Package 'growthrates'

August 12, 2022

Encoding UTF-8

Type Package

Title Estimate Growth Rates from Experimental Data

Version 0.8.3

Date 2022-08-12

LazyData yes

Maintainer Thomas Petzoldt <thomas.petzoldt@tu-dresden.de>

Description A collection of methods to determine growth rates from
experimental data, in particular from batch experiments and
plate reader trials.

Depends R (>= 3.2), lattice, deSolve

Imports stats, graphics, methods, parallel, utils, FME

Suggests knitr, rmarkdown

License GPL (>= 2)

URL https://github.com/tpetzoldt/growthrates

VignetteBuilder knitr

RoxygenNote 7.2.1

Collate 'aaa_growthmodel-class.R' 'aab_growthmodel-constructor.R'
'aac_classes.R' 'aad_set_generics.R' 'all_easylinear.R'
'all_growthmodels.R' 'all_splines.R' 'antibiotic.R'
'bactgrowth.R' 'cost.R' 'extcoef_logistic.R' 'extract.R'
'fit_easylinear.R' 'fit_growthmodel.R' 'fit_spline.R'
'grow_baranyi.R' 'grow_exponential.R' 'grow_genlogistic.R'
'grow_gompertz.R' 'grow_gompertz2.R' 'grow_huang.R'
'grow_logistic.R' 'grow_richards.R' 'grow_twostep.R'
'growthrates-package.R' 'lm_window.R' 'methods.R'
'multisplit.R' 'names.R' 'parse_formula.R'
'parse_formula_nonlin.R' 'plot.R' 'predict.R' 'utilities.R'

NeedsCompilation yes

Author Thomas Petzoldt [aut, cre] (<https://orcid.org/0000-0002-4951-6468>)

Repository CRAN

Date/Publication 2022-08-12 12:20:09 UTC

R **topics documented:**

---

growthrates-package   *Estimate Growth Rates from Experimental Data*

Description

A collection of methods to determine growth rates from experimental data, in particular from batch experiments and plate reader trials.

The package contains basically three methods:

- fit a linear regression to a subset of data with the steepest log-linear increase (a method, similar to Hall et al., 2013),

- fit parametric nonlinear models to the complete data set, where the model functions can be given either in closed form or as numerically solved (system of) differential equation(s),

- use maximum of the 1st derivative of a smoothing spline with log-transformed y-values (similar to Kahm et al., 2010).

The package can fit data sets of single experiments or complete series containing multiple data sets. Included are functions for extracting estimates and for plotting. The package supports growth models given as numerically solved differential equations. Multi-core computation is used to speed up fitting of parametric models.

Author(s)

Thomas Petzoldt

References

Hall, B. G., Acar, H. and Barlow, M. 2013. Growth Rates Made Easy. Mol. Biol. Evol. 31, 232-238, doi:10.1093/molbev/mst197

Kahm, M., Hasenbrink, G., Lichtenberg-Frate, H., Ludwig, J., Kschischo, M. 2010. grofit: Fitting Biological Growth Curves with R. Journal of Statistical Software, 33(7), 1-21, doi:10.18637/jss.v033.i07

Soetaert, K. and Petzoldt, T. 2010. Inverse Modelling, Sensitivity and Monte Carlo Analysis in R Using Package FME. Journal of Statistical Software, 33(3), 1-28, doi:10.18637/jss.v033.i03

Soetaert, K., Petzoldt, T. Setzer, R. W. 2010. Solving Differential Equations in R: Package deSolve. Journal of Statistical Software, 33(9), 1-25, doi:10.18637/jss.v033.i09

See Also

fit_easylinear, fit_spline, fit_growthmodel, all_easylinear, all_splines, all_growthmodels

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))

## get table from single experiment
dat <- splitted.data[["D:0:1"]]

fit1 <- fit_spline(dat$time, dat$value)
plot(fit1, log="y")
plot(fit1)

## derive start parameters from spline fit
p <- coef(fit1)

## subset of first 10 data
first10 <-  dat[1:10, ]
fit2 <- fit_growthmodel(grow_exponential, p=p, time=first10$time, y=first10$value)

## use parameters from spline fit and take K from the data maximum
p <- c(coef(fit1), K = max(dat$value))
```

```
fit3 <- fit_growthmodel(grow_logistic, p=p, time=dat$time, y=dat$value, transform="log")

plot(fit1)
lines(fit2, col="green")
lines(fit3, col="red")
```

---

all_easylinear   *Easy Growth Rates Fit to data Frame*

Description

Determine maximum growth rates from log-linear part of the growth curve for a series of experiments.

Usage

```
all_easylinear(...)

## S3 method for class 'formula'
all_easylinear(formula, data, h = 5, quota = 0.95, subset = NULL, ...)

## S3 method for class 'data.frame'
all_easylinear(
  data,
  grouping,
  time = "time",
  y = "value",
  h = 5,
  quota = 0.95,
  ...
)
```

Arguments

`...`	generic parameters, reserved for future extensions.
`formula`	model formula specifying dependent, independent and grouping variables in the form: dependent ~ independent | group1 + group2 + ...
`data`	data frame of observational data.
`h`	with of the window (number of data).
`quota`	part of window fits considered for the overall linear fit (relative to max. growth rate).
`subset`	a specification of the rows to be used: defaults to all rows.
`grouping`	model formula or character vector of criteria defining subsets in the data frame.
`time`	character vectors with name independent variabl.e.
`y`	character vector with name of dependent variable

Value

object with parameters of all fits.

References

Hall, BG., Acar, H, Nandipati, A and Barlow, M (2014) Growth Rates Made Easy. Mol. Biol. Evol. 31: 232-38, doi:10.1093/molbev/mst187

See Also

Other fitting functions: all_growthmodels(), all_splines(), fit_easylinear(), fit_growthmodel(), fit_spline()

Examples

```
library("growthrates")
L <- all_easylinear(value ~ time | strain + conc + replicate, data=bactgrowth)
summary(L)
coef(L)
rsquared(L)

results <- results(L)

library(lattice)
xyplot(mumax ~ conc|strain, data=results)
```

---

all_growthmodels   *Fit Nonlinear Growth Models to Data Frame*

Description

Determine maximum growth rates by nonlinear fits for a series of experiments.

Usage

```
all_growthmodels(...)

