list(),
  lower = list(),
  upper = list(),
  nls.args = list(),
  callbackError = NULL
)
```

Arguments

x	index series or time series.
y	values or list of values to regress (should not be logged).
clip	a pair of values indicating in which interval to clip the data y. When clip is missing, default values are used.
start	a named list of starting estimates. When start is missing, default values are used.
lower	a named list of lower bounds. When lower is missing, default values are used.
upper	a named list of upper bounds. When upper is missing, default values are used.
nls.args	additional parameters to use when calling nls.
callbackError	function to call on error during regression.

Details

The default values for clip, start, lower and upper are calculated based on the given data. These default values can be known through the call member of the returned value.

The nls.args argument is a list that can contain any nls function argument except formula, algorithm, start, lower and upper which are already fixed (via a homonymous or hard-coded argument).

For the callbackError argument, prefer the stop function to block or warning to not be blocking.

Value

a MicrobialGrowth-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.

message	contains the error message if the regression fails, NULL otherwise.
reg	the nls object returned by the nls function.

See Also

MicrobialGrowth, .baranyi.formula

`.MicrobialGrowth.gompertz`

Gompertz regression function

Description

Regression function for Gompertz's model

Usage

```
.MicrobialGrowth.gompertz(
  x,
  y,
  clip = c(-Inf, Inf),
  start = list(),
  lower = list(),
  upper = list(),
  nls.args = list(),
  callbackError = NULL
)
```

Arguments

x	index series or time series.
y	values or list of values to regress (should not be logged).
clip	a pair of values indicating in which interval to clip the data y. When clip is missing, default values are used.
start	a named list of starting estimates. When start is missing, default values are used.
lower	a named list of lower bounds. When lower is missing, default values are used.
upper	a named list of upper bounds. When upper is missing, default values are used.
nls.args	additional parameters to use when calling nls.
callbackError	function to call on error during regression.

Details

The default values for `clip`, `start`, `lower` and `upper` are calculated based on the given data. These default values can be known through the `call` member of the returned value.

The `nls.args` argument is a list that can contain any nls function argument except `formula`, `algorithm`, `start`, `lower` and `upper` which are already fixed (via a homonymous or hard-coded argument).

For the `callbackError` argument, prefer the `stop` function to block or `warning` to not be blocking.

Value

a MicrobialGrowth-object composed of

`call`	the matched call with several components.
`coefficients`	coefficients obtained by regression.
`data`	data used for regression, once the y values are clipped
`f`	a list of functions such as `formula` to retrieve the function of the model with the coefficients obtained by regression, `confint` to retrieve the confidence intervals, etc.
`isValid`	a boolean indicating whether the regression was successful or not.
`message`	contains the error message if the regression fails, `NULL` otherwise.
`reg`	the `nls` object returned by the `nls` function.

See Also

MicrobialGrowth, .gompertz.formula

.MicrobialGrowth.linear

Linear regression function

Description

Regression function for Linear's model

Usage

```
.MicrobialGrowth.linear(
  x,
  y,
  clip = c(-Inf, Inf),
  start = list(),
  lower = list(),
  upper = list(),
  nls.args = list(),
  callbackError = NULL
)
```

Arguments

x	index series or time series.
y	values or list of values to regress (should not be logged).
clip	a pair of values indicating in which interval to clip the data y. When clip is missing, default values are used.
start	a named list of starting estimates. When start is missing, default values are used.
lower	a named list of lower bounds. When lower is missing, default values are used.
upper	a named list of upper bounds. When upper is missing, default values are used.
nls.args	additional parameters to use when calling nls.
callbackError	function to call on error during regression.

Details

The default values for clip, start, lower and upper are calculated based on the given data. These default values can be known through the call member of the returned value.

The nls.args argument is a list that can contain any nls function argument except formula, algorithm, start, lower and upper which are already fixed (via a homonymous or hard-coded argument).

For the callbackError argument, prefer the stop function to block or warning to not be blocking.

Value

a MicrobialGrowth-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.
message	contains the error message if the regression fails, NULL otherwise.
reg	the nls object returned by the nls function.

See Also

MicrobialGrowth, .linear.formula

`.MicrobialGrowth.rosso` *Rosso regression function*

Description

Regression function for Rosso's model

Usage

```
.MicrobialGrowth.rosso(
  x,
  y,
  clip = c(-Inf, Inf),
  start = list(),
  lower = list(),
  upper = list(),
  nls.args = list(),
  callbackError = NULL
)
```

Arguments

`x`	index series or time series.
`y`	values or list of values to regress (should not be logged).
`clip`	a pair of values indicating in which interval to clip the data `y`. When `clip` is missing, default values are used.
`start`	a named list of starting estimates. When `start` is missing, default values are used.
`lower`	a named list of lower bounds. When `lower` is missing, default values are used.
`upper`	a named list of upper bounds. When `upper` is missing, default values are used.
`nls.args`	additional parameters to use when calling nls.
`callbackError`	function to call on error during regression.

Details

The default values for `clip`, `start`, `lower` and `upper` are calculated based on the given data. These default values can be known through the `call` member of the returned value.

The `nls.args` argument is a list that can contain any nls function argument except `formula`, `algorithm`, `start`, `lower` and `upper` which are already fixed (via a homonymous or hard-coded argument).

For the `callbackError` argument, prefer the `stop` function to block or `warning` to not be blocking.

Value

a MicrobialGrowth-object composed of

`call`	the matched call with several components.
`coefficients`	coefficients obtained by regression.
`data`	data used for regression, once the y values are clipped
`f`	a list of functions such as `formula` to retrieve the function of the model with the coefficients obtained by regression, `confint` to retrieve the confidence intervals, etc.
`isValid`	a boolean indicating whether the regression was successful or not.
`message`	contains the error message if the regression fails, `NULL` otherwise.
`reg`	the `nls` object returned by the `nls` function.

See Also

MicrobialGrowth, .rosso.formula

`.new.baranyi.core` *Baranyi object*

Description

A MicrobialGrowth object specialized for the baranyi model. Most of the methods are pre-implemented (some of these can be overwritten for a specific regression/create function). Must be completed for `data`, `isValid` (regression successful), etc.

Usage

