Package ‘nlraa’

June 12, 2020

Version 0.73

Title Nonlinear Regression for Agricultural Applications

Description Additional nonlinear regression functions using self-start (SS) algorithms. One of the functions is the Beta growth function proposed by Yin et al. (2003) <doi:10.1093/aob/mcg029>. There are several other functions with breakpoints (e.g. linear-plateau, plateau-linear, exponential-plateau, plateau-exponential, quadratic-plateau, plateau-quadratic and bilinear), a non-rectangular hyperbola and a bell-shaped curve. Twenty one (21) new self-start (SS) functions in total. This package also supports the publication 'Nonlinear regression Models and applications in agricultural research' by Archontoulis and Miguez (2015) <doi:10.2134/agronj2012.0506>, a book chapter with similar material <doi:10.2134/appliedstatistics.2016.0003.c15> and a publication by Oddi et. al. (2019) in Ecology and Evolution <doi:10.1002/ece3.5543>. The function 'nlsLMList' uses nlsLM for fitting, but it is otherwise almost identical to 'nlme::nlsList'.In addition, this release of the package provides functions for conducting simulation for 'nlme' and 'gnls' objects as well as bootstrapping. These functions are intended to work with the modeling framework of the 'nlme' package. It also provides four vignettes with extended examples.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

BugReports https://github.com/femiguez/nlraa/issues

Imports boot, knitr, MASS, Matrix, mgcv, nlme, stats

Suggests bbmle, car, emmeans, ggplot2, HydroMe, lattice, minpack.lm, NISTnls, nlstools, nls2, parallel, segmented

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Fernando Miguez [aut, cre] (<https://orcid.org/0000-0002-4627-8329>),
José Pinheiro [ctb, cph] (author of nlme::nlsList, nlme::predict.gnls, nlme::predict.nlme),
Douglas Bates [ctb, cph] (author of nlme::nlsList, nlme::predict.gnls,

nlme::predict.nlme),
R-core [ctb, cph]

Maintainer Fernando Miguez <femiguez@iastate.edu>

Repository CRAN

Date/Publication 2020-06-12 14:30:02 UTC

R topics documented:

`barley` *Barley response to nitrogen fertilizer*

Description

Data from a paper by Arild Vold on response of barley to nitrogen fertilizer

Usage

```
barley
```

Format

A data frame with 76 rows and 3 columns

year Year when the trial was conducted (1970-1988).

NF Nitrogen fertilizer (g/m^2).

yield Grain yield of barley (g/m^2).

Source

Aril Vold (1998). A generalization of ordinary yield response functions. Ecological Applications. 108:227-236.

`boot_lm` *Bootstrapping for linear models*

Description

Bootstraping for linear models

Usage

```
boot_lm(
  object,
  f = NULL,
  R = 999,
  psim = 2,
  resid.type = c("resample", "normal", "wild"),
  ...
)
```

Arguments

object	object of class lm
f	function to be applied (and bootstrapped), default coef
R	number of bootstrap samples, default 999
psim	simulation level for simulate_lm
resid.type	either “resample”, “normal” or “wild”.
...	additional arguments to be passed to function boot

Details

The residuals can either be generated by resampling with replacement (default), from a normal distribution (parameteric) or by changing their signs (wild). This last one is called “wild bootstrap”.

Examples

```
require(car)
data(barley, package = "nlraa")
## Fit a linear model (quadratic)
fit.lm <- lm(yield ~ NF + I(NF^2), data = barley)

## Bootstrap coefficients by default
fit.lm.bt <- boot_lm(fit.lm)
## Compute confidence intervals
confint(fit.lm.bt, type = "perc")
## Visualize
hist(fit.lm.bt, 1, ci = "perc", main = "Intercept")
hist(fit.lm.bt, 2, ci = "perc", main = "NF term")
hist(fit.lm.bt, 3, ci = "perc", main = "I(NF^2) term")
```

---

boot_lme *Bootstraping for linear mixed models*

Description

Bootstraping tools for linear mixed-models using a consistent interface

Usage

```
boot_lme(object, f = NULL, R = 999, psim = 1, cores = 1L, ...)
```

Arguments

object	object of class lme or gnls
f	function to be applied (and bootstrapped), default coef (gls) or fixef (lme)
R	number of bootstrap samples, default 999
psim	simulation level for vector of fixed parameters either for simulate_gls or simulate_lme
cores	number of cores to use for parallel computation
...	additional arguments to be passed to function boot

Details

This function is inspired by Boot, which does not seem to work with 'gls' or 'lme' objects. This function makes multiple copies of the original data, so it can be very hungry in terms of memory use, but I do not believe this to be a big problem given the models we typically fit.

Examples

```
require(nlme)
require(car)
data(Orange)

fm1 <- lme(circumference ~ age, random = ~ 1 | Tree, data = Orange)
fm1.bt <- boot_lme(fm1, R = 50)

hist(fm1.bt)
```

---

boot_nlme *Bootstraping for generalized nonlinear models and nonlinear mixed models*

Description

Bootstraping tools for nonlinear models using a consistent interface

Usage

```
boot_nlme(object, f = NULL, R = 999, psim = 1, cores = 1L, ...)
```

Arguments

object	object of class nlme or gnls
f	function to be applied (and bootstrapped), default coef (gnls) or fixef (nlme)
R	number of bootstrap samples, default 999
psim	simulation level for vector of fixed parameters either for simulate_gnls or simulate_nlme_one
cores	number of cores to use for parallel computation
...	additional arguments to be passed to function boot

Details

This function is inspired by Boot, which does not seem to work with 'gnls' or 'nlme' objects. This function makes multiple copies of the original data, so it can be very hungry in terms of memory use, but I do not believe this to be a big problem given the models we typically fit.

