Package ‘nlraa’

August 18, 2021

Version 0.98
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 simulations 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, lattice, minpack.lm, NISTnls, nlstools, nls2, parallel, rmarkdown, segmented

LazyData true
LazyDataCompression xz
RoxygenNote 7.1.1
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,
R topics documented:

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 2021-08-18 21:50:07 UTC

R topics documented:

barley ................................................. 3
boot_lm ............................................. 4
boot_lme ........................................... 5
boot_nlme .......................................... 6
boot_nls ........................................... 7
fm1.P.at.x.0.4 ..................................... 8
fm1.P.bt .......................................... 9
fm1.P.bt.ft ....................................... 9
fm2.Lob.bt ....................................... 10
fmm1.bt .......................................... 10
IA_tab ............................................ 11
IC_tab ............................................ 13
lfmc .............................................. 13
Lob.bt.pe ......................................... 14
maizeleafext ...................................... 14
nlraa.env ........................................ 15
nlsLMList ......................................... 15
nlsLMList.formula ................................. 16
predict2_nls ...................................... 17
predict_gam ....................................... 19
predict_nlme ...................................... 21
predict_nls ....................................... 23
R2M ................................................. 25
simulate_gam ...................................... 27
simulate_gls ...................................... 29
simulate_gnls .................................... 30
simulate_lm ....................................... 31
simulate_lme ...................................... 33
simulate_nlme .................................... 34
simulate_nlme_one ............................... 36
simulate_nls ..................................... 37
sm .................................................. 38
SSbell ............................................ 39
SSbeta5 ........................................... 40
SSbg4rp .......................................... 41
SSbgf ............................................. 42
SSbgf4 ........................................... 44
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


---

### boot_lm

**Bootstrapping for linear models**

### Description

Bootstrapping for linear models

### Usage

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

### 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”.
- `data`: optional data argument (useful/needed when data are not in an available environment).
- `...`: additional arguments to be passed to function `boot`

### Details

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

### Note

at the moment, when the argument data is used, it is not possible to check that it matches the original data used to fit the model. It will also override the fetching of data.
### Examples

```r
top
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

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

```r
boot_gls(object, f = NULL, R = 999, psim = 1, cores = 1L, data = NULL, ...)
```

### Arguments

- **object**: object of class `lme` or `gls`
- **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
- **data**: optional data argument (useful/needed when data are not in an available environment).
- **...**: 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

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

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

hist(fml.bt)
```

---

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

**Description**

Bootstraping tools for nonlinear models using a consistent interface bootstrap function for objects of class `gnls`

**Usage**

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

```r
boot_gnls(object, f = NULL, R = 999, psim = 1, cores = 1L, data = NULL, ...)
```

**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
- `data`  
  optional data argument (useful/needed when data are not in an available environment).
- `...`  
  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

```r
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")
```

---

boot_nls

### Bootstrapping for nonlinear models

**Description**

Bootstraping for nonlinear models

**Usage**

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

**Arguments**

- **object**: object of class `nls`
- **f**: function to be applied (and bootstrapped), default `coef`
- **R**: number of bootstrap samples, default 999
- **psim**: simulation level for `simulate_nls`
- **resid.type**: either “resample”, “normal” or “wild”.
data  optional data argument (useful/needed when data are not in an available environment).
... additional arguments to be passed to function boot

Details
The residuals can either be generated by resampling with replacement (default or non-parametric), from a normal distribution (parameteric) or by changing their signs (wild). This last one is called “wild bootstrap”. There is more information in boot.lm.

Note
at the moment, when the argument data is used, it is not possible to check that it matches the original data used to fit the model. It will also override the fetching of data.

See Also
Boot

Examples

```r
require(car)
data(barley, package = "nlraa")
## Fit a linear-plateau
fit.nls <- nls(yield ~ SSlinp(NF, a, b, xs), data = barley)

## Bootstrap coefficients by default
## Keeping R small for simplicity, increase R for a more realistic use
fit.nls.bt <- boot_nls(fit.nls, R = 1e2)
## Compute confidence intervals
confint(fit.nls.bt, type = "perc")
## Visualize
hist(fit.nls.bt, 1, ci = "perc", main = "Intercept")
hist(fit.nls.bt, 2, ci = "perc", main = "linear term")
hist(fit.nls.bt, 3, ci = "perc", main = "xs break-point term")
```

Object for confidence bands vignette fm1.P.at.x.0.4

Description
object for confidence bands vignette fm1.P.at.x.0.4

Usage

`fm1.P.at.x.0.4`
Format
An object of class ‘boot’

fm1.P.at.x.0.4  object created in the vignette in chunk ‘Puromycin-6’

Source
this package vignette

fm1.P.bt  

Description
object for confidence bands vignette fm1.P.bt

Usage
fm1.P.bt

Format
An object of class ‘boot’

fm1.P.bt  object created in the vignette in chunk ‘Puromycin-2’