## S3 method for class 'formula'
all_growthmodels(
  formula,
  data,
  p,
  lower = -Inf,
  upper = Inf,
```

```
  which = names(p),
  FUN = NULL,
  method = "Marq",
  transform = c("none", "log"),
  ...,
  subset = NULL,
  ncores = detectCores(logical = FALSE)
)

## S3 method for class '`function`'
all_growthmodels(
  FUN,
  p,
  data,
  grouping = NULL,
  time = "time",
  y = "value",
  lower = -Inf,
  upper = Inf,
  which = names(p),
  method = "Marq",
  transform = c("none", "log"),
  ...,
  ncores = detectCores(logical = FALSE)
)
```

Arguments

...	generic parameters, including parameters passed to the optimizer.
formula	model formula specifying dependent, independent and grouping variables in the form: dependent ~ independent | group1 + group2 + ...
data	data frame of observational data.
p	named vector of start parameters and initial values of the growth model.
lower	lower bound of the parameter vector.
upper	upper bound of the parameter vector.
which	vector of parameter names that are to be fitted.
FUN	function of growth model to be fitted.
method	character vector specifying the optimization algorithm.
transform	fit model to non-transformed or log-transformed data.
subset	a specification of the rows to be used: defaults to all rows.
ncores	number of CPU cores used for parallel computation. The number of real cores is detected automatically by default, but fort debugging purposes it could be wise to set ncores = 1. Usage of logical (hyperthreading) cores does not speed up computation.
grouping	vector of grouping variables defining subsets in the data frame.

time	character vector with name of independent variable.
y	character vector with name of dependent variable.

Value

object containing the parameters of all fits.

See Also

Other fitting functions: all_easylinear(), all_splines(), fit_easylinear(), fit_growthmodel(), fit_spline()

Examples

```
data(bactgrowth)
splitted.data <- multisplit(value ~ time | strain + conc + replicate,
                 data = bactgrowth)

## show which experiments are in splitted.data
names(splitted.data)

## get table from single experiment
dat <- splitted.data[["D:0:1"]]

fit0 <- fit_spline(dat$time, dat$value)

fit1 <- all_splines(value ~ time | strain + conc + replicate,
                 data = bactgrowth, spar = 0.5)


## these examples require some CPU power and may take a bit longer

## initial parameters
p <- c(coef(fit0), K = max(dat$value))

## avoid negative parameters
lower = c(y0 = 0, mumax = 0, K = 0)

## fit all models
fit2 <- all_growthmodels(value ~ time | strain + conc + replicate,
        data = bactgrowth, FUN=grow_logistic,
        p = p, lower = lower, ncores = 2)

results1 <- results(fit1)
results2 <- results(fit2)
plot(results1$mumax, results2$mumax, xlab="smooth splines", ylab="logistic")

## experimental: nonlinear model as part of the formula

fit3 <- all_growthmodels(
        value ~ grow_logistic(time, parms) | strain + conc + replicate,
```

```
            data = bactgrowth, p = p, lower = lower, ncores = 2)

## this allows also to fit to the 'global' data set or any subsets
fit4 <- all_growthmodels(
          value ~ grow_logistic(time, parms),
          data = bactgrowth, p = p, lower = lower, ncores = 1)
plot(fit4)

fit5 <- all_growthmodels(
          value ~ grow_logistic(time, parms) | strain + conc,
          data = bactgrowth, p = p, lower = lower, ncores = 2)
plot(fit5)
```

---

all_splines   *Fit Exponential Growth Model with Smoothing Spline*

Description

Determine maximum growth rates from log-linear part of the growth curve for a series of experiments by using smoothing splines.

Usage

```
all_splines(...)

## S3 method for class 'formula'
all_splines(formula, data = NULL, optgrid = 50, subset = NULL, ...)

## S3 method for class 'data.frame'
all_splines(
  data,
  grouping = NULL,
  time = "time",
  y = "value",
  optgrid = 50,
  ...
)
```

Arguments

...	generic parameters, including parameters passed to [smooth.spline](#), see details.
formula	model formula specifying dependent, independent and grouping variables in the form: dependent ~ independent | group1 + group2 + ...
data	data frame of observational data.

optgrid	number of steps on the x-axis used for searching the maximum of the first derivative of the spline. The default should work in most cases, as long as the data are equally spaced. A smaller number may lead to non-detectable speed-up, but has the risk that the search is trapped in a local minimum.
subset	a specification of the rows to be used: defaults to all rows.
grouping	vector of grouping variables defining subsets in the data frame.
time	character vectors with name independent variable.
y	character vector with name of dependent variable.

Details

The method was inspired by an algorithm of Kahm et al. (2010), with different settings and assumptions. In the moment, spline fitting is always done with log-transformed data, assuming exponential growth at the time point of the maximum of its first derivative.

All the hard work is done by function `smooth.spline` from package **stats**, that is highly user configurable. Normally, smoothness is automatically determined via cross-validation. This works well in many cases, whereas manual adjustment is required otherwise, e.g. by setting spar to a fixed value $[0, 1]$ that also disables cross-validation. A typical case where cross validation does not work is, if time dependent measurements are taken as pseudoreplicates from the same experimental unit.

Value

object with parameters of the fit.

References

Kahm, M., Hasenbrink, G., Lichtenberg-Frate, H., Ludwig, J., Kschischo, M. 2010. grofit: Fitting Biological Growth Curves with R. Journal of Statistical Software, 33(7), 1-21, doi:10.18637/jss.v033.i07

See Also

Other fitting functions: `all_easylinear()`, `all_growthmodels()`, `fit_easylinear()`, `fit_growthmodel()`, `fit_spline()`

Examples

```
data(bactgrowth)
L <- all_splines(value ~ time | strain + conc + replicate,
                 data = bactgrowth, spar = 0.5)

par(mfrow=c(4, 3))
plot(L)
results <- results(L)
xyplot(mumax ~ log(conc + 1)|strain, data=results)

## fit splines at lower grouping levels
```

```
L2 <- all_splines(value ~ time | conc + strain,
                       data = bactgrowth, spar = 0.5)
plot(L2)

## total data set without any grouping
L3 <- all_splines(value ~ time,
                       data = bactgrowth, spar = 0.5)
par(mfrow=c(1, 1))
plot(L3)
```

---

antibiotic   *Plate Reader Data of Bacterial Growth*

Description

Example data set from growth experiments with Pseudomonas putida on a tetracycline concentration gradient.

Format

Data frame with the following columns:

time time in hours.

variable sample code.

value bacteria concentration measured as optical density.

conc concentration of the antibiotics (Tetracycline).

repl Replicate.

Details

The sample data set shows four out of six replicates of the original experiment.

Source

Claudia Seiler, TU Dresden, Institute of Hydrobiology.

Examples