Examples

```
require(car)
require(nlme)
data(barley, package = "nlraa")
barley2 <- subset(barley, year < 1974)
fit.lp.gnls2 <- gnls(yield ~ SSlinp(NF, a, b, xs), data = barley2)
barley2$year.f <- as.factor(barley2$year)
cfs <- coef(fit.lp.gnls2)
fit.lp.gnls3 <- update(fit.lp.gnls2,
                       params = list(a + b + xs ~ year.f),
                       start = c(cfs[1], 0, 0, 0,
                                 cfs[2], 0, 0, 0,
                                 cfs[3], 0, 0, 0))
## This will take a few seconds
fit.lp.gnls.Bt3 <- boot_nlme(fit.lp.gnls3, R = 300)
confint(fit.lp.gnls.Bt3, type = "perc")
```

---

lfmc *Live fuel moisture content*

Description

Live fuel moisture content

Usage

```
lfmc
```

Format

A data frame with 247 rows and 5 variables:

leaf.type -factor- Species for which data was recorded ("Grass E", "Grass W", "M. spinosum", "S. bracteolactus")

time -integer- time in days 1-80

plot -factor- plot with levels 1-6 (discrete)

site -factor- either P ("East") or SR ("West")

lfmc -numeric- Live fuel moisture content (percent)

group grouping for regression

Details

A dataset containing the leaf.type, time, plot, site and lfmc (live fuel mass concentration)

Source

`https://doi.org/10.1002/ece3.5543`

`nlraa.env` *Environment to store options and data for nlraa*

Description

Environment which stores indecies and data for bootstraping mostly

Usage

```
nlraa.env
```

Format

An object of class `environment` of length 2.

Details

Create an nlraa environment for bootstrapping

nlsLMList	*Create a list of nls objects with the option of using nlsLM in addition to nls*

Description

This function is a copy of 'nlsList' from the 'nlme' package modified to use the 'nlsLM' function in addition to (optinally) 'nls'. By changing the algorithm argument it is possible to use 'nls' as well

Usage

```
nlsLMList(
  model,
  data,
  start,
  control,
  level,
  subset,
  na.action = na.fail,
  algorithm = c("LM", "default", "port", "plinear"),
  pool = TRUE,
  warn.nls = NA
)
```

Arguments

`model`	either a nonlinear model formula, with the response on the left of a ~ operator and an expression involving parameters, covariates, and a grouping factor separated by the | operator on the right, or a selfStart function.
`data`	a data frame
`start`	list with starting values
`control`	control list, see `nls`
`level`	an optional integer specifying the level of grouping to be used when multiple nested levels of grouping are present.
`subset`	subset of rows to use
`na.action`	a function that indicates what should happen when the data contain NAs. The default action (na.fail) causes nlsList to print an error message and terminate if there are any incomplete observations.
`algorithm`	choice of algorithm. Default is 'LM' which uses 'nlsLM' from the **minpack.lm** package. Other options are: "default", "port" and "plinear" (nls).
`pool`	an optional logical value that is preserved as an attribute of the returned value. This will be used as the default for pool in calculations of standard deviations or standard errors for summaries.
`warn.nls`	logical indicating if nls errors (all of which are caught by tryCatch) should be signalled as a "summarizing" warning.

Details

See function `nlsList` and `nlsLM`. This function is a copy of nlsList but with minor changes to use LM instead as the default algorithm. The authors of the original function are Pinheiro and Bates.

Author(s)

Jose C. Pinheiro and Douglas M. Bates <`bates@stat.wisc.edu`> wrote the original `nlsList`. Fernando E. Miguez made minor changes to use `nlsLM` in addition to (optionally) `nls`. R-Core maintains copyright after 2006.

`nlsLMList.formula` *Formula method for nls 'LM' list method*

Description

formula method for nlsLMList

Usage

```
## S3 method for class 'formula'
nlsLMList(
  model,
  data,
  start = NULL,
  control,
  level,
  subset,
  na.action = na.fail,
  algorithm = c("LM", "default", "port", "plinear"),
  pool = TRUE,
  warn.nls = NA
)
```

Arguments

`model`	see `nlsList`
`data`	see `nlsList`
`start`	see `nlsList`
`control`	see `nls`
`level`	see `nlsList`
`subset`	see `nlsList`
`na.action`	see `nlsList`
`algorithm`	choice of algorithm default is 'LM' which uses 'nlsLM' from the minpack.lm package.
`pool`	see `nlsList`
`warn.nls`	see `nlsList`

simulate_gls *Simulate fitted values from an object of class* gls

Description

Simulate values from an object of class gls. Unequal variances, as modeled using the 'weights' option are supported, and there is experimental code for dealing with the 'correlation' structure.

Usage

```
simulate_gls(object, psim = 1, na.action = na.fail, naPattern = NULL, ...)
```

Arguments

object	object of class gls
psim	parameter simulation level, 0: for fitted values, 1: for simulation from fixed parameters (assuming a fixed vcov matrix), 2: for simulation considering the uncertainty in the residual standard error (sigma), this returns data which will appear similar to the observed values
na.action	default 'na.fail'. See predict.gls
naPattern	missing value pattern. See predict.gls
...	additional arguments (it is possible to supply a newdata this way)

Details

This function is based on predict.gls function

It uses function mvrnorm to generate new values for the coefficients of the model using the Variance-Covariance matrix vcov. This variance-covariance matrix refers to the one for the parameters 'beta', not the one for the residuals.

Value

It returns a vector with simulated values with length equal to the number of rows in the original data

See Also

predict.gls

Examples

```
require(nlme)
data(Orange)

fit.gls <- gls(circumference ~ age, data = Orange,
               weights = varPower())
```

```
## Visualize covariance matrix
fit.gls.vc <- var_cov(fit.gls)
image(log(fit.gls.vc[,ncol(fit.gls.vc):1]))

sim <- simulate_gls(fit.gls)
```

`simulate_gnls` *Simulate fitted values from an object of class* `gnls`

Description

Simulate values from an object of class gnls. Unequal variances, as modeled using the 'weights' option are supported, and there is experimental code for dealing with the 'correlation' structure.

Usage

```
simulate_gnls(object, psim = 1, na.action = na.fail, naPattern = NULL, ...)
```

Arguments

`object`	object of class `gnls`
`psim`	parameter simulation level, 0: for fitted values, 1: for simulation from fixed parameters (assuming a fixed vcov matrix), 2: for simulation considering the uncertainty in the residual standard error (sigma), this returns data which will appear similar to the observed values
`na.action`	default 'na.fail'. See `predict.gnls`
`naPattern`	missing value pattern. See `predict.gnls`
`...`	additional arguments (it is possible to supply a newdata this way)

Details

This function is based on `predict.gnls` function

It uses function `mvrnorm` to generate new values for the coefficients of the model using the Variance-Covariance matrix `vcov`. This variance-covariance matrix refers to the one for the parameters 'beta', not the one for the residuals.