Source
this package vignette

fm1.P.bt.ft  

Description
object for confidence bands vignette fm1.P.bt.ft

Usage
fm1.P.bt.ft

Format
An object of class ‘boot’

fm1.P.bt.ft  object created in the vignette in chunk ‘Puromycin-4’

Source
this package vignette
### fm2.Lob.bt

**object for confidence bands vignette fm2.Lob.bt**

**Description**

object for confidence bands vignette fm2.Lob.bt

**Usage**

fm2.Lob.bt

**Format**

An object of class 'boot'

**fm2.Lob.bt** object created in the vignette in chunk 'Loblolly-methods-2'

**Source**

this package vignette

---

### fmm1.bt

**object for confidence bands vignette fmm1.bt**

**Description**

object for confidence bands vignette fmm1.bt

**Usage**

fmm1.bt

**Format**

An object of class 'boot'

**fmm1.bt** object created in the vignette in chunk 'maizeleafext-2'

**Source**

this package vignette
Description

Indexes of agreement
plotting function for a IA_tab, it requires ‘ggplot2’

Usage

IA_tab(obs, sim, object, null.object)

## S3 method for class 'IA_tab'
plot(x, y, ..., type = c("OvsS", "RvsS"))

Arguments

obs vector with observed data
sim vector with simulated data (should be the same length as observed)
object alternative to the previous two arguments. An object of class ‘lm’, ‘nls’ or ‘lme’
null.object optional object which represents the ‘null’ model. It is an intercept-only model by default.
x object of class ‘IA_tab’.
y not used at the moment
... additional plotting arguments (none use at the moment).
type either “OvsS” (observed vs. simulated) or “RvsS” (residuals vs. simulated).

Details

This function returns several indexes that might be useful for interpretation
For objects of class ‘lm’ or ‘nls’

bias: mean(obs - sim)
intercept: intercept of the model obs ~ beta_0 + beta_1 * sim + error
slope: slope of the model obs ~ beta_0 + beta_1 * sim + error
RSS (deviance): residual sum of squares of the previous model
MSE (RSS / n): mean squared error; where n is the number of observations
RMSE: squared root of the previous index
R2.1: R-squared extracted from an ‘lm’ object
R2.2: R-squared computed as the correlation between observed and simulated to the power of 2.
ME: model efficiency
NME: Normalized model efficiency
Corr: correlation between observed and simulated
ConCorr: concordance correlation
For objects of class ‘gls’, ‘gnls’, ‘lme’ or ‘nlme’ there are additional metrics such as:

https://en.wikipedia.org/wiki/Nash-Sutcliffe_model_efficiency_coefficient
https://en.wikipedia.org/wiki/Concordance_correlation_coefficient

See Also

IC_tab

Examples

```r
require(nlme)
require(ggplot2)
## Fit a simple model and then compute IAs
data(swpg)
'## Linear model
fit0 <- lm(lfgr ~ ftsw + I(ftsw^2), data = swpg)
ias0 <- IA_tab(object = fit0)
ias0$IA_tab
## Nonlinear model
fit1 <- nls(lfgr ~ SSblin(ftsw, a, b, xs, c), data = swpg)
ias1 <- IA_tab(object = fit1)
ias1$IA_tab
plot(ias1)
## Linear Mixed Models
data(barley, package = "nlraa")
fit2 <- lme(yield ~ NF + I(NF^2), random = ~ 1 | year, data = barley)
ias2 <- IA_tab(object = fit2)
ias2$IA_tab
## Nonlinear Mixed Model
barleyG <- groupedData(yield ~ NF | year, data = barley)
fit3L <- nlsLMList(yield ~ SSquadp3(NF, a, b, c), data = barleyG)
fit3 <- nlme(fit3L, random = pdDiag(a + b ~ 1))
ias3 <- IA_tab(object = fit3)
ias3$IA_tab
plot(ias3)
## Plotting model
prds <- predict_nlme(fit3, interval = "conf", plevel = 0)
barleyGA <- cbind(barleyG, prds)
ggplot(data = barleyGA, aes(x = NF, y = yield)) +
geom_point() +
geom_line(aes(y = Estimate)) +
geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5),
    fill = "purple", alpha = 0.2)
## R2M for model 2
R2M(fit2)
## R2M for model 3
R2M(fit3)
```
IC_tab

Information Criteria Table

Description

Information criteria table with weights

Usage

IC_tab(..., criteria = c("AIC", "AICc", "BIC"), sort = TRUE)

Arguments

... model fit objects fitted to the same data
criteria either ‘AIC’, ‘AICc’ or ‘BIC’.
sort whether to sort by weights (default to TRUE)

Note

The delta and weights are calculated based on the ‘criteria’

See Also

ICtab

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
maizeleafext

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

Source
doi: 10.1002/ecd3.5543

| Lob.bt.pe | object for confidence bands vignette Lob.bt.pe |

Description
object for confidence bands vignette Lob.bt.pe

Usage
Lob.bt.pe

Format
An object of class ‘boot’