```
.new.baranyi.core(...)
```

Arguments

`...`	further arguments passed to or from other methods.

Details

the three dots `...` are passed to the .new.MicrobialGrowth.core function.

Value

a Baranyi object skeleton.

Examples

```
# First, create the skeleton.
model.object = .new.baranyi.core()

# Then complete with data, functions, etc.
model.object$data$x = c(1,2,3)
model.object$data$y = c(1,2,3)
model.object$coefficients = list(N0 = 0, Nmax=0, mu=0, lambda=0)

# You can print, plot, etc., with the generic functions of MicrobialGrowth.
print(model.object)
##MicrobialGrowth, model specialized.model:
##    N0   Nmax     mu lambda
##     0      0      0      0
plot(model.object)

# Don't forget to change `isValid` to TRUE to confirm the success of the regression.
model.object$isValid = TRUE # Not a good idea here, since we have no `reg` value.
```

.new.gompertz.core *Gompertz object*

Description

A MicrobialGrowth object specialized for the gompertz model. Most of the methods are pre-implemented (some of these can be overwritten for a specific regression/create function). Must be completed for `data`, `isValid` (regression successful), etc.

Usage

```
.new.gompertz.core(...)
```

Arguments

...	further arguments passed to or from other methods.

Details

the three dots ... are passed to the .new.MicrobialGrowth.core function.

Value

a Gompertz object skeleton.

Examples

```
# First, create the skeleton.
model.object = .new.gompertz.core()

# Then complete with data, functions, etc.
model.object$data$x = c(1,2,3)
model.object$data$y = c(1,2,3)
model.object$coefficients = list(N0 = 0, Nmax=0, mu=0, lambda=0)

# You can print, plot, etc., with the generic functions of MicrobialGrowth.
print(model.object)
##MicrobialGrowth, model specialized.model:
##    N0   Nmax     mu lambda
##     0      0      0      0
plot(model.object)

# Don't forget to change `isValid` to TRUE to confirm the success of the regression.
model.object$isValid = TRUE # Not a good idea here, since we have no `reg` value.
```

.new.linear.core *Linear object*

Description

A MicrobialGrowth object specialized for the linear model. Most of the methods are pre-implemented (some of these can be overwritten for a specific regression/create function). Must be completed for `data`, `isValid` (regression successful), etc.

Usage

```
.new.linear.core(...)
```

Arguments

`...`	further arguments passed to or from other methods.

Details

the three dots `...` are passed to the .new.MicrobialGrowth.core function.

Value

a linear object skeleton.

Examples

```
# First, create the skeleton.
model.object = .new.linear.core()

# Then complete with data, functions, etc.
model.object$data$x = c(1,2,3)
model.object$data$y = c(1,2,3)
model.object$coefficients = list(N0 = 0, Nmax=0, mu=0, lambda=0)

# You can print, plot, etc., with the generic functions of MicrobialGrowth.
print(model.object)
##MicrobialGrowth, model specialized.model:
##    N0   Nmax     mu lambda
##     0      0      0      0
plot(model.object)

# Don't forget to change `isValid` to TRUE to confirm the success of the regression.
model.object$isValid = TRUE # Not a good idea here, since we have no `reg` value.
```

.new.MicrobialGrowth.core

MicrobialGrowth object

Description

Provide the skeleton of the MicrobialGrowth object. Must be completed for each model.

Usage

```
.new.MicrobialGrowth.core(...)
```

Arguments

...	further arguments passed to or from other methods.

Details

the three dots ... are passed to new.env function.

Value

a MicrobialGrowth object skeleton.

Examples

```
# First, create the skeleton.
model.object = .new.MicrobialGrowth.core()

# Then complete with data, functions, etc.
model.object$data$x = c(1,2,3)
model.object$data$y = c(1,2,3)
model.object$coefficients = list(N0 = 0, Nmax=0, mu=0, lambda=0)
model.object$f$formula = function(x){ return(x) }
model.object$f$confint.lower = function(x){ return(x - 1) }
model.object$f$confint.upper = function(x){ return(x + 1) }

# Specialize the object by adding a class name at first position.
class(model.object) = c("specialized.model", class(model.object))

# You can print, plot, etc., with the generic functions of MicrobialGrowth.
print(model.object)
##MicrobialGrowth, model specialized.model:
##    N0   Nmax     mu lambda
##     0      0      0      0
plot(model.object)

# Don't forget to change `isValid` to TRUE to confirm the success of the regression.
model.object$isValid = TRUE # Not a good idea here, since we have no `reg` value.
```

`.new.rosso.core` *Rosso object*

Description

A MicrobialGrowth object specialized for the rosso model. Most of the methods are pre-implemented (some of these can be overwritten for a specific regression/create function). Must be completed for `data`, `isValid` (regression successful), etc.

Usage

```
.new.rosso.core(...)
```

Arguments

`...`	further arguments passed to or from other methods.

Details

the three dots `...` are passed to the .new.MicrobialGrowth.core function.

Value

a Rosso object skeleton.

Examples

```
# First, create the skeleton.
model.object = .new.rosso.core()

# Then complete with data, functions, etc.
model.object$data$x = c(1,2,3)
model.object$data$y = c(1,2,3)
model.object$coefficients = list(N0 = 0, Nmax=0, mu=0, lambda=0)

# You can print, plot, etc., with the generic functions of MicrobialGrowth.
print(model.object)
##MicrobialGrowth, model specialized.model:
##    N0   Nmax     mu lambda
##     0      0      0      0
plot(model.object)