Value

It returns a vector with simulated values with length equal to the number of rows in the original data

See Also

`predict.gnls`

Examples

```
require(nlme)
data(barley, package = "nlraa")

fit.gnls <- gnls(yield ~ SSlinp(NF, a, b, xs), data = barley)

sim <- simulate_gnls(fit.gnls)
```

---

simulate_lm *Simulate responses from a linear model* lm

Description

The function simulate does not consider the uncertainty in the estimation of the model parameters. This function will attempt to do this

Usage

```
simulate_lm(
  object,
  psim = 1,
  nsim = 1,
  resid.type = c("resample", "normal", "wild"),
  value = c("matrix", "data.frame"),
  ...
)
```

Arguments

object	object of class lm
psim	parameter simulation level (an integer, 0, 1, 3, 4).
nsim	number of simulations to perform
resid.type	type of residual to include (resample, normal or wild)
value	either 'matrix' or 'data.frame'
...	additional arguments (none used at the moment)

Details

Simulate responses from a linear model lm

These are the options that control the parameter simulation level

psim = 0 returns the fitted values

psim = 1 simulates a beta vector (mean response)

psim = 2 simulates a beta vector and adds resampled residuals (similar to observed data)

psim = 3 simulates a beta vector, considers uncertainty in the variance covariance matrix of beta and adds residuals (prediction)

psim = 4 only adds residuals according to resid.type (similar to simulate.lm)

The residual type (resid.type) controls how the residuals are generated. They are either resampled, simulated from a normal distribution or 'wild' where the Rademacher distribution is used (https://en.wikipedia.org/wiki/Rademacher_distribution). Resampled and normal both assume iid, but 'normal' makes the stronger assumption of normality. 'wild' does not assume constant variance, but it assumes symmetry.

Value

matrix or data.frame with responses

References

See "Inference Based on the Wild Bootstrap" James G. MacKinnon https://www.math.kth.se/matstat/gru/sf2930/papers/wild.bootstrap.pdf "Bootstrap in Nonstationary Autoregression" Zuzana Praskova https://dml.cz/bitstream/handle/10338.dmlcz/135473/Kybernetika_38-2002-4_1.pdf "Jackknife, Bootstrap and other Resampling Methods in Regression Analysis" C. F. J. Wu. The Annals of Statistics. 1986. Vol 14. 1261-1295.

Examples

```
require(ggplot2)
data(Orange)
fit <- lm(circumference ~ age, data = Orange)
sims <- simulate_lm(fit, nsim = 100, value = "data.frame")

ggplot(data = sims) +
  geom_line(aes(x = age, y = sim.y, group = ii),
            color = "gray", alpha = 0.5) +
  geom_point(aes(x = age, y = circumference))
```

---

simulate_lme *Simulate values from an object of class* lme

Description

Simulate values from an object of class lme. Unequal variances, as modeled using the 'weights' option are supported, but the 'correlation' is not taken into account when sampling residuals (at the moment).

Usage

```
simulate_lme(
  object,
  nsim = 1,
  psim = 1,
  value = c("matrix", "data.frame"),
  ...
)
```

Arguments

object	object of class lme
nsim	number of samples, default 1
psim	parameter simulation level, 0: for fitted values, 1: for simulation from fixed parameters (assuming a fixed vcov matrix), 2: for simulation considering the uncertainty in the residual standard error (sigma), this returns data which will appear similar to the observed values. 3: in addition samples a new set of random effects.
value	whether to return a matrix (default) or an augmented data frame
...	additional arguments (it is possible to supply a newdata this way)

Details

This function is based on `predict.lme` function

It uses function `mvrnorm` to generate new values for the coefficients of the model using the Variance-Covariance matrix `vcov`. This variance-covariance matrix refers to the one for the parameters 'beta', not the one for the residuals.

Value

It returns a vector with simulated values with length equal to the number of rows in the original data

Note

I find the simulate.merMod in the lme4 pacakge confusing. There is use.u and several versions of re.form. From the documentation it seems that if use.u = TRUE, then the current values of the random effects are used. This would mean that it is equivalent to psim = 2 in this function. Then use.u = FALSE, would be equivalent to psim = 3. re.form allows for specifying the formula of the random effects.

See Also

`predict.lme`

Examples

```
require(nlme)
data(Orange)

fm1 <- lme(circumference ~ age, random = ~ 1 | Tree, data = Orange)

sims <- simulate_lme(fm1, nsim = 10)
```

`simulate_nlme` *Simulate samples from a nonlinear mixed model from fixed effects*

Description

Simulate multiple samples from a nonlinear model

Usage

```
simulate_nlme(
  object,
  nsim = 1,
  psim = 1,
  value = c("matrix", "data.frame"),
  ...
)
```

Arguments

`object`	object of class `gnls` or `nlme`
`nsim`	number of samples, default 1
`psim`	simulation level for vector of fixed parameters for `simulate_nlme_one`
`value`	whether to return a matrix (default) or an augmented data frame
`...`	additional arguments to be passed to either `simulate_gnls` or `simulate_nlme_one`

Details

The details can be found in either `simulate_gnls` or `simulate_nlme_one`. This function is very simple and it only sets up a matrix and a loop in order to simulate several instances of model outputs.

Value

It returns a matrix with simulated values from the original object with number of rows equal to the number of rows of `fitted` and number of columns equal to the number of simulated samples ('nsim'). In the case of 'data.frame' it returns an augmented data.frame, which can potentially be a very large object, but which makes furhter plotting more convenient.