```
## plot data and determine growth rates
data(antibiotic)

dat <- subset(antibiotic, conc==0.078 & repl=="R4")
parms <- c(y0=0.01, mumax=0.2, K=0.5)
fit <- fit_growthmodel(grow_logistic, parms, dat$time, dat$value)
plot(fit); plot(fit, log="y")
```

---

bactgrowth   *Plate Reader Data of Bacterial Growth*

Description

Example data set from growth experiments with different concentrations of antibiotics.

Format

Data frame with the following columns:

strain identifier of the bacterial strain, D=donor, R=recipient, T=transconjugant.

replicate replicate of the trial.

conc concentration of the antibiotics (Tetracycline).

time time in hours.

value bacteria concentration measured as optical density.

Details

This rather 'difficult' data set was intentionally selected to make model fitting by the package more challenging.

Source

Claudia Seiler, TU Dresden, Institute of Hydrobiology.

Examples

```
## plot data and determine growth rates
data(bactgrowth)


library(lattice)
xyplot(value ~ time | strain + as.factor(conc),
      data = bactgrowth, groups = replicate)
```

---

extcoef_logistic   *Extended Set of Coefficients of a Logistic Growth Model*

Description

Estimate model-specific derived parameters of the logistic growth model

Usage

```
extcoef_logistic(object, quantile = 0.95, time = NULL, ...)
```

Arguments

object	model object fited by `fit_growthmodel`
quantile	fraction of the capacity parameter (`K`) for the quantile method
time	2-valued vector of the search interval for the independent variable (`time`). Note: this needs to be set this manually if saturation is not reached within the observation time period taken from the data.
...	reserved for future extensions

Details

This function returns the estimated parameters of a logistic growth model (`y0`, `mumax`, `K`) and a series of estimates for the time of approximate saturation. The estimates are defined as follows:

- `turnpoint`: time of turnpoint (50% saturation)
- `sat1`: time of the minimum of the 2nd derivative
- `sat2`: time of the intercept between the steepest increase (the tangent at `mumax`) and the carrying capacity `K`
- `sat3`: time when a quantile of `K` (default 0.95) is reached

This function is normally not directly called by the user. It is usually called indirectly from `coef` or `results` if `extended=TRUE`.

Value

vector that contains the fitted parameters and some derived characteristics (extended parameters) of the logistic function.

Note

The estimates for the turnpoint and the time of approximate saturation (`sat1`, `sat2`, `sat3`) may be unreliable, if saturation is not reached within the observation time period. See example below. A set of extended parameters exists currently only for the standard logistic growth model (`grow_logistic`). The code and naming of the parameters is preliminary and may change in future versions.

Examples

```
## =======================================================================
## The 'extended parameters' are usually derived
## =======================================================================

data(antibiotic)

## fit a logistic model to a single data set
dat <- subset(antibiotic, conc==0.078 & repl=="R4")

parms <- c(y0=0.01, mumax=0.2, K=0.5)
fit <- fit_growthmodel(grow_logistic, parms, dat$time, dat$value)
coef(fit, extended=TRUE)

## fit the logistic to all data sets
myData <- subset(antibiotic, repl=="R3")
parms <- c(y0=0.01, mumax=0.2, K=0.5)
all <- all_growthmodels(value ~ time | conc,
                        data = myData, FUN=grow_logistic,
                        p = parms, ncores = 2)


par(mfrow=c(3,4))
plot(all)
results(all, extended=TRUE)
## we see that the the last 3 series (10...12) do not go into saturation
## within the observation time period.

## We can try to extend the search range:
results(all[10:12], extended=TRUE, time=c(0, 5000))


## =======================================================================
## visualisation how the 'extended parameters' are derived
## =======================================================================

# Derivatives of the logistic:
#   The 1st and 2nd derivatives are internal functions of the package.
#   They are used here for the visualisation of the algorithm.

deriv1 <- function(time, y0, mumax, K) {
  ret <- (K*mumax*y0*(K - y0)*exp(mumax * time))/
    ((K + y0 * (exp(mumax * time) - 1))^2)
  unname(ret)
}

deriv2 <- function(time, y0, mumax, K) {
  ret <- -(K * mumax^2 * y0 * (K - y0) * exp(mumax * time) *
            (-K + y0 * exp(mumax * time) + y0))/
    (K + y0 * (exp(mumax * time) - 1))^3
  unname(ret)
```

```
}
## =========================================================================

data(bactgrowth)
## extract one growth experiment by name
dat <- multisplit(bactgrowth, c("strain", "conc", "replicate"))[["D:0:1"]]


## unconstraied fitting
p <- c(y0 = 0.01, mumax = 0.2, K = 0.1) # start parameters
fit1 <- fit_growthmodel(FUN = grow_logistic, p = p, dat$time, dat$value)
summary(fit1)
p <- coef(fit1, extended=TRUE)

## copy parameters to separate variables to improve readability ------------
y0 <-     p["y0"]
mumax <- p["mumax"]
K    <-   p["K"]
turnpoint <- p["turnpoint"]
sat1 <-  p["sat1"]  # 2nd derivative
sat2 <-  p["sat2"]  # intercept between steepest increase and K
sat3 <-  p["sat3"]  # a given quantile of K, default 95\%

## show saturation values in growth curve and 1st and 2nd derivatives ------
opar <- par(no.readonly=TRUE)
par(mfrow=c(3, 1), mar=c(4,4,0.2,0))
plot(fit1)

## 95% saturation
abline(h=0.95*K, col="magenta", lty="dashed")

## Intercept between steepest increase and 100% saturation
b <- deriv1(turnpoint, y0, mumax, K)
a <- K/2 - b*turnpoint
abline(a=a, b=b, col="orange", lty="dashed")
abline(h=K, col="orange", lty="dashed")
points(sat2, K, pch=16, col="orange")
points(turnpoint, K/2, pch=16, col="blue")

## sat2 is the minimum of the 2nd derivative
abline(v=c(turnpoint, sat1, sat2, sat3),
       col=c("blue", "grey", "orange", "magenta"), lty="dashed")

## plot the derivatives
with(dat, plot(time, deriv1(time, y0, mumax, K), type="l", ylab="y'"))
abline(v=c(turnpoint, sat1), col=c("blue", "grey"), lty="dashed")

with(dat, plot(time, deriv2(time, y0, mumax, K), type="l",  ylab="y''"))
abline(v=sat1, col="grey", lty="dashed")
par(opar)
```

---

fit_easylinear *Fit Exponential Growth Model with a Heuristic Linear Method*

Description

Determine maximum growth rates from the log-linear part of a growth curve using a heuristic approach similar to the "growth rates made easy"-method of Hall et al. (2013).

Usage