# Don't forget to change `isValid` to TRUE to confirm the success of the regression.
model.object$isValid = TRUE # Not a good idea here, since we have no `reg` value.
```

.parseMicrobialGrowthCreateArgs

Coefficient argument parser (create function)

Description

Parses the coefficients passed to the create function to obtain 3 values: one for the main curve and two for the confint curves. These values are sorted.

Usage

```
.parseMicrobialGrowthCreateArgs(x)
```

Arguments

x	value(s) for a given coefficient.

Value

the 3 ordered values for the given coefficient.

Examples

```
.parseMicrobialGrowthCreateArgs(1)
## [1] 1 1 1

.parseMicrobialGrowthCreateArgs(c(1,2))
## [1] 1.0 1.5 2.0

.parseMicrobialGrowthCreateArgs(c(1,2,3))
```

```
## [1] 1 2 3

.parseMicrobialGrowthCreateArgs(c(3,1,2))
## [1] 1 2 3
```

`.rosso.formula` *Rosso equation.*

Description

Rosso equation.

Usage

```
.rosso.formula(N0, Nmax, mu, lambda, base = exp(1))
```

Arguments

`N0`	initial population.
`Nmax`	final/maximum population.
`mu`	growth rate.
`lambda`	latency time.
`base`	the logarithm base used for plot y-scaling. By default, the natural logarithm is used. Set `NULL` to not scale.

Details

The output result is by default in the form `ln(N_t/N0)` (with `N_t` the population at time `t`). The base used can be modified by specifying the desired base in the `base` argument. For example, specifying `base=10` corresponds to output in the form `log_{10}(N_t/N0)`. It is possible to specify `base = NULL` to retrieve the normal `N_t` output.

Value

a function taking as input x (the time) and outputting the value of the Rosso equation.

Examples

```
f <- .rosso.formula(0.1, 2, 0.2, 5)
f(4)
## [1] 0
f(20)
## [1] 2.32998
```

Acid *Acid*

Description

Modeling of an acid with alpha its sensitivity and MIC its minimum inhibition concentration. The default concentration is 1g/L.

Usage

```
Acid(alpha, MIC, concentration = 1)
```

Arguments

alpha	sensitivity.
MIC	concentration minimale d'inhibition.
concentration	acid concentration (in g/L).

Details

The arguments alpha and MIC can be given as one to three values.

A single value means that getCoefMin, getCoefMid and getCoefMax will return the same coefficient.

Two values symbolize some sort of uncertainty about alpha and/or MIC. The functions getCoefMin and getCoefMax will use the pair (alpha, MIC) which respectively minimizes and maximizes the coefficients. The getCoefMid function will return a coefficient based on the average of the values entered.

Three values act as for two values except that for the function getCoefMid will use this third value (middle value) for the calculation of the coefficient.

Please note, entering several values acts as a pool of available values, and not as pairs (alpha, MIC). For example, the getCoefMin function will use the minimum value alpha and the minimum value MIC. If you wish to specify pairs (alpha, MIC), see Acid.SpecificPair which will determine, for example for getCoefMin, the pair (alpha, MIC) minimizing the coefficient.

Value

the acid modeled with the following accessible attributes:

alpha	the alpha value or list of values.
MIC	the MIC value or list of values.
concentration	the acid concentration (in g/L).
getCoefMin	function returning the minimum coefficient to apply to a MicrobialGrowth-object (see details section).
getCoefMid	function returning the "middle" coefficient to apply to a MicrobialGrowth-object (see details section).
getCoefMax	function returning the maximum coefficient to apply to a MicrobialGrowth-object (see details section).

See Also

Acid.SpecificPair

Examples

```
# Classic instantiation
aceticAcid <- Acid(1.245, 5.47)
print(aceticAcid)
## acid {alpha=1.245, MIC=5.47g/L, concentration=1g/L}

# Classic instantiation by specifying a concentration
print( Acid(1.245, 5.47, 3) )
## acid {alpha=1.245, MIC=5.47g/L, concentration=3g/L}

# Instantiation with multiple `alpha` and `MIC` values (see details section)
print( Acid(c(0.98, 1.1, 1.51), c(5.26, 5.68)) )
## acid {alpha=[0.98, 1.1, 1.51], MIC=[5.26, 5.68]g/L, concentration=1g/L}

# Generic operators (`+`, `*`, etc.) can change the `concentration` of the acid.
print(aceticAcid / 2)
## acid {alpha=1.245, MIC=5.47g/L, concentration=0.5g/L}
print(aceticAcid + 2)
## acid {alpha=1.245, MIC=5.47g/L, concentration=3g/L}

# Without having to pre-define specific concentrations, and with the default `concentration` (1g/L),
# you can dynamically change the acid concentration as follows:
for (concentration in c(0.5, 1, 5, 10)) {
  print(concentration * aceticAcid)
}
## acid {alpha=1.245, MIC=5.47g/L, concentration=0.5g/L}
## acid {alpha=1.245, MIC=5.47g/L, concentration=1g/L}
## acid {alpha=1.245, MIC=5.47g/L, concentration=5g/L}
## acid {alpha=1.245, MIC=5.47g/L, concentration=10g/L}

try({
  # Acid can be applied to a MicrobilogicalGrowth-object with the `+` addition operator.
  # Note that the acid should be on the right side, otherwise an error is raised.
  MyMicrobialGrowthObject + aceticAcid
  ## returns the MicrobialGrowth-object affected by the acid (several acids can be applied)
})
```

`Acid.SpecificPair` *Acid - specific pair (*`alpha`, `MIC`*)*

Description

Modeling of an acid with `alpha` its sensitivity and `MIC` its minimum inhibition concentration. The default `concentration` is 1g/L.

Usage