Examples

```
require(car)
require(nlme)
data(barley)
barley2 <- subset(barley, year < 1974)
fit.lp.gnls2 <- gnls(yield ~ SSlinp(NF, a, b, xs), data = barley2)
barley2$year.f <- as.factor(barley2$year)
cfs <- coef(fit.lp.gnls2)
fit.lp.gnls3 <- update(fit.lp.gnls2,
                       params = list(a + b + xs ~ year.f),
                       start = c(cfs[1], 0, 0, 0,
                                 cfs[2], 0, 0, 0,
                                 cfs[3], 0, 0, 0))

sims <- simulate_nlme(fit.lp.gnls3, nsim = 3)
```

---

simulate_nlme_one *Simulate fitted values from an object of class* nlme

---

Description

This function is based on predict.nlme function

Usage

```
simulate_nlme_one(
  object,
  psim = 1,
  level = Q,
  asList = FALSE,
  na.action = na.fail,
  naPattern = NULL,
  ...
)
```

Arguments

object	object of class nlme
psim	parameter simulation level, 0: for fitted values, 1: for simulation from fixed parameters (assuming a fixed vcov matrix), 2: for simulation considering the residual error (sigma), this returns data which will appear similar to the observed values
level	level at which simulations are performed. See predict.nlme. An important difference is that for this function multiple levels are not allowed.

`asList`	optional logical value. See `predict.nlme`
`na.action`	missing value action. See `predict.nlme`
`naPattern`	missing value pattern. See `predict.nlme`
`...`	additional arguments to be passed (possible to pass newdata this way)

Details

It uses function `mvrnorm` to generate new values for the coefficients of the model using the Variance-Covariance matrix `vcov`

Value

This function should always return a vector with the same dimensions as the original data

---

`sm` *Sorghum and Maize growth in Greece*

Description

Sorghum and Maize growth in Greece

Usage

```
sm
```

Format

A data frame with 235 rows and 5 columns

DOY -integer- Day of the year 141-303

Block -integer- Block in the experimental design 1-4

Input -integer- Input level 1 (Low) or 2 (High)

Crop -factor- either F (Fiber Sorghum), M (Maize), S (Sweet Sorghum)

Yield -numeric- Biomass yield in Mg/ha

Details

A dataset containing growth data for sorghum and maize in Greece.

Danalatos, N.G., S.V. Archontoulis, and K. Tsiboukas. 2009. Comparative analysis of sorghum versus corn growing under optimum and under water/nitrogen limited conditions in central Greece. In: From research to industry and markets: Proceedings of the 17th European Biomass Conference, Hamburg, Germany. 29 June–3 July 2009. ETA–Renewable Energies, Florence, Italy. p. 538–544.

Source

See above reference. (Currently available on ResearchGate).

SSbell — *self start for a bell-shaped curve*

Description

Self starter for a type of bell-shaped curve

Usage

```
bell(x, ymax, a, b, xc)

SSbell(x, ymax, a, b, xc)
```

Arguments

x	input vector
ymax	maximum value of y
a	parameter controlling the spread (associated with a quadratic term)
b	parameter controlling the spread (associated with a cubic term)
xc	centering parameter

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506). One example application is Hammer et al. (2009) (doi:10.2135/cropsci2008.03.0152).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

bell: vector of the same length as x using a bell-shaped curve

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:20
y <- bell(x, 8, -0.0314, 0.000317, 13) + rnorm(length(x), 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSbell(x, ymax, a, b, xc), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSbg4rp *self start for the reparameterized Beta growth function with four parameters*

Description

Self starter for Beta Growth function with parameters w.max, lt.m, ldt

Usage

```
bg4rp(time, w.max, lt.e, ldtm, ldtb)

SSbg4rp(time, w.max, lt.e, ldtm, ldtb)
```

Arguments

`time`	input vector (x) which is normally 'time', the smallest value should be close to zero.
`w.max`	value of weight or mass at its peak
`lt.e`	log of the time at which the maximum weight or mass has been reached.
`ldtm`	log of the difference between time at which the weight or mass reaches its peak and half its peak.
`ldtb`	log of the difference between time at which the weight or mass reaches its peak and when it starts growing

Details

For details see the publication by Yin et al. (2003) "A Flexible Sigmoid Function of Determinate Growth". This is a reparameterization of the beta growth function (4 parameters) with guaranteed constraints, so it is expected to behave numerically better than `SSbgf4`.

Reparameterizing the four parameter beta growth

- ldtm = log(t.e - t.m)
- ldtb = log(t.m - t.b)
- t.e = exp(lt.e)
- t.m = exp(lt.e) - exp(ldtm)
- t.b = (exp(lt.e) - exp(ldtm)) - exp(ldtb)

The form of the equation is:

$$w.max * (1 + (exp(lt.e) - time)/exp(ldtm)) * ((time - (exp(lt.e) - exp(ldtb)))/exp(ldtb))^(exp(ldtb)/exp(ldtm))$$

This is a reparameterized version of the Beta-Growth function in which the parameters are unconstrained, but they are expressed in the log-scale.

Value

bg4rp: vector of the same length as x (time) using the beta growth function with four parameters

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:100
y <- bg4rp(x, 20, log(70), log(30), log(20)) + rnorm(100, 0, 1)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSbg4rp(x, w.max, lt.e, ldtm, ldtb), data = dat)
## We are able to recover the original values
exp(coef(fit)[2:4])
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSbgf *self start for Beta Growth Function*

Description

Self starter for Beta Growth function with parameters w.max, t.m and t.e

Usage

```
bgf(time, w.max, t.e, t.m)

SSbgf(time, w.max, t.e, t.m)

bgf2(time, w.max, w.b, t.e, t.m, t.b)
```

Arguments

time	input vector (x) which is normally 'time', the smallest value should be close to zero.
w.max	value of weight or mass at its peak
t.e	time at which the weight or mass reaches its peak.
t.m	time at which half of the maximum weight or mass has been reached.
w.b	weight or biomass at initial time
t.b	initial time offset

Details

For details see the publication by Yin et al. (2003) "A Flexible Sigmoid Function of Determinate Growth".

The form of the equation is:

$$w.max * (1 + (t.e - time)/(t.e - t.m)) * (time/t.e)^(t.e/(t.e - t.m))$$

. Given this function weight is expected to decay and reach zero again at $2 * t.e - t.m$.