```
fit_easylinear(time, y, h = 5, quota = 0.95)
```

Arguments

time	vector of independent variable.
y	vector of dependent variable (concentration of organisms).
h	width of the window (number of data).
quota	part of window fits considered for the overall linear fit (relative to max. growth rate)

Details

The algorithm works as follows:

1. Fit linear regressions to all subsets of h consecutive data points. If for example $h = 5$, fit a linear regression to points $1 \ldots 5, 2 \ldots 6, 3 \ldots 7$ and so on. The method seeks the highest rate of exponential growth, so the dependent variable is of course log-transformed.

2. Find the subset with the highest slope b_{max} and include also the data points of adjacent subsets that have a slope of at least $quota \cdot b_{max}$, e.g. all data sets that have at least 95% of the maximum slope.

3. Fit a new linear model to the extended data window identified in step 2.

Value

object with parameters of the fit. The lag time is currently estimated as the intersection between the fit and the horizontal line with $y = y_0$, where y0 is the first value of the dependent variable. The intersection of the fit with the abscissa is indicated as y0_lm (lm for linear model). These identifieres and their assumptions may change in future versions.

References

Hall, BG., Acar, H, Nandipati, A and Barlow, M (2014) Growth Rates Made Easy. Mol. Biol. Evol. 31: 232-38, doi:10.1093/molbev/mst187

See Also

Other fitting functions: all_easylinear(), all_growthmodels(), all_splines(), fit_growthmodel(), fit_spline()

Examples

```
data(bactgrowth)

splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))
dat <- splitted.data[[1]]

plot(value ~ time, data=dat)
fit <- fit_easylinear(dat$time, dat$value)

plot(fit)
plot(fit, log="y")
plot(fit, which="diagnostics")

fitx <- fit_easylinear(dat$time, dat$value, h=8, quota=0.95)

plot(fit, log="y")
lines(fitx, pch="+", col="blue")

plot(fit)
lines(fitx, pch="+", col="blue")
```

---

fit_growthmodel   *Fit Nonlinear Parametric Growth Model*

Description

Determine maximum growth rates by fitting nonlinear models.

Usage

```
fit_growthmodel(
  FUN,
  p,
  time,
  y,
  lower = -Inf,
  upper = Inf,
  which = names(p),
  method = "Marq",
  transform = c("none", "log"),
  control = NULL,
  ...
)
```

Arguments

FUN	function of growth model to be fitted.
p	named vector of start parameters and initial values of the growth model.
time	vector of independent variable.
y	vector of dependent variable (concentration of organisms).
lower	lower bound of the parameter vector (optional).
upper	upper bound of the parameter vector (optional).
which	vector of parameter names that are to be fitted.
method	character vector specifying the optimization algorithm (see modFit).
transform	fit model to non-transformed or log-transformed data.
control	A list of control parameters for the optimizers. See Details.
...	additional parameters passed to the optimizer.

Details

This function calls modFit from package **FME**. Syntax of control parameters and available options may differ, depending on the optimizer used, except control=list(trace=...) that switches tracing on and off for all methods and is either TRUE, or FALSE, or an integer value like 0, 1, 2, 3, depending on the optimizer.

Value

object with parameters of the fit.

See Also

modFit about constrained fitting of models to data

Other fitting functions: all_easylinear(), all_growthmodels(), all_splines(), fit_easylinear(), fit_spline()

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))

## get one element either by index or by name
dat <- splitted.data[[1]]
dat <- splitted.data[["D:0:1"]]

p <- c(y0 = 0.01, mumax = 0.2, K = 0.1)

## unconstraied fitting
fit1 <- fit_growthmodel(FUN = grow_logistic, p = p, dat$time, dat$value)
coef(fit1)
summary(fit1)
```

```
## optional box-constraints
lower <- c(y0 = 1e-6, mumax = 0,   K = 0)
upper <- c(y0 = 0.05, mumax = 5,   K = 0.5)
fit1 <- fit_growthmodel(
  FUN = grow_logistic, p = p, dat$time, dat$value,
  lower = lower, upper = upper)

plot(fit1, log="y")
```

---

fit_spline   *Fit Exponential Growth Model with Smoothing Spline*

Description

Determine maximum growth rates from the first derivative of a smoothing spline.

Usage

```
fit_spline(time, y, optgrid = length(time), ...)
```

Arguments

time	vector of independent variable.
y	vector of dependent variable (concentration of organisms).
optgrid	number of steps on the x-axis used for the optimum search . algorithm. The default should work in most cases, as long as the data are equally spaced. A smaller number may lead to non-detectable speed-up, but has the risk that the search gets trapped in a local minimum.
...	other parameters passed to `smooth.spline`, see details.

Details

The method was inspired by an algorithm of Kahm et al. (2010), with different settings and assumptions. In the moment, spline fitting is always done with log-transformed data, assuming exponential growth at the time point of the maximum of the first derivative of the spline fit.

All the hard work is done by function `smooth.spline` from package **stats**, that is highly user configurable. Normally, smoothness is automatically determined via cross-validation. This works well in many cases, whereas manual adjustment is required otherwise, e.g. by setting spar to a fixed value $[0, 1]$ that also disables cross-validation.

Value

object with parameters of the fit

References

Kahm, M., Hasenbrink, G., Lichtenberg-Frate, H., Ludwig, J., Kschischo, M. 2010. grofit: Fitting Biological Growth Curves with R. Journal of Statistical Software, 33(7), 1-21, doi:10.18637/jss.v033.i07

See Also

Other fitting functions: all_easylinear(), all_growthmodels(), all_splines(), fit_easylinear(), fit_growthmodel()

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))

dat <- splitted.data[[2]]
time <- dat$time
y    <- dat$value

## automatic smoothing with cv
res <- fit_spline(time, y)

plot(res, log="y")
plot(res)
coef(res)

## a more difficult data set
dat <- splitted.data[[56]]
time <- dat$time
y <- dat$value

## default parameters
res <- fit_spline(time, y)
plot(res, log="y")

## small optgrid, trapped in local minimum
res <- fit_spline(time, y, optgrid=5)
plot(res, log="y")

## manually selected smoothing parameter
res <- fit_spline(time, y, spar=.5)
plot(res, log="y")
plot(res, ylim=c(0.005, 0.03))
```

---

function_growthmodel-class

Union Class of Growth Model or Function

Description

This class union comprises parametric model functions from class growthmodel and ordinary functions to describe time-dependent growth of organisms.

See Also

the constructor function [growthmodel](#) how to create instances of class growthmodel.

---

growthmodel   *Create a User-defined Parametric Growth Model*

Description

This constructor method allows to create user-defined functions that can be used as parametric models describing time-dependent growth of organisms.

Usage