```
Acid.SpecificPair(pairs, concentration = 1)
```

Arguments

pairs	list of pairs (alpha, MIC).
concentration	acid concentration (in g/L).

Details

The pairs argument can be given as one to three pairs.

A single pair means that getCoefMin, getCoefMid and getCoefMax will return the same coefficient.

Two pairs means that one of them will be used for getCoefMin and the other for getCoefMax. The getCoefMid function will use an average value of the two pairs.

Three pairs acts like two pairs except that the getCoefMid function will use this third pair (middle value) to calculate the coefficient. Note that the pair (alpha, MIC) used by getCoefMid neither minimizes nor maximizes the coefficient (in other words, it is the pair which is neither used in getCoefMin nor in getCoefMax).

Please note that if you do not want to use specific pairs but ranges of values for alpha and/or MIC, use the parent function Acid.

Value

the acid modeled with the following accessible attributes:

pairs	list of pairs (alpha, MIC).
concentration	the acid concentration (in g/L).
getCoefMin	function returning the minimum coefficient to apply to a MicrobialGrowth-object (see details section).
getCoefMid	function returning the "middle" coefficient to apply to a MicrobialGrowth-object (see details section).
getCoefMax	function returning the maximum coefficient to apply to a MicrobialGrowth-object (see details section).

See Also

Acid

Examples

```
# Classic instantiation
print(Acid.SpecificPair(list(c(1.245, 5.47))))
## acid {{alpha=1.245, MIC=5.47g/L}, concentration=1g/L}

# Classic instantiation by specifying a concentration
print(Acid.SpecificPair(list(c(1.245, 5.47)), 3))
```

```
## acid {{alpha=1.245, MIC=5.47g/L}, concentration=3g/L}

# Instantiation with multiple couples (`alpha`, `MIC`) (see details section)
aceticAcid <- Acid.SpecificPair(list(c(0.98,5.68),c(1.51,5.26)))
print(aceticAcid)
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=1g/L}

# Generic operators (`+`, `*`, etc.) can change the `concentration` of the acid.
print(aceticAcid / 2)
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=0.5g/L}
print(aceticAcid + 2)
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=3g/L}

# Without having to pre-define specific concentrations, and with the default `concentration` (1g/L),
# you can dynamically change the acid concentration as follows:
for (concentration in c(0.5, 1, 5, 10)) {
  print(concentration * aceticAcid)
}
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=0.5g/L}
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=1g/L}
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=5g/L}
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=10g/L}

try({
  # Acid can be applied to a MicrobilogicalGrowth-object with the `+` addition operator.
  # Note that the acid should be on the right side, otherwise an error is raised.
  MyMicrobialGrowthObject + aceticAcid
 ## returns the MicrobialGrowth-object affected by the acid (several acids can be applied)
})
```

`baranyi.create` *Baranyi create function*

Description

Baranyi-object creator from the 4 biological meaning parameters.

Usage

```
baranyi.create(N0, Nmax, mu, lambda, xlim, n = 101)
```

Arguments

N0	initial population.
Nmax	final/maximum population.
mu	growth rate.
lambda	latency time.
xlim	range of values to simulate x and y (and hence plotting, *etc.*)
n	number of points to simulate in the interval xlim.

Value

a Baranyi-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.
message	always with this method.
reg	always with this method.

See Also

MicrobialGrowth.create

---

example_data *Example data from the MicrobialGrowth package*

Description

TODO : Describe them (origine, type, etc.)

Author(s)

Clarisse Breard <clarisse.breard@inrae.fr>

getCreateFunctionName *Create function name getter*

Description

Returns the name of the creation function associated with the model.

Usage

```
getCreateFunctionName(model)
```

Arguments

model	the model name.

Value

the string corresponding to the creation function of the model. Warning, this function does not check the existence of the corresponding function.

Examples

```
getCreateFunctionName("gompertz")
## [1] "gompertz.create"

# Note that this does not verify the existence
getCreateFunctionName("NonExistentFunction")
## [1] "NonExistentFunction.create"
```

getFormula *Formula getter*

Description

Returns the formula associated with the specified model.

Usage

```
getFormula(model)
```

Arguments

model	the model name.

Value

the function corresponding to the formula of the model.

Examples

```
f <- getFormula("gompertz")
# We need to set the parameters (N0, ..., lambda)
f2 <- f(0.1, 2, 0.2, 5)
# And we can then use the function "f(x)" with x the time
f2(4)
## [1] 0.1150952

# The same, more direct
F <- getFormula("gompertz")(0.1, 2, 0.2, 5)
F(4)
## [1] 0.1150952
```

`getFunctionName` *Regression function name getter*

Description

Returns the name of the regression function associated with the model.

Usage

```
getFunctionName(model)
```

Arguments

`model`	the model name.

Value

the string corresponding to the regression function of the model. Warning, this function does not check the existence of the corresponding function.

Examples

```
getFunctionName("gompertz")
## [1] ".MicrobialGrowth.gompertz"

# Note that this does not verify the existence
getFunctionName("NonExistentFunction")
## [1] ".MicrobialGrowth.NonExistentFunction"
```

getModelName *Model name getter*

Description

Returns the name of the model used.

Usage

```
getModelName(x)
```

Arguments

x	a MicrobialGrowth-object

Details

scans the classes of the object which must correspond on the one hand to the generic class "MicrobialGrowth" and on the other hand to the class-model. It is this second that is returned.

Value

the name of the model used.

Examples

```
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")
getModelName(g)
## [1] "gompertz"
```

gompertz.create *Gompertz create function*

Description

Gompertz-object creator from the 4 biological meaning parameters.

Usage

```
gompertz.create(N0, Nmax, mu, lambda, xlim, n = 101)
```

Arguments

N0	initial population.
Nmax	final/maximum population.
mu	growth rate.
lambda	latency time.
xlim	range of values to simulate x and y (and hence plotting, *etc.*)
n	number of points to simulate in the interval xlim.

Value

a Gompertz-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.
message	always with this method.
reg	always with this method.

See Also

MicrobialGrowth.create

---

gompertz.explain *Graphical Example of Modified Gompertz Equation.*

Description

Graphical Example of Modified Gompertz Equation.

Usage