Value

bgf: vector of the same length as x (time) using the beta growth function

bgf2: a numeric vector of the same length as x (time) containing parameter estimates for equation specified

Examples

```
## See extended example in vignette 'nlraa-AgronJ-paper'
x <- seq(0, 17, by = 0.25)
y <- bgf(x, 5, 15, 7)
plot(x, y)
```

SSbgf4 — *self start for Beta growth function with four parameters*

Description

Self starter for Beta Growth function with parameters w.max, t.e, t.m and t.b

Usage

```
bgf4(time, w.max, t.e, t.m, t.b)

SSbgf4(time, w.max, t.e, t.m, t.b)
```

Arguments

`time`	input vector (x) which is normally 'time'.
`w.max`	value of weight or mass at its peak.
`t.e`	time at which the weight or mass reaches its peak.
`t.m`	time at which half of the maximum weight or mass has been reached.
`t.b`	time at which growth starts.

Details

For details see the publication by Yin et al. (2003) "A Flexible Sigmoid Function of Determinate Growth". This is a difficult function to fit because the linear constrains are not explicitly introduced in the optimization process. See function SSbgrp for a reparameterized version.

This is equation 11 (pg. 368) in the publication by Yin, but with correction (https://doi.org/10.1093/aob/mcg091) and with 'w.b' equal to zero.

Value

a numeric vector of the same length as x (time) containing parameter estimates for equation specified

bgf4: vector of the same length as x (time) using the beta growth function with four parameters

Examples

```
data(sm)
## Let's just pick one crop
sm2 <- subset(sm, Crop == "M")
fit <- nls(Yield ~ SSbgf4(DOY, w.max, t.e, t.m, t.b), data = sm2)
plot(Yield ~ DOY, data = sm2)
lines(sm2$DOY,fitted(fit))
## For this particular problem it could be better to 'fix' t.b and w.b
fit0 <- nls(Yield ~ bgf2(DOY, w.max, w.b = 0, t.e, t.m, t.b = 141),
           data = sm2, start = list(w.max = 16, t.e= 240, t.m = 200))

x <- seq(0, 17, by = 0.25)
y <- bgf4(x, 20, 15, 10, 2)
plot(x, y)
```

SSbgrp *self start for the reparameterized Beta growth function*

Description

Self starter for Beta Growth function with parameters w.max, lt.m and ldt

Usage

```
bgrp(time, w.max, lt.e, ldt)

SSbgrp(time, w.max, lt.e, ldt)
```

Arguments

`time` — input vector (x) which is normally 'time', the smallest value should be close to zero.

`w.max` — value of weight or mass at its peak

`lt.e` — log of the time at which the maximum weight or mass has been reached.

`ldt` — log of the difference between time at which the weight or mass reaches its peak and half its peak ($log(t.e - t.m)$).

Details

For details see the publication by Yin et al. (2003) "A Flexible Sigmoid Function of Determinate Growth". This is a reparameterization of the beta growth function with guaranteed constraints, so it is expected to behave numerically better than SSbgf.

The form of the equation is:

$$w.max * (1 + (exp(lt.e) - time)/exp(ldt)) * (time/exp(lt.e))^(exp(lt.e)/exp(ldt))$$

. Given this function weight is expected to decay and reach zero again at $2 * ldt$. This is a reparameterized version of the Beta-Growth function in which the parameters are unconstrained, but they are expressed in the log-scale.

Value

bgrp: vector of the same length as x (time) using the beta growth function (reparameterized).

Note

In a few tests it seems that zero values of 'time' can cause the error message 'NA/NaN/Inf in foreign function call (arg 1)', so it might be better to remove them before running this function.

Examples

```
require(ggplot2)
x <- 1:30
y <- bgrp(x, 20, log(25), log(5)) + rnorm(30, 0, 1)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSbgrp(x, w.max, lt.e, ldt), data = dat)
## We are able to recover the original values
exp(coef(fit)[2:3])
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSblin *self start for a bilinear Function*

Description

Self starter for a bilinear function with parameters a (intercept), b (first slope), xs (break-point), c (second slope)

Usage

```
blin(x, a, b, xs, c)

SSblin(x, a, b, xs, c)
```

Arguments

x	input vector
a	the intercept
b	the first-phase slope
xs	break-point of transition between first-phase linear and second-phase linear
c	the second-phase slope

Details

This is a special case with just two parts but a more general approach is to consider a segmented function with several breakpoints and linear segments. Splines would be even more general. Also this model assumes that there is a break-point that needs to be estimated.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

blin: vector of the same length as x using the bilinear function

See Also

package **segmented**.

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:30
y <- blin(x, 0, 0.75, 15, 1.75) + rnorm(30, 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSblin(x, a, b, xs, c), data = dat)
## plot
```

```
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Minimal example
## This is probably about the smallest dataset you
## should use with this function
dat2 <- data.frame(x = 1:5, y = c(1.1, 1.9, 3.1, 2, 0.9))
fit2 <- nls(y ~ SSblin(x, a, b, xs, c), data = dat2)
ggplot(data = dat2, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit2)))
```

SSdlf *self start for Declining Logistic Function*

Description

Self starter for declining logistic function with parameters asym, a2, xmid and scal

Usage

```
dlf(time, asym, a2, xmid, scal)

SSdlf(time, asym, a2, xmid, scal)
```

Arguments

`time`	input vector (x) which is normally 'time', the smalles value should be close to zero.
`asym`	value of weight or mass at its peak (maximum)
`a2`	value of weight or mass at its trough (minimum)
`xmid`	time at which half of the maximum weight or mass has bean reached.
`scal`	scale parameter which controls the spread also interpreted in terms of time to go from xmid to approx. 0.63 asym

Details

Response function:

$$y = (asym - a2)/(1 + exp((xmid - time)/scal))) + a2$$

.

- asym: upper asymptote
- xmid: time when y is midway between w and a

- scal: controls the spread
- a2: lower asymptote

The four parameter logistic SSfpl is essentially equivalent to this function, but here the interpretation of the parameters is different and this is the form used in Oddi et. al. (2019) (see vignette).