```
growthmodel(x, pnames = NULL)
```

Arguments

x   a function with arguments times and parms, and returning a matrix with two
                    columns time and y.
pnames   character vector with the names of the model parameters.

Details

Package **growthrates** has a plug-in architecture allowing user-defined growth models of the following form:

```
identifier <- function(time, parms) {
  ... content of function here ...
  return(as.matrix(data.frame(time=time, y=y)))
}
```

where time is a numeric vector and parms a named, non-nested list of model parameters. The constructor function growthmodel is used to attach the names of the parameters as an optional attribute.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_logistic(), grow_richards(), ode_genlogistic(), ode_twostep()

Examples

```
test <- function(time, parms) {
  with(as.list(parms), {
    y <- (K * y0) / (y0 + (K - y0) * exp(-mumax * time)) + y_shift
    return(as.matrix(data.frame(time=time, y=y)))
 })
}

mygrowthmodel <- growthmodel(test, c("y0", "mumax", "K", "y_shift"))
```

---

growthmodel-class   *Class of Growth Model Functions*

Description

This class is used for the parametric grow_... functions of the package and can also be used for user-defined functions to describe time-dependent growth of organisms.

See Also

the constructor function growthmodel how to create instances of this class.

---

growthrates_fit-class   *S4 Classes of Package* **growthrates**

Description

growthrates_fit: top-level class representing a growthrates fit with any method.

Slots

FUN model function used.

fit results of the model fit.

obs observation data used for model fitting.

rsquared coefficient of determination.

par parameters of the fit.

ndx index values of the time points used (for easylinear_fit).

xy x and y values at the maximum of the spline.

---

grow_baranyi   *The Baranyi and Roberts Growth Model*

Description

The growth model of Baranyi and Roberts (1995) written as analytical solution of the system of differential equations.

Usage

```
grow_baranyi(time, parms)
```

Arguments

time   vector of time steps (independent variable).

parms   named parameter vector of the Baranyi growth model with:

- y0 initial value of abundance,
- mumax maximum growth rate (1/time),
- K carrying capacity (max. abundance),
- h0 parameter specifying the initial physiological state of organisms (e.g. cells) and in consequence the lag phase (h0 = max growth rate * lag phase).

Details

The version of the equation used in this package has the following form:

$$A = time + 1/mumax * log(exp(-mumax * time) + exp(-h0) - exp(-mumax * time - h0))$$

$$log(y) = log(y0) + mumax * A - log(1 + (exp(mumax * A) - 1)/exp(log(K) - log(y0)))$$

Value

vector of dependent variable (y).

References

Baranyi, J. and Roberts, T. A. (1994). A dynamic approach to predicting bacterial growth in food. International Journal of Food Microbiology, 23, 277-294.

Baranyi, J. and Roberts, T.A. (1995). Mathematics of predictive microbiology. International Journal of Food Microbiology, 26, 199-218.

See Also

Other growth models: grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_logistic(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_baranyi(time, c(y0=0.01, mumax=.5, K=0.1, h0=5))[,"y"]
plot(time, y, type="l")
plot(time, y, type="l", log="y")
```

---

grow_exponential   *Exponential Growth Model*

Description

Unlimited exponential growth model.

Usage

```
grow_exponential(time, parms)
```

Arguments

time	vector of time steps (independent variable).
parms	named parameter vector of the exponential growth model with:

- y0 initial abundance (e.g. concentration of bacterial cells).
- mumax maximum growth rate (1/time).

Details

The equation used is:
$$y = y0 * exp(mumax * time)$$

Value

vector of dependent variable (y).

See Also

Other growth models: grow_baranyi(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_logistic(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y <- grow_exponential(time, c(y0=1, mumax=0.5))[,"y"]
plot(time, y, type="l")
```

---

grow_gompertz   *Growth Model According to Gompertz*

Description

Gompertz growth model written as analytical solution of the differential equation system.

Usage

```
grow_gompertz(time, parms)
```

Arguments

time   vector of time steps (independent variable).

parms   named parameter vector of the Gompertz growth model with:

- y0 initial value of abundance,
- mumax maximum growth rate (1/time),
- K maximum abundance (carrying capacity).

Details

The equation used here is:

$$y = K * exp(log(y0/K) * exp(-mumax * time))$$

Value

vector of dependent variable (y)

Note

The naming of parameter "mumax" was done in analogy to the other growth models, but it turned out that it was not consistent with the maximum growth rate of the population. This can be considered as bug. The function will be removed or replaced in future versions of the package. Please use grow_gompertz2 instead.

References

Tsoularis, A. (2001) Analysis of Logistic Growth Models. Res. Lett. Inf. Math. Sci, (2001) 2, 23-46.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_huang(), grow_logistic(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_gompertz(time, c(y0=1, mumax=.2, K=10))[,"y"]
plot(time, y, type="l", ylim=c(0, 20))
```

---

grow_gompertz2   *Growth Model According to Gompertz*

Description

Gompertz growth model written as analytical solution of the differential equation system.

Usage

```
grow_gompertz2(time, parms)

grow_gompertz3(time, parms)
```

Arguments

time   vector of time steps (independent variable).

parms   named parameter vector of the Gompertz growth model with:

- y0 initial value of abundance,
- mumax maximum growth rate (1/time),
- K maximum abundance (carrying capacity),
- lambda time of lag phase of the 3 parameter Gompertz model .

Details

The equation used here is:

$$y = y0 * (K/y0)^{(}exp(-exp((exp(1) * mumax * (lambda - time))/log(K/y0) + 1)))$$

Functions `grow_gompert2` and `grow_gompertz3` describe sigmoidal growth with an exponentially decreasing intrinsic growth rate with or without an additional lag parameter. The formula follows the reparametrization of Zwietering et al (1990), with parameters that have a biological meaning.

Value

vector of dependent variable (y)

References

Tsoularis, A. (2001) Analysis of Logistic Growth Models. Res. Lett. Inf. Math. Sci, (2001) 2, 23-46.