```
gompertz.explain(
  N0 = 0.14,
  Nmax = 1.43,
  mu = 0.07,
  lambda = 40,
  xlim = c(0, 100)
)
```

Arguments

N0	initial population.
Nmax	final/maximum population.
mu	growth rate.
lambda	latency time.
xlim	range of values to simulate x and y.

Value

No return value, called to plot a MicrobialGrowth object with the Gompertz model to illustrate the different coefficients.

Examples

```
gompertz.explain()

gompertz.explain(0.15, 2, 0.1, 40, c(0,100))
```

---

is.acid *Acid class*

Description

Test if a variable is an Acid-object.

Usage

```
is.acid(x)
```

Arguments

x	variable to test.

Value

TRUE if the object is of class acid.

Examples

```
# TRUE return
is.acid( Acid(1.245, 5.47, 3) )
is.acid( Acid(c(0.98, 1.1, 1.51), c(5.26, 5.68)) )
# Acid.SpecificPair-objects are also Acid-objects
is.acid( Acid.SpecificPair(list(c(0.98, 5.68), c(1.51, 5.26))) )

# FALSE return
is.acid(1)
is.acid( list(Acid(1.245, 5.47, 3), Acid(1.245, 5.47, 3)) )
```

is.acid.specific.pair *Acid.SpecificPair class*

Description

Test if a variable is an Acid.SpecificPair-object.

Usage

```
is.acid.specific.pair(x)
```

Arguments

x	variable to test.

Value

TRUE if the object is of class acid.specific.pair.

Examples

```
# TRUE return
is.acid.specific.pair( Acid.SpecificPair(list(c(0.98, 5.68), c(1.51, 5.26))) )

# FALSE return
is.acid.specific.pair(1)
is.acid.specific.pair( Acid(1.245, 5.47, 3) )
is.acid.specific.pair( list(Acid(1.245, 5.47, 3), Acid(1.245, 5.47, 3)) )
```

is.baranyi *Baranyi class*

Description

Test if a variable or variable list is/are baranyi-object(s).

Usage

```
is.baranyi(x)
```

Arguments

x	variable or list.

Value

TRUE if the object or all objects are of class baranyi.

Examples

```
# TRUE return
r1 <- MicrobialGrowth(example_data$time, example_data$y1, model="baranyi")
r2 <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                    xlim = c(0, 100), model="baranyi")
is.baranyi(r1)
is.baranyi(r2)
is.baranyi(c(r1, r2))
is.baranyi(list(r1,r2))

# FALSE return
is.baranyi(1)
is.baranyi(list())
is.baranyi(c(r1, r2, 1))
is.baranyi(list(r1, r2, 1))
```

`is.gompertz` *Gompertz class*

Description

Test if a variable or variable list is/are gompertz-object(s).

Usage

```
is.gompertz(x)
```

Arguments

`x`	variable or list.

Value

`TRUE` if the object or all objects are of class `gompertz`.

Examples

```
# TRUE return
g1 <- MicrobialGrowth(example_data$time, example_data$y1)
g2 <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                    xlim = c(0, 100), model="gompertz")
is.gompertz(g1)
is.gompertz(g2)
is.gompertz(c(g1, g2))
is.gompertz(list(g1,g2))

# FALSE return
is.gompertz(1)
is.gompertz(list())
```

```
is.gompertz(c(g1, g2, 1))
is.gompertz(list(g1, g2, 1))
```

`is.linear` *Linear class*

Description

Test if a variable or variable list is/are linear-object(s).

Usage

```
is.linear(x)
```

Arguments

x	variable or list.

Value

`TRUE` if the object or all objects are of class `linear`.

Examples

```
# TRUE return
r1 <- MicrobialGrowth(example_data$time, example_data$y1, model="linear")
r2 <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                    xlim = c(0, 100), model="linear")
is.linear(r1)
is.linear(r2)
is.linear(c(r1, r2))
is.linear(list(r1,r2))

# FALSE return
is.linear(1)
is.linear(list())
is.linear(c(r1, r2, 1))
is.linear(list(r1, r2, 1))
```

is.MicrobialGrowth *MicrobialGrowth class*

Description

Test if a variable or variable list is/are MicrobialGrowth-object(s).

Usage

```
is.MicrobialGrowth(x)
```

Arguments

x	variable or list.

Value

TRUE if the object or all objects are of class MicrobialGrowth.

Examples

```
# TRUE return
g1 <- MicrobialGrowth(example_data$time, example_data$y1)
g2 <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                  xlim = c(0, 100), model="gompertz")
is.MicrobialGrowth(g1)
is.MicrobialGrowth(g2)
is.MicrobialGrowth(list(g1,g2))

# FALSE return
is.MicrobialGrowth(1)
is.MicrobialGrowth(list())
is.MicrobialGrowth(c(g1, g2))
is.MicrobialGrowth(list(g1, g2, 1))
```

is.rosso *Rosso class*

Description

Test if a variable or variable list is/are rosso-object(s).

Usage

```
is.rosso(x)
```

Arguments

x	variable or list.

Value

TRUE if the object or all objects are of class rosso.

Examples

```
# TRUE return
r1 <- MicrobialGrowth(example_data$time, example_data$y1, model="rosso")
r2 <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                    xlim = c(0, 100), model="rosso")
is.rosso(r1)
is.rosso(r2)
is.rosso(c(r1, r2))
is.rosso(list(r1,r2))

# FALSE return
is.rosso(1)
is.rosso(list())
is.rosso(c(r1, r2, 1))
is.rosso(list(r1, r2, 1))
```

---

linear.create *Linear create function*

Description

Linear-object creator from the 4 biological meaning parameters.

Usage