Value

a numeric vector of the same length as x (time) containing parameter estimates for equation specified

dlf: vector of the same length as x (time) using the declining logistic function

Examples

```
## Extended example in the vignette 'nlraa-Oddi-LFMC'
x <- seq(0, 17, by = 0.25)
y <- dlf(x, 2, 10, 8, 1)
plot(x, y, type = "l")
```

SSexpf *self start for an exponential function*

Description

Self starter for a simple exponential function

Usage

```
expf(x, a, c)

SSexpf(x, a, c)
```

Arguments

x	input vector (x)
a	represents the value at x = 0
c	represents the exponential rate

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

expf: vector of the same length as x using the profd function

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:15
y <- expf(x, 10, -0.3) + rnorm(15, 0, 0.2)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSexpf(x, a, c), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSexpfp — *self start for an exponential-plateau function*

Description

Self starter for an exponential-plateau function

Usage

```
expfp(x, a, c, xs)

SSexpfp(x, a, c, xs)
```

Arguments

x	input vector (x)
a	represents the value at x = 0
c	represents the exponential rate
xs	represents the breakpoint at which the plateau starts

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

expfp: vector of the same length as x using the expfp function

Examples

```
require(ggplot2)
set.seed(12345)
x <- 1:30
y <- expfp(x, 10, 0.1, 15) + rnorm(30, 0, 1.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSexpfp(x, a, c, xs), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

`SSexplin` *self start for the exponential-linear growth equation*

Description

Self starter for an exponential-linear growth equation

Usage

```
explin(t, cm, rm, tb)

SSexplin(t, cm, rm, tb)
```

Arguments

`t`	input vector (time)
`cm`	parameter related to the maximum growth during the linear phase
`rm`	parameter related to the maximum growth during the exponential phase
`tb`	time at which switch happens

Details

J. GOUDRIAAN, J. L. MONTEITH, A Mathematical Function for Crop Growth Based on Light Interception and Leaf Area Expansion, Annals of Botany, Volume 66, Issue 6, December 1990, Pages 695–701, https://doi.org/10.1093/oxfordjournals.aob.a088084

The equation is:

$$(cm/rm) * log(1 + exp(rm * (t - tb)))$$

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

explin: vector of the same length as x using a explin function

Examples

```
require(ggplot2)
set.seed(12345)
x <- seq(1,100, by = 5)
y <- explin(x, 20, 0.14, 30) + rnorm(length(x), 0, 5)
y <- abs(y)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSexplin(x, cm, rm, tb), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

`SShill` *self start for Hill Function*

Description

Self starter for Hill function with parameters Ka, n and a

Usage

```
hill1(x, Ka)

SShill1(x, Ka)

hill2(x, Ka, n)

SShill2(x, Ka, n)

hill3(x, Ka, n, a)

SShill3(x, Ka, n, a)
```

Arguments

`x`	input vector (x). Concentration of substrate in the original Hill model.
`Ka`	parameter representing the concentration at which half of maximum y is attained
`n`	parameter which controls the curvature
`a`	parameter which controls the maximum value of the response (asymptote)

Details

For details see https://en.wikipedia.org/wiki/Hill_equation_(biochemistry)

The form of the equations are:
hill1:
$$1/(1 + (Ka/x))$$
.
hill2:
$$1/(1 + (Ka/x)^n)$$
.
hill3:
$$a/(1 + (Ka/x)^n)$$
.

Value

hill1: vector of the same length as x (time) using the Hill 1 function

hill2: vector of the same length as x (time) using the Hill 2 function

hill3: vector of the same length as x (time) using the Hill 3 function

Note

Zero values are not allowed.

Examples

```
require(ggplot2)
## Example for hill1
set.seed(1234)
x <- 1:20
y <- hill1(x, 10) + rnorm(20, sd = 0.03)
dat1 <- data.frame(x = x, y = y)
fit1 <- nls(y ~ SShill1(x, Ka), data = dat1)

## Example for hill2
y <- hill2(x, 10, 1.5) + rnorm(20, sd = 0.03)
dat2 <- data.frame(x = x, y = y)
fit2 <- nls(y ~ SShill2(x, Ka, n), data = dat2)

## Example for hill3
y <- hill3(x, 10, 1.5, 5) + rnorm(20, sd = 0.03)
dat3 <- data.frame(x = x, y = y)
fit3 <- nls(y ~ SShill3(x, Ka, n, a), data = dat3)

ggplot(data = dat3, aes(x, y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit3)))
```

SSlinp *self start for linear-plateau function*

Description

Self starter for linear-plateau function with parameters a (intercept), b (slope), xs (break-point)

Usage

```
linp(x, a, b, xs)

SSlinp(x, a, b, xs)
```

Arguments

x	input vector
a	the intercept
b	the slope
xs	break-point of transition between linear and plateau

Details

This function is linear when $x < xs : (a + b * x)$ and flat $(asymptote = a + b * xs)$ when $x >= xs$.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

linp: vector of the same length as x using the linear-plateau function

See Also

package **segmented**.

Examples

```
require(ggplot2)
set.seed(123)
x <- 1:30
y <- linp(x, 0, 1, 20) + rnorm(30, 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSlinp(x, a, b, xs), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Confidence intervals
confint(fit)
```

SSlogis5 *self start for five-parameter logistic function*

Description

Self starter for a five-parameter logistic function.

Usage

```
logis5(x, asym1, asym2, xmid, iscal, theta)

SSlogis5(x, asym1, asym2, xmid, iscal, theta)
```

Arguments

x	input vector (x) which is normally light intensity (PPFD, Photosynthetic Photon Flux Density).
asym1	asymptotic value for low values of x
asym2	asymptotic value for high values of x
xmid	value of x at which y = (asym1 + asym2)/2
iscal	steepness of transition from asym1 to asym2 (inverse of the scale)
theta	asymmetry parameter, if it is equal to 1, this is the four parameter logistic

Details

This is known as the Richards' function or the log-logistic and it is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x (time) containing parameter estimates for equation specified

logis5: vector of the same length as x (time) using the 5-parameter logistic

Examples

```
require(ggplot2)
set.seed(1234)
x <- seq(0, 2000, 100)
y <- logis5(x, 35, 10, 800, 5, 2) + rnorm(length(x), 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSlogis5(x, asym1, asym2, xmid, scal, theta), data = dat)
## plot
```

```
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))

x <- seq(0, 2000)
y <- logis5(x, 30, 10, 800, 5, 2)
plot(x, y)
```