Zwietering, M. H., Jongenburger, I., Rombouts, F. M., and Van't Riet, K. (1990). Modeling of the bacterial growth curve. Appl. Environ. Microbiol., 56(6), 1875-1881.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz(), grow_huang(), grow_logistic(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_gompertz(time, c(y0=1, mumax=.2, K=10))[,"y"]
plot(time, y, type="l", ylim=c(0, 12))
```

---

grow_huang   *Growth Model According to Huang*

Description

Huangs growth model written as analytical solution of the differential equations.

Usage

```
grow_huang(time, parms)
```

Arguments

time   vector of time steps (independent variable).

parms   named parameter vector of Huang's growth model with:

- y0 initial value of abundance,
- mumax maximum growth rate (1/time),
- K carrying capacity (max. total concentration of cells),
- alpha shape parameter determining the curvature,
- lambda parameter determining the lag time.

Details

The version of the equation used in this package has the following form:

$$B = time + 1/alpha * log((1 + exp(-alpha * (time - lambda)))/(1 + exp(alpha * lambda)))$$

$$log(y) = log(y0) + log(K) - log(y0 + (K - y0) * exp(-mumax * B))$$

In contrast to the original publication, all parameters related to population abundance (y, y0, K) are given as untransformed values. They are not log-transformed.

In general, using log-transformed parameters would indeed be a good idea to avoid the need of constained optimization, but tests showed that box-constrained optimization worked resonably well. Therefore, handling of optionally log-transformed parameters was removed from the package to avoid confusion. If you want to discuss this, please let me know.

Value

vector of dependent variable (y).

References

Huang, Lihan (2008) Growth kinetics of Listeria monocytogenes in broth and beef frankfurters - determination of lag phase duration and exponential growth rate under isothermal conditions. Journal of Food Science 73(5), E235 – E242. doi:10.1111/j.17503841.2008.00785.x

Huang, Lihan (2011) A new mechanistic growth model for simultaneous determination of lag phase duration and exponential growth rate and a new Belehdradek-type model for evaluating the effect of temperature on growth rate. Food Microbiology 28, 770 – 776. doi:10.1016/j.fm.2010.05.019

Huang, Lihan (2013) Introduction to USDA Integrated Pathogen Modeling Program (IPMP). Residue Chemistry and Predictive Microbiology Research Unit. USDA Agricultural Research Service.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_logistic(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_huang(time, c(y0=0.01, mumax=.1, K=0.1, alpha=1.5, lambda=3))[,"y"]
plot(time, y, type="l")
plot(time, y, type="l", log="y")
```

---

grow_logistic *Logistic Growth Model*

Description

Classical logistic growth model written as analytical solution of the differential equation.

Usage

```
grow_logistic(time, parms)
```

Arguments

time   vector of time steps (independent variable)

parms   named parameter vector of the logistic growth model with:

- y0 initial value of population measure
- mumax intrinsic growth rate (1/time)
- K carrying capacity (max. total concentration of cells)

Details

The equation used is:

$$y = (K * y0)/(y0 + (K - y0) * exp(-mumax * time))$$

Value

vector of dependent variable (y).

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_richards(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_logistic(time, c(y0=1, mumax=0.5, K=10))[,"y"]
plot(time, y, type="l")
```

---

grow_richards *Growth Model According to Richards*

Description

Richards growth model written as analytical solution of the differential equation.

Usage

```
grow_richards(time, parms)
```

Arguments

time vector of time steps (independent variable).

parms named parameter vector of the Richards growth model with:

- y0 initial value of abundance,
- mumax maximum growth rate (note different interpretation compared to exponential growth),
- K carrying capacity (max. total concentration of cells),
- beta shape parameter determining the curvature.

Details

The equation used is:

$$y = K * (1 - exp(-beta * mumax * time) * (1 - (y0/K)^{-}beta))^{(} - 1/beta)$$

The naming of parameters used here follows the convention of Tsoularis (2001), but uses mumax for growthrate and y for abundance to make them consistent to other growth functions.

Value

vector of dependent variable (y).

References

Richards, F. J. (1959) A Flexible Growth Function for Empirical Use. Journal of Experimental Botany 10 (2): 290–300.

Tsoularis, A. (2001) Analysis of Logistic Growth Models. Res. Lett. Inf. Math. Sci, (2001) 2, 23–46.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_logistic(), growthmodel, ode_genlogistic(), ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
y    <- grow_richards(time, c(y0=1, mumax=.5, K=10, beta=2))[,"y"]
plot(time, y, type="l")
y    <- grow_richards(time, c(y0=1, mumax=.5, K=10, beta=100))[,"y"]
lines(time, y, col="red")
y    <- grow_richards(time, c(y0=1, mumax=.5, K=10, beta=.2))[,"y"]
lines(time, y, col="blue")
```

---

names.growthmodel   *Get Names Attributes of Growth Models*

Description

Methods to get the parameter names of a growth model or to get or set identifiers of `multiple_fits` objects.

Usage

```
## S3 method for class 'growthmodel'
names(x)

## S4 method for signature 'multiple_fits'
names(x)

## S4 replacement method for signature 'multiple_fits'
names(x) <- value
```

Arguments

x	either a function being a parametric growth model of package **growthmodels** or an object with multiple fits.
value	a character vector of up to the same length as x, or NULL

Value

character vector of the parameter names

Methods

Method for class `growthmodel`**:** returns information about valid parameter names if a pnames attribute exists, else NULL. NULL.

Method for class `multiple_fits`**:** can be applied to objects returned by all_growthmodels, all_splines or all_easylinear respectively. This can be useful for selecting subsets, e.g. for plotting, see example below.

See Also

[multiple_fits](), [all_growthmodels](), [all_splines](), [all_easylinear]()

Examples

```
## growthmodel-method
names(grow_baranyi)

## multiple_fits-method
L <- all_splines(value ~ time | strain + conc + replicate,
        data = bactgrowth)

names(L)

## plot only the 'R' strain
par(mfrow=c(4, 6))
plot(L[grep("R:", names(L))])
```

---

ode_genlogistic   *Generalized Logistic Growth Model*

Description

Generalized logistic growth model solved as differential equation.

Usage

```
ode_genlogistic(time, y, parms, ...)

grow_genlogistic(time, parms, ...)
```

Arguments

time	vector of simulation time steps
y	named vector with initial value of the system (e.g. cell concentration)
parms	parameters of the generalized logistic growth model

- mumax maximum growth rate (1/time)
- K carrying capacity (max. abundance)
- alpha, beta, gamma parameters determining the shape of growth. Setting all values to one returns the ordinary logistic function.

...	additional parameters passed to the ode-function.

Details

The model is given as its first derivative:

$$dy/dt = mumax * y^a lpha * (1 - (y/K)^b eta)^g amma$$

that is then numerically integrated ('simulated') according to time (t).

The generalized logistic according to Tsoularis (2001) is a flexible model that covers exponential and logistic growth, Richards, Gompertz, von Bertalanffy, and some more as special cases.

The differential equation is solved numerically, where function ode_genlogistic is the differential equation, and grow_genlogistic runs a numerical simulation over time.