```
linear.create(N0, Nmax, mu, lambda, xlim, n = 101)
```

Arguments

N0	initial radius.
Nmax	final/maximum radius.
mu	growth rate.
lambda	latency time.
xlim	range of values to simulate x and y (and hence plotting, *etc.*)
n	number of points to simulate in the interval xlim.

Value

a Linear-object composed of

`call`	the matched call with several components.
`coefficients`	coefficients obtained by regression.
`data`	data used for regression, once the y values are clipped
`f`	a list of functions such as `formula` to retrieve the function of the model with the coefficients obtained by regression, `confint` to retrieve the confidence intervals, etc.
`isValid`	a boolean indicating whether the regression was successful or not.
`message`	always with this method.
`reg`	always with this method.

See Also

MicrobialGrowth.create

`listAvailableModels` *List available models.*

Description

List available models.

Usage

```
listAvailableModels()
```

Details

lists the models by scanning the available ".MicrobialGrowth.m" regression functions, with "m" the name of the model.

Value

the list of available models.

Examples

```
listAvailableModels()
## [1] "baranyi"  "gompertz"  "rosso"
```

`MicrobialGrowth` *MicrobialGrowth regression function*

Description

Regression function to different microbial growth models.

Usage

```
MicrobialGrowth(
  x,
  y,
  model = "gompertz",
  clip = c(-Inf, Inf),
  start = list(),
  lower = list(),
  upper = list(),
  nls.args = list(),
  callbackError = NULL,
  ...
)
```

Arguments

`x`	index series or time series.
`y`	values or list of values to regress (should not be logged).
`model`	wanted growth model : "baranyi", "gompertz" or "rosso".
`clip`	a pair of values indicating in which interval to clip the data `y`. When `clip` is missing, default values are used.
`start`	a named list of starting estimates. When `start` is missing, default values are used.
`lower`	a named list of lower bounds. When `lower` is missing, default values are used.
`upper`	a named list of upper bounds. When `upper` is missing, default values are used.
`nls.args`	additional parameters to use when calling nls.
`callbackError`	function to call on error during regression.
`...`	further arguments passed to or from other methods.

Details

Use `listAvailableModels()` function to see all values accepted by `model` parameter.

The default values for `clip`, `start`, `lower` and `upper` are calculated based on the given data. These default values can be known through the `call` member of the returned value.

The `nls.args` argument is a list that can contain any nls function argument except `formula`, `algorithm`, `start`, `lower` and `upper` which are already fixed (via a homonymous or hard-coded argument).

For the `callbackError` argument, prefer the `stop` function to block or `warning` to not be blocking.

Value

a MicrobialGrowth-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.
message	contains the error message if the regression fails, NULL otherwise.
reg	the nls object returned by the nls function.

Examples

```
# Using the embedded data example_data
# Simple example
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")

# Multiple regression example
G <- MicrobialGrowth(example_data$time, example_data[2:ncol(example_data)], model="gompertz")

# Example of multiple parameter changes
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz",
                           clip = c(0.15, Inf), start = list(N0=0.1, Nmax=2,
                           mu=0.05, lambda=40), lower = list(lambda = 40))

# Example of using `nls.args` to apply weight to some data
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz",
nls.args = list(weights = (function(x){(x >= 50 & x <= 70)*9 + 1})(example_data$time)))

# Example of callbackError (remaining non-blocking)
g <- MicrobialGrowth(example_data$time, example_data$y15, model="gompertz",
                           callbackError = warning)

# Example of callbackError (becoming blocking)
try(
  g <- MicrobialGrowth(c(1,2,3,4,5),c(1,1,1,1,1), model="gompertz", callbackError = stop)
)
```

MicrobialGrowth.create

MicrobialGrowth create function

Description

MicrobialGrowth-object creator from the 4 biological meaning parameters.

Usage

```
MicrobialGrowth.create(
  N0,
  Nmax,
  mu,
  lambda,
  xlim,
  model = "gompertz",
  n = 101,
  ...
)
```

Arguments

N0	initial population.
Nmax	final/maximum population.
mu	growth rate.
lambda	latency time.
xlim	range of values to simulate x and y (and hence plotting, *etc.*)
model	wanted growth model: "baranyi", "gompertz" or "rosso"
n	number of points to simulate in the interval xlim.
...	further arguments passed to or from other methods.

Details

The N0, Nmax, mu and lambda parameter-coefficients can be given as one to three values.

A single value means that the coefficient and the confidence interval values will be identical.

Two values means that they will correspond to the confidence interval, and the coefficient will be calculated as the average of these two values.

Three values means that each of these values will be associated with the coefficient or the confidence interval.

Values are always sorted automatically, which means that c(2,1,3) is equivalent to c(1,2,3).

Value

a MicrobialGrowth-object composed of

call	the matched call with several components.
coefficients	coefficients obtained by regression.
data	data used for regression, once the y values are clipped
f	a list of functions such as formula to retrieve the function of the model with the coefficients obtained by regression, confint to retrieve the confidence intervals, etc.
isValid	a boolean indicating whether the regression was successful or not.
message	always with this method.
reg	always with this method.

Examples

```
# Simple example
g <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
     xlim = c(0, 100), model="gompertz")

# Example from a regression (whose values can be stored and then reused later)
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")
c <- g$coefficients
g2 <- MicrobialGrowth.create(c$N0, c$Nmax, c$mu, c$lambda,
      xlim = c(min(g$data$x),max(g$data$x)), n = length(g$data$x), model="gompertz")