SSnrh *self start for non-rectangular hyperbola (photosynthesis)*

Description

Self starter for Non-rectangular Hyperbola with parameters: asymptote, quantum efficiency, curvature and dark respiration

Usage

```
nrh(x, asym, phi, theta, rd)

SSnrh(x, asym, phi, theta, rd)
```

Arguments

x	input vector (x) which is normally light intensity (PPFD, Photosynthetic Photon Flux Density).
asym	asymptotic value for photosynthesis
phi	quantum efficiency (mol CO2 per mol of photons) or initial slope of the light response
theta	curvature parameter for smooth transition between limitations
rd	dark respiration or value of CO2 uptake at zero light levels

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x (time) containing parameter estimates for equation specified

nrh: vector of the same length as x (time) using the non-rectangular hyperbola

Examples

```
require(ggplot2)
set.seed(1234)
x <- seq(0, 2000, 100)
y <- nrh(x, 35, 0.04, 0.83, 2) + rnorm(length(x), 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSnrh(x, asym, phi, theta, rd), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))

x <- seq(0, 2000)
y <- nrh(x, 30, 0.04, 0.85, 2)
plot(x, y)
```

`SSpexpf` *self start for plateau-exponential function*

Description

Self starter for an plateau-exponential function

Usage

```
pexpf(x, a, xs, c)

SSpexpf(x, a, xs, c)
```

Arguments

x	input vector (x)
a	represents the value for the plateau
xs	represents the breakpoint at which the plateau ends
c	represents the exponential rate

Details

The equation is: $forx < xs : y = aandx >= xs : a * exp(c * (x - xs))$.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

pexpf: vector of the same length as x using the pexpf function

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:30
y <- pexpf(x, 20, 15, -0.2) + rnorm(30, 0, 1)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSpexpf(x, a, xs, c), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

`SSplin` *self start for plateau-linear function*

Description

Self starter for plateau-linear function with parameters a (plateau), xs (break-point), b (slope)

Usage

```
plin(x, a, xs, b)

SSplin(x, a, xs, b)
```

Arguments

`x`	input vector
`a`	the initial plateau
`xs`	break-point of transition between plateau and linear
`b`	the slope

Details

Initial plateau with a second linear phase. When $x < xs : y = a$ and when $x >= xs : y = a + b * (x - xs)$.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

plin: vector of the same length as x using the plateau-linear function

Examples

```
require(ggplot2)
set.seed(123)
x <- 1:30
y <- plin(x, 10, 20, 1) + rnorm(30, 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSplin(x, a, xs, b), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Confidence intervals
confint(fit)
```

SSpquad *self start for plateau-quadratic function*

Description

Self starter for plateau-quadratic function with parameters a (plateau), xs (break-point), b (slope), c (quadratic)

Usage

```
pquad(x, a, xs, b, c)

SSpquad(x, a, xs, b, c)
```

Arguments

x	input vector
a	the plateau value
xs	break-point of transition between plateau and quadratic
b	the slope (linear term)
c	quadratic term

Details

Reference for nonlinear regression Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

pquad: vector of the same length as x using the plateau-quadratic function

Examples

```
require(ggplot2)
set.seed(12345)
x <- 1:40
y <- pquad(x, 5, 20, 1.7, -0.04) + rnorm(40, 0, 0.6)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSpquad(x, a, xs, b, c), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
confint(fit)
```

SSprofd *self start for profile decay function*

Description

Self starter for a decay of a variable within a canopy (e.g. nitrogen concentration).

Usage

```
profd(x, a, b, c, d)

SSprofd(x, a, b, c, d)
```

Arguments

x	input vector (x)
a	represents the maximum value
b	represents the minimum value
c	represents the rate of decay
d	represents an empirical coefficient which provides flexibility

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506) and originally in Johnson et al. (2010) Annals of Botany 106: 735–749, 2010. (doi:10.1093/aob/mcq183).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

profd: vector of the same length as x using the profd function

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:10
y <- profd(x, 0.3, 0.05, 0.5, 4) + rnorm(10, 0, 0.01)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSprofd(x, a, b, c, d), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## profiling
## It does not work at a lower alphamax level
## Use this if you want to look at all four parameters
## par(mfrow=c(2,2))
plot(profile(fit, alphamax = 0.016))
## Reset graphical parameter as appropriate: par(mfrow=c(1,1))
## parameter 'd' is not well constrained
confint(fit, level = 0.9)
```

SSquadp *self start for quadratic-plateau function*

Description

Self starter for quadratic plateau function with parameters a (intercept), b (slope), c (quadratic), xs (break-point)

Usage

```
quadp(x, a, b, c, xs)

SSquadp(x, a, b, c, xs)
```

Arguments

x	input vector
a	the intercept
b	the slope
c	quadratic term
xs	break point of transition between quadratic and plateau

Details

Reference for nonlinear regression Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506).

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

quadp: vector of the same length as x using the quadratic-plateau function

Examples

```
require(ggplot2)
set.seed(123)
x <- 1:30
y <- quadp(x, 5, 1.7, -0.04, 20) + rnorm(30, 0, 0.6)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSquadp(x, a, b, c, xs), data = dat, algorithm = "port")
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSratio *self start for a rational curve*

Description

Self starter for a rational curve

Usage

```
ratio(x, a, b, c, d)

SSratio(x, a, b, c, d)
```

Arguments

x	input vector
a	parameter related to the maximum value of the response (numerator)
b	power exponent for numerator
c	parameter related to the maximum value of the response (denominator)
d	power exponent for denominator

Details

The equation is:

$$a * x^c/(1 + b * x^d)$$

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506). One example application is in Bril et al. (1994) https://edepot.wur.nl/333930 - pages 19 and 21. The parameters are difficult to interpret, but the function is very flexible. I have not tested this, but it might be beneficial to re-scale x and y to the (0,1) range if this function is hard to fit. https://en.wikipedia.org/wiki/Rational_function.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

ratio: vector of the same length as x using a rational function

Examples

```
require(ggplot2)
require(minpack.lm)
set.seed(1234)
x <- 1:100
y <- ratio(x, 1, 0.5, 1, 1.5) + rnorm(length(x), 0, 0.025)
dat <- data.frame(x = x, y = y)
fit <- nlsLM(y ~ SSratio(x, a, b, c, d), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SSricker *self start for Ricker Function*

Description

Self starter for Ricker function with parameters a and b

Usage