The default version grow_genlogistic is run directly as compiled code, whereas the R versions ode_logistic is provided for testing by the user.

Value

For ode_genlogistic: matrix containing the simulation outputs. The return value of has also class deSolve.

For grow_genlogistic: vector of dependent variable (y).

- time time of the simulation
- y abundance of organisms

References

Tsoularis, A. (2001) Analysis of Logistic Growth Models. Res. Lett. Inf. Math. Sci, (2001) 2, 23-46.

See Also

Other growth models: grow_baranyi(), grow_exponential(), grow_gompertz2(), grow_gompertz(), grow_huang(), grow_logistic(), grow_richards(), growthmodel, ode_twostep()

Examples

```
time <- seq(0, 30, length=200)
parms <- c(mumax=0.5, K=10, alpha=1, beta=1, gamma=1)
y0    <-  c(y=.1)
out   <- ode(y0, time, ode_genlogistic, parms)
plot(out)

out2 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.2, K=10, alpha=2, beta=1, gamma=1))
out3 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.2, K=10, alpha=1, beta=2, gamma=1))
out4 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.2, K=10, alpha=1, beta=1, gamma=2))
out5 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.2, K=10, alpha=.5, beta=1, gamma=1))
out6 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.2, K=10, alpha=1, beta=.5, gamma=1))
out7 <- ode(y0, time, ode_genlogistic, parms = c(mumax=0.3, K=10, alpha=1, beta=1, gamma=.5))
plot(out, out2, out3, out4, out5, out6, out7)
```

```
## growth with lag (cf. log_y)
plot(ode(y0, time, ode_genlogistic, parms = c(mumax=1, K=10, alpha=2, beta=.8, gamma=5)))
```

---

ode_twostep   *Twostep Growth Model*

Description

System of two differential equations describing bacterial growth as two-step process of activation (or adaptation) and growth.

Usage

```
ode_twostep(time, y, parms, ...)

grow_twostep(time, parms, ...)
```

Arguments

time	actual time (for the ode) resp. vector of simulation time steps.
y	named vector with state of the system (yi, ya: abundance of inactive and active organisms, e.g. concentration of inactive resp. active cells).
parms	parameters of the two-step growth model:

- yi, ya initial abundance of active and inactive organisms,
- kw activation ("wakeup") constant (1/time),
- mumax maximum growth rate (1/time),
- K carrying capacity (max. abundance).

...	placeholder for additional parameters (for user-extended versions of this function)

Details

The model is given as a system of two differential equations:

$$dy_i/dt = -kw * yi$$

$$dy_a/dt = kw * yi + mumax * (1 - (yi + ya)/K) * ya$$

that are then numerically integrated ('simulated') according to time (t). The model assumes that the population consists of active (y_a) and inactive (y_i) cells so that the observed abundance is ($y = y_i + y_a$). Adapting inactive cells change to the active state with a first order 'wakeup' rate (kw).

Function ode_twostep is the system of differential equations, whereas grow_twostep runs a numerical simulation over time.

A similar two-compartment model, but without the logistic term, was discussed by Baranyi (1998).

Value

For `ode_twostep`: matrix containing the simulation outputs. The return value of has also class `deSolve`.

For `grow_twostep`: vector of dependent variable (y):

- `time` time of the simulation
- `yi` concentration of inactive cells
- `ya` concentration of active cells
- `y` total cell concentration

References

Baranyi, J. (1998). Comparison of stochastic and deterministic concepts of bacterial lag. J. heor. Biol. 192, 403–408.

See Also

Other growth models: `grow_baranyi()`, `grow_exponential()`, `grow_gompertz2()`, `grow_gompertz()`, `grow_huang()`, `grow_logistic()`, `grow_richards()`, `growthmodel`, `ode_genlogistic()`

Other growth models: `grow_baranyi()`, `grow_exponential()`, `grow_gompertz2()`, `grow_gompertz()`, `grow_huang()`, `grow_logistic()`, `grow_richards()`, `growthmodel`, `ode_genlogistic()`

Examples

```
time <- seq(0, 30, length=200)
parms <- c(kw = 0.1,mumax=0.2, K=0.1)
y0    <-  c(yi=0.01, ya=0.0)
out   <- ode(y0, time, ode_twostep, parms)

plot(out)

o <- grow_twostep(0:100, c(yi=0.01, ya=0.0, kw = 0.1,mumax=0.2, K=0.1))
plot(o)
```

---

plot   *Plot Model Fits*

Description

Methods to plot growth model fits together with the data and, alternatively, plot diagnostics

Usage

```
## S4 method for signature 'nonlinear_fit,missing'
plot(x, y, log = "", which = c("fit", "diagnostics"), ...)

## S4 method for signature 'nonlinear_fit'
lines(x, ...)

## S4 method for signature 'easylinear_fit,missing'
plot(x, y, log = "", which = c("fit", "diagnostics"), ...)

## S4 method for signature 'smooth.spline_fit,missing'
plot(x, y, ...)

## S4 method for signature 'easylinear_fit'
lines(x, ...)

## S4 method for signature 'multiple_fits,missing'
plot(x, y, ...)
```

Arguments

x	an object returned by a model fitting function of package **growthrates**, that can contain one or multiple fits.
y	(ignored) for compatibility with the default plot method.
log	a character string which contains "y" if the y axis is to be logarithmic.
which	either "fit" (default) or "diagnostics".
...	other arguments pased to the plotting methods, see [plot.default](#) and [par](#).

Details

The plot methods detect automatically which type of plot is appropriate, depending on the class of x and can plot either one single model fit or a complete series (multiple fits). In the latter case it may be wise to redirect the graphics to an external file (e.g. a pdf) and / or to use tomething like par(mfrow=c(3,3)).

The lines-method is currently only available for single fits.

If you need more control, you can of course also write own plotting functions.