# Example with confidence intervals
g <- MicrobialGrowth.create(N0 = c(0.13, 0.15), Nmax = 1.43, mu = c(0.05, 0.07, 0.09),
lambda = c(45, 49, 43), xlim = c(0, 100), model="gompertz")
# Coefficient N0 is 0.14 and the confidence interval is (0.13, 0.15)
# Coefficient Nmax is 1.43 and the confidence interval is (1.43, 1.43)
# Coefficient mu is 0.07 and the confidence interval is (0.05, 0.09)
# Coefficient lambda is 45 and the confidence interval is (43, 49)
```

Ops.acid *Operators on the Acid Class*

Description

Operators for the "Acid" class.

Usage

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

Arguments

e1	acid-object, numeric or MicrobialGrowth-object.
e2	acid-object or numeric.

Details

Operations between an acid and a numeric are the most common case. In this case, the operation is carried out on the `concentration` member of the acid. A new acid-object is returned with the new concentration.

Operations between acids are tolerated (but not recommended). To do this, the two acids must have the same `alpha` sensitivity and the same `MIC`, and the operation is carried out between the concentrations of the two acids. A new acid-object is returned with the new concentration.

The addition operator + can be used between MicrobialGrowth-object (left side) and an acid-object (right side). This operation symbolizes the application of the acid to the MicrobialGrowth-object. A new MicrobialGrowth-object is returned with its coefficients (and confidence intervals) modified by the acid.

Value

a new acid or MicrobialGrowth-object.

Examples

```
# Acids and numerics
print( Acid(1.245, 5.47) * 5 )
## acid {alpha=1.245, MIC=5.47g/L, concentration=5g/L}

print( Acid(1.245, 5.47) / 3 )
## acid {alpha=1.245, MIC=5.47g/L, concentration=0.333333333333333g/L}

print( 3 / Acid(1.245, 5.47) )
## acid {alpha=1.245, MIC=5.47g/L, concentration=3g/L}

print( 3 / Acid(1.245, 5.47, 0.5) )
## acid {alpha=1.245, MIC=5.47g/L, concentration=6g/L}

# Acids and acids
print( Acid(1.245, 5.47, 0.5) + Acid(1.245, 5.47, 2) )
## acid {alpha=1.245, MIC=5.47g/L, concentration=2.5g/L}

try({
  print( Acid(1.245, 5.47, 0.5) + Acid(1, 5.47, 2) )
  ## throws an error since `alpha` and/or `MIC` are not equal
})

# Acids and MicrobialGrowth-object
g <- MicrobialGrowth.create(N0 = c(0.13, 0.15), Nmax = 1.43, mu = c(0.05, 0.07, 0.09),
lambda = c(45, 49, 43), xlim = c(0, 100), model="gompertz")
aceticAcid <- Acid(1.245, 5.47)
{
  cat("Before :\n")
  print(g)
  cat("After:\n")
  print(g + aceticAcid)
}
## Before :
## MicrobialGrowth, model gompertz:
##     N0   Nmax     mu lambda
##   0.14   1.43   0.07  45.00
## After:
## MicrobialGrowth, model gompertz:
##          N0        Nmax          mu      lambda
##  0.14000000  1.43000000  0.06156075 51.16896670

# Also works with the `acid.specific.pair` subclass
print( Acid.SpecificPair(list(c(0.98, 5.68), c(1.51, 5.26))) )
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=6g/L}
```

plot.gompertz *Gompertz plot function*

Description

Default plot.MicrobialGrowth function can be overwritten with the following function

Usage

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

Arguments

x	gompertz-object.
...	further arguments passed to or from other methods.

Value

No return value, called to plot a MicrobialGrowth-object based on the Gompertz model.

See Also

plot.MicrobialGrowth

plot.linear *Linear plot function*

Description

Default plot.MicrobialGrowth function can be overwritten with the following function

Usage

```
## S3 method for class 'linear'
plot(x, base = NULL, ...)
```

Arguments

x	linear-object.
base	base used for plot y-scaling.
...	further arguments passed to or from other methods.

Value

No return value, called to plot a MicrobialGrowth-object based on the linear model.

See Also

plot.MicrobialGrowth

`plot.MicrobialGrowth` *MicrobialGrowth plot function*

Description

Plot function of MicrobialGrowth-objects.

Usage

```
## S3 method for class 'MicrobialGrowth'
plot(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  n = 101,
  base = exp(1),
  display.coefficients = TRUE,
  display.model = TRUE,
  display.confint = FALSE,
  reg.args = list(col = "red"),
  title.args = list(line = 2),
  model.args = list(side = 4, line = 0),
  coefficients.args = list(cex = 0.9, line = 0.2, side = 3),
  confint.args = list(),
  ...
)
```

Arguments

`x`	MicrobialGrowth-object.
`main`	main title for the plot.
`xlab`	title for the x axis.
`ylab`	title for the y axis.
`n`	the number of x values at which to evaluate. See details section.
`base`	the logarithm base used for plot y-scaling. By default, the natural logarithm is used. Set `NULL` to not scale.
`display.coefficients`	boolean indicating the display or not of the values of coefficients (under the main title).
`display.model`	boolean indicating the model used for regression (on right side).

`display.confint`
: boolean indicating the display or not of confidence intervals (in the form of curves and area).

`reg.args`
: customization parameters of the curve obtained by regression (see curve for possible parameters).

`title.args`
: title customization parameters `main`, `xlab` and `ylab` (see title for possible parameters).

`model.args`
: model display customization parameters (see mtext for possible parameters).

`coefficients.args`
: coefficient display customization parameters (see mtext for possible parameters).

`confint.args`
: parameters for customizing the plotting of curves and area, corresponding to the confidence interval (see details section).

`...`
: other graphical parameters (see plot).

Details

Similar to the `curve` function, the `n` argument corresponds to the number of points evaluated to draw the curves of regression, confidence bounds and the associated area. Increase its value for a more accurate representation.

When `base` is not `NULL`, the plot produced is $log_n(N/N0)$, where `n` is the value specified in the `base` argument.

Value

No return value, called to plot a MicrobialGrowth-object.

Examples

```
# Example plot of a MicrobialGrowth-object obtained by regression
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")
plot(g)