```
ricker(time, a, b)

SSricker(time, a, b)
```

Arguments

`time`	input vector (x) which is normally 'time', the smallest value should be close to zero.
`a`	which is related to the initial growth slope
`b`	which is related to the slowing down or decline

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506) and originally in Ricker, W. E. (1954) Stock and Recruitment Journal of the Fisheries Research Board of Canada, 11(5): 559–623. (doi:10.1139/f54-039). The equation is: $a * time * exp(-b * time)$.

Value

a numeric vector of the same length as x (time) containing parameter estimates for equation specified

ricker: vector of the same length as x (time) using the ricker function

Examples

```
require(ggplot2)
set.seed(123)
x <- 1:30
y <- 30 * x * exp(-0.3 * x) + rnorm(30, 0, 0.25)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSricker(x, a, b), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
```

SStrlin *self start for a trilinear Function*

Description

Self starter for a tri-linear function with parameters a (intercept), b (first slope), xs1 (first break-point), c (second slope), xs2 (second break-point) and d (third slope)

Usage

```
trlin(x, a, b, xs1, c, xs2, d)

SStrlin(x, a, b, xs1, c, xs2, d)
```

Arguments

x	input vector
a	the intercept
b	the first-phase slope
xs1	first break-point of transition between first-phase linear and second-phase linear
c	the second-phase slope
xs2	second break-point of transition between second-phase linear and third-phase linear
d	the third-phase slope

Details

This is a special case with just three parts (and two break points) but a more general approach is to consider a segmented function with several breakpoints and linear segments. Splines would be even more general. Also this model assumes that there are two break-points that needs to be estimated. The guess for the initial values splits the dataset in half, so it this will work when one break-point is in the first half and the second is in the second half.

Value

a numeric vector of the same length as x containing parameter estimates for equation specified

trlin: vector of the same length as x using the tri-linear function

See Also

package **segmented**.

Examples

```
require(ggplot2)
set.seed(1234)
x <- 1:30
y <- trlin(x, 0.5, 2, 10, 0.1, 20, 1.75) + rnorm(30, 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SStrlin(x, a, b, xs1, c, xs2, d), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Minimal example
## This is probably about the smallest dataset you
## should use with this function
dat2 <- data.frame(x = 1:8, y = c(1.1, 1.9, 3.1, 2.5, 1.4, 0.9, 2.2, 2.9))
fit2 <- nls(y ~ SStrlin(x, a, b, xs1, c, xs2, d), data = dat2)
## expangin for plotting
ndat <- data.frame(x = seq(1, 8, by = 0.1))
ndat$prd <- predict(fit2, newdata = ndat)
```

```
ggplot() +
  geom_point(data = dat2, aes(x = x, y = y)) +
  geom_line(data = ndat, aes(x = x, y = prd))
```

---

swpg *Water limitations for Soybean growth*

Description

Simulated data based on obseved data presented in Sinclair (1986) - Fig. 1A

Usage

```
swpg
```

Format

A data frame with 20 rows and 3 columns

ftsw Fraction of Transpirable Soil Water (0-1)

lfgr Relative Leaf Growth scaled from 0 to 1

Details

Sinclair, T.R. Water and Nitrogen Limitations in Soybean Grain Production I. Model Development. Field Crops Research. 125-141.

Source

Simluated data (much cleaner than original) based on the above publication

---

var_cov *Variance Covariance matrix of for g(n)ls and (n)lme models*

Description

Extracts the variance covariance matrix (residuals, random or all)

Usage

```
var_cov(
  object,
  type = c("residual", "random", "all"),
  aug = FALSE,
  sparse = FALSE
)
```

Arguments

object	object which inherits class lm, gls or lme
type	"residual" for the variance-covariance for the residuals, "random" for the variance-covariance of the random effects or "all" for the sum of both.
aug	whether to augment the matrix of the random effects to the dimensions of the data
sparse	whether to return a sparse matrix (default is FALSE)

Details

Variance Covariance matrix for (non)linear mixed models

Value

It returns a matrix or a sparse matrix Matrix.

Note

See Chapter 5 of Pinheiro and Bates. This returns potentially a very large matrix of N x N, where N is the number of rows in the data.frame. The function getVarCov only works well for lme objects.
The equivalence is more or less:
getVarCov type = "random.effects" equivalent to var_cov type = "random".
getVarCov type = "conditional" equivalent to var_cov type = "residual".
getVarCov type = "marginal" equivalent to var_cov type = "all".
The difference is that getVarCov has an argument that specifies the individual for which the matrix is being retrieved and var_cov returns the full matrix only.

See Also

getVarCov

Examples

```
require(graphics)
require(nlme)
data(ChickWeight)
## First a linear model
flm <- lm(weight ~ Time, data = ChickWeight)
vlm <- var_cov(flm)
## First model with no modeling of the Variance-Covariance
fit0 <- gls(weight ~ Time, data = ChickWeight)
v0 <- var_cov(fit0)
## Only modeling the diagonal (weights)
fit1 <- gls(weight ~ Time, data = ChickWeight, weights = varPower())
v1 <- var_cov(fit1)
## Only the correlation structure is defined and there are no groups
fit2 <- gls(weight ~ Time, data = ChickWeight, correlation = corAR1())
v2 <- var_cov(fit2)
## The correlation structure is defined and there are groups present
```

```
fit3 <- gls(weight ~ Time, data = ChickWeight, correlation = corCAR1(form = ~ Time | Chick))
v3 <- var_cov(fit3)
## There are both weights and correlations
fit4 <- gls(weight ~ Time, data = ChickWeight,
            weights = varPower(),
            correlation = corCAR1(form = ~ Time | Chick))
v4 <- var_cov(fit4)
## Tip: you can visualize these matrices using
image(v4[,ncol(v4):1])
```

Index