See Also

[plot.default](#), [par](#), [fit_growthmodel](#), [fit_easylinear](#), [all_growthmodels](#), [all_easylinear](#)

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))
```

```
## get table from single experiment
dat <- splitted.data[["D:0:1"]]

fit1 <- fit_spline(dat$time, dat$value)
plot(fit1, log="y")
plot(fit1)

## derive start parameters from spline fit
p <- coef(fit1)

## subset of first 10 data
first10 <-  dat[1:10, ]
fit2 <- fit_growthmodel(grow_exponential, p=p, time=first10$time, y=first10$value)

p <- c(coef(fit1), K = max(dat$value))
fit3 <- fit_growthmodel(grow_logistic, p=p, time=dat$time, y=dat$value, transform="log")

plot(fit1)
lines(fit2, col="green")
lines(fit3, col="red")


all.fits <- all_splines(value ~ time | strain + conc + replicate, data = bactgrowth)
par(mfrow=c(3,3))
plot(all.fits)

## it is also possible to plot a single fit or a subset of the fits
par(mfrow=c(1,1))
plot(all.fits[["D:0:1"]])
par(mfrow=c(2,2))
plot(all.fits[1:4])

## plot only the 'R' strain
par(mfrow=c(4, 6))
plot(all.fits[grep("R:", names(all.fits))])
```

---

predict,growthrates_fit-method
: *Model Predictions for* **growthrates** *Fits*

Description

Class-specific methods of package **growthrates** to make predictions.

Usage

```
## S4 method for signature 'growthrates_fit'
predict(object, ...)
```

```
## S4 method for signature 'smooth.spline_fit'
predict(object, newdata = NULL, ..., type = c("exponential", "spline"))

## S4 method for signature 'easylinear_fit'
predict(object, newdata = NULL, ..., type = c("exponential", "no_lag"))

## S4 method for signature 'nonlinear_fit'
predict(object, newdata, ...)

## S4 method for signature 'multiple_fits'
predict(object, ...)
```

Arguments

`object`	name of a 'growthrates' object for which prediction is desired.
`...`	additional arguments affecting the predictions produced.
`newdata`	an optional data frame with column 'time' for new time steps with which to predict.
`type`	type of predict. Can be `'exponential'` or `'spline'` for `fit_spline`, resp. code'exponential' or `'no_lag'` for `fit_easylinear`.

Details

The implementation of the predict methods is still experimental and under discussion.

See Also

[methods](#), [predict.smooth.spline](#), [predict.lm](#), [predict.nls](#)

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))

## get table from single experiment
dat <- splitted.data[[1]]

## --- linear fit ------------------------------------------------------------
fit <- fit_easylinear(dat$time, dat$value)

plot(fit)
pr <- predict(fit)
lines(pr[,1:2], col="blue", lwd=2, lty="dashed")

pr <- predict(fit, newdata=list(time=seq(2, 6, .1)), type="no_lag")
lines(pr[,1:2], col="magenta")


## --- spline fit ------------------------------------------------------------
```

```
fit1 <- fit_spline(dat$time, dat$value, spar=0.5)
coef(fit1)
summary(fit1)

plot(fit1)
pr <- predict(fit1)
lines(pr[,1:2], lwd=2, col="blue", lty="dashed")
pr <- predict(fit1, newdata=list(time=2:10), type="spline")
lines(pr[,1:2], lwd=2, col="cyan")


## --- nonlinear fit -------------------------------------------------------
dat <- splitted.data[["T:0:2"]]

p    <- c(y0 = 0.02, mumax = .5, K = 0.05, h0 = 1)
fit2 <- fit_growthmodel(grow_baranyi, p=p, time=dat$time, y=dat$value)

## prediction for given data
predict(fit2)

## prediction for new data
pr <- predict(fit2, newdata=data.frame(time=seq(0, 50, 0.1)))

plot(fit2, xlim=c(0, 50))
lines(pr[, c("time", "y")], lty="dashed", col="red")
```

---

rsquared,growthrates_fit-method

Accessor Methods of Package **growthrates**.

Description

Functions to access the results of fitted growthrate objects: summary, coef, rsquared, deviance, residuals, df.residual, obs, results.

Usage

```
## S4 method for signature 'growthrates_fit'
rsquared(object, ...)

## S4 method for signature 'growthrates_fit'
obs(object, ...)

## S4 method for signature 'growthrates_fit'
coef(object, extended = FALSE, ...)

## S4 method for signature 'easylinear_fit'
coef(object, ...)
```

```
## S4 method for signature 'smooth.spline_fit'
coef(object, extended = FALSE, ...)

## S4 method for signature 'growthrates_fit'
deviance(object, ...)

## S4 method for signature 'growthrates_fit'
summary(object, ...)

## S4 method for signature 'nonlinear_fit'
summary(object, cov = TRUE, ...)

## S4 method for signature 'growthrates_fit'
residuals(object, ...)

## S4 method for signature 'growthrates_fit'
df.residual(object, ...)

## S4 method for signature 'smooth.spline_fit'
summary(object, cov = TRUE, ...)

## S4 method for signature 'smooth.spline_fit'
df.residual(object, ...)

## S4 method for signature 'smooth.spline_fit'
deviance(object, ...)

## S4 method for signature 'multiple_fits'
coef(object, ...)

## S4 method for signature 'multiple_fits'
rsquared(object, ...)

## S4 method for signature 'multiple_fits'
deviance(object, ...)

## S4 method for signature 'multiple_fits'
results(object, ...)

## S4 method for signature 'multiple_easylinear_fits'
results(object, ...)

## S4 method for signature 'multiple_fits'
summary(object, ...)

## S4 method for signature 'multiple_fits'
residuals(object, ...)
```

Arguments

`object`	name of a 'growthrate' object.
`...`	other arguments passed to the methods.
`extended`	boolean if extended set of parameters shoild be printed
`cov`	boolean if the covariance matrix should be printed.

Examples

```
data(bactgrowth)
splitted.data <- multisplit(bactgrowth, c("strain", "conc", "replicate"))

## get table from single experiment
dat <- splitted.data[[10]]

fit1 <- fit_spline(dat$time, dat$value, spar=0.5)
coef(fit1)
summary(fit1)

## derive start parameters from spline fit
p <- c(coef(fit1), K = max(dat$value))
fit2 <- fit_growthmodel(grow_logistic, p=p, time=dat$time, y=dat$value, transform="log")
coef(fit2)
rsquared(fit2)
deviance(fit2)

summary(fit2)

plot(residuals(fit2) ~ obs(fit2)[,2])
```

`[,multiple_fits,ANY,missing-method`
Extract or Replace Parts of a 'multiple_fits' Object

Description

Operators to access parts of 'multiple_fits' objects

Usage

```
## S4 method for signature 'multiple_fits,ANY,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'multiple_fits,ANY,missing'
x[[i, j, ...]]
```

Arguments

x	object of class multiple_fits
i	numeric or character index
j	NULL (for compatibility with other uses of [or [[)
...	optional arguments passed to [
drop	If TRUE the result is coerced to the lowest possible dimension

Examples

```
data(bactgrowth)
L <- all_splines(value ~ time | strain + conc +replicate, data=bactgrowth)

coef(L[[1]])

plot(L[["R:0:2"]])

par(mfrow=c(2, 2))
plot(L[1:4])
```

Index