# Example plot of a user-created MicrobialGrowth-object (via MicrobialGrowth.create)
g <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                                  xlim = c(0, 100), model="gompertz")
plot(g)

# Example plot with usual graphical parameters
plot(g, pch = 4, cex = 2, col = "blue", xlab = "Time (hours)", main = "Gompertz regression")

# Example of plot hiding the coefficients and customizing the curve obtained by regression
plot(g, display.coefficients = FALSE, reg.args = list(col = "green", lty = 2, lwd = 5))

# Example of a plot displaying the curves and area of the confidence interval
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")
plot(g, display.confint = TRUE)

# Example of a plot customizing the confidence interval
```

```
plot(g, xlim = c(80, 100), ylim = c(1.8, 2.4), # Zoom in to see the example better
display.confint = TRUE,
confint.args = list(
  lines = list(col = "purple", lty = 2, lwd = 2),
  area = list(col = "green", opacity = 0.1)
))

# Example of a plot customizing the display of coefficients and titles
plot(g, main = "Gompertz",
coefficients.args = list(cex = 1.5, side = 4, line = 1),
title.args = list(col.main = "blue", col.lab = "red"))
```

`print.acid` *Acid print function*

Description

Print function of Acid-object.

Usage

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

Arguments

`x`	Acid-object.
`...`	further arguments passed to or from other methods.

Value

No return value, called to print information about a Acid-object.

See Also

Acid

Examples

```
print( Acid(1.245, 5.47, 3) )
## acid {alpha=1.245, MIC=5.47g/L, concentration=3g/L}

print( Acid(c(0.98, 1.1, 1.51), c(5.26, 5.68)) )
## acid {alpha=[0.98, 1.1, 1.51], MIC=[5.26, 5.68]g/L, concentration=1g/L}
```

print.acid.specific.pair *Acid.SpecificPair print function*

Description

Print function of Acid.SpecificPair-object.

Usage

```
## S3 method for class 'acid.specific.pair'
print(x, sep = ",\n      ", ...)
```

Arguments

x	Acid.SpecificPair-object.
sep	a character string to separate the different pairs.
...	further arguments passed to or from other methods.

Value

No return value, called to print information about a Acid.SpecificPair-object.

See Also

Acid.SpecificPair

Examples

```
print( Acid.SpecificPair(list(c(1.245, 5.47)), 3) )
## acid {{alpha=1.245, MIC=5.47g/L}, concentration=3g/L}

print( Acid.SpecificPair(list(c(0.98, 5.68), c(1.51, 5.26))) )
## acid {{alpha=0.98, MIC=5.68g/L},
##       {alpha=1.51, MIC=5.26g/L}, concentration=1g/L}
```

print.gompertz *Gompertz print function*

Description

Default print.MicrobialGrowth function can be overwritten with the following function

Usage

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

Arguments

x	gompertz-object.
...	further arguments passed to or from other methods.

Value

No return value, called to print information about a MicrobialGrowth-object based on the Gompertz model.

See Also

print.MicrobialGrowth

---

print.MicrobialGrowth *MicrobialGrowth print function*

Description

Print function of MicrobialGrowth-objects.

Usage

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

Arguments

x	MicrobialGrowth-object.
...	further arguments passed to or from other methods.

Value

No return value, called to print information about a MicrobialGrowth-object.

See Also

MicrobialGrowth, MicrobialGrowth.create

Examples

```
# Print from regressed MicrobialGrowth-object
g <- MicrobialGrowth(example_data$time, example_data$y1, model="gompertz")
print(g) # or just `g` in the console

# Print from a user-created MicrobialGrowth-object (via MicrobialGrowth.create)
g <- MicrobialGrowth.create(N0 = 0.14, Nmax = 1.43, mu = 0.07, lambda = 45,
                            xlim = c(0, 100), model="gompertz")
print(g) # or just `g` in the console
```

`rosso.create` *Rosso create function*

Description

Rosso-object creator from the 4 biological meaning parameters.

Usage

```
rosso.create(N0, Nmax, mu, lambda, xlim, n = 101)
```

Arguments

`N0`	initial population.
`Nmax`	final/maximum population.
`mu`	growth rate.
`lambda`	latency time.
`xlim`	range of values to simulate `x` and `y` (and hence plotting, *etc.*)
`n`	number of points to simulate in the interval `xlim`.

Value

a Rosso-object composed of

`call`	the matched call with several components.
`coefficients`	coefficients obtained by regression.
`data`	data used for regression, once the `y` values are clipped
`f`	a list of functions such as `formula` to retrieve the function of the model with the coefficients obtained by regression, `confint` to retrieve the confidence intervals, etc.
`isValid`	a boolean indicating whether the regression was successful or not.
`message`	always with this method.
`reg`	always with this method.

See Also

MicrobialGrowth.create

summary.MicrobialGrowth *MicrobialGrowth summary function*

Description

Summarizes the regression of an MicrobialGrowth-object.

Usage

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

Arguments

object	MicrobialGrowth-object.
...	additional arguments affecting the summary produced.

Details

Equivalent to summary(MicrobialGrowthObject$reg, ...) to which we add the corresponding model member and the summary.MicrobialGrowth class.

Value

The summary of the successful regression, NULL otherwise.

Examples

```
# Simple example
g <- MicrobialGrowth(example_data$time, example_data$y1)
summary(g)

# Example without summary available
g <- MicrobialGrowth(example_data$time, example_data$y15)
summary(g)

g <- MicrobialGrowth.create(0.14, 1.5, 0.07, 45, c(0,100), model="gompertz")
summary(g)
```

THRESHOLD_FEW_DATA *Threshold when little data is used in a regression*

Description

Number of data below which the usual methods for choosing starting values (`start`) and limits (`lower` and `upper`) will not be used in favor of a secondary method more suited to the low number of data.

Usage

```
THRESHOLD_FEW_DATA
```

Format

An object of class `numeric` of length 1.

Index