Lob.bt.pe object created in the vignette in chunk ‘Loblolly-bootstrap-estimates-1’

Source
this package vignette

| maizeleafext | Maize leaf extension rate as a response to temperature |

Description
Data on leaf extension rate as a response to meristem temperature in maize. The data are re-created liberally from Walls, W.R., 1971. Role of temperature in the regulation of leaf extension in Zea mays. Nature, 229: 46-47. The data points are not the same as in the original paper. Some additional points were inserted to fill in the blanks and allow for reasonable parameter estimates

Usage
maizeleafext

Format
A data frame with 10 rows and 2 columns

temp Meristem temperature (in Celsius).
rate Leaf extension rate (relative to 25 degrees).
Source

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 (optionally) '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.
Usage

```r
## 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`

Description

The method used in this function is described in Battes and Watts (2007) Nonlinear Regression Analysis and Its Applications (see pages 58-59). It is known as the Delta method.

Usage

```r
predict2_nls(
  object,
  newdata = NULL,
  interval = c("none", "confidence", "prediction"),
  level = 0.95
)
```
predict2_nls

Arguments

- **object**: object of class ‘nls’
- **newdata**: data frame with values for the predictor
- **interval**: either ‘none’, ‘confidence’ or ‘prediction’
- **level**: probability level (default is 0.95)

Details

This method is approximate and it works better when the distribution of the parameter estimates are normally distributed. This assumption can be evaluated by using bootstrap.

Value

- a data frame with Estimate, Est.Error, lower interval bound and upper interval bound. For example, if the level = 0.95, the lower bound would be named Q2.5 and the upper bound would be name Q97.5

See Also

- `predict.nls` and `predict_nls`

Examples

```r
require(ggplot2)
require(nlme)
data(Soybean)
SoyF <- subset(Soybean, Variety == "F" & Year == 1988)
fml <- nls(weight ~ SSlogis(Time, Asym, xmid, scal), data = SoyF)
## The SSlogis also supplies analytical derivatives
## therefore the predict function returns the gradient too
prd1 <- predict(fm1, newdata = SoyF)
## Gradient
head(attr(prd1, "gradient"))
## Prediction method using gradient
prds <- predict2_nls(fm1, interval = "conf")
SoyFA <- cbind(SoyF, prds)
ggplot(data = SoyFA, aes(x = Time, y = weight)) +
  geom_point() +
  geom_line(aes(y = Estimate)) +
  geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5), fill = "purple", alpha = 0.3) +
  ggtitle("95% Confidence Bands")
fm2 <- nls(weight ~ Asym/(1 + exp((xmid - Time)/scal)), data = SoyF,
  start = c(Asym = 20, xmid = 56, scal = 8))
## Using predict.nls on this object will not return a gradient
## so predict2_nls will fail
predict(fm2)
## For this reason, functions which use a selfStart are recommended
## Prediction interval
prdi <- predict2_nls(fm1, interval = "pred")
```
## Make prediction interval plot

```r
ggplot(data = SoyFA.PI, aes(x = Time, y = weight)) +
  geom_point() +
  geom_line(aes(y = Estimate)) +
  geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5), fill = "purple", alpha = 0.3) +
  ggtitle("95% Prediction Band")
```

## For these data we should be using `gnls` instead with an increasing variance

```r
fmg1 <- `gnls`(weight ~ SSlogis(Time, Asym, xmid, scal),
  data = SoyF, weights = varPower())
```

```r
IC_tab(fm1, fmg1)
```

```r
prdg <- predict_nlme(fmg1, plevel = 1, interval = "pred")
```

```r
SoyFA.GPI <- cbind(SoyF, prdg)
```

## These prediction bands are not perfect, but they could be smoothed

```r
## to eliminate the ragged appearance

ggplot(data = SoyFA.GPI, aes(x = Time, y = weight)) +
  geom_point() +
  geom_line(aes(y = Estimate)) +
  geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5), fill = "purple", alpha = 0.3) +
  ggtitle("95% Prediction Band. NLS model which \naccomodates an increasing variance")
```

---

### predict_gam

**Modified prediction function based on predict.gam**

#### Description

Largely based on `predict.gam`, but with some minor modifications to make it compatible with `predict_nls`

#### Usage

```r
predict_gam(
  object,
  newdata = NULL,
  type = "link",
  se.fit = TRUE,
  terms = NULL,
  exclude = NULL,
  block.size = NULL,
  newdata.guaranteed = FALSE,
  na.action = na.pass,
  unconditional = FALSE,
  iters.type = NULL,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  tvalue = NULL,
  ...
)
```
**predict_gam**

**Arguments**

- **object**: object of class ‘gam’ or as returned by function ‘gamm’
- **newdata**: see `predict.gam`
- **type**: see `predict.gam`
- **se.fit**: see `predict.gam`. Notice that the default is changed to TRUE.
- **terms**: see `predict.gam`
- **exclude**: see `predict.gam`
- **block.size**: see `predict.gam`
- **newdata.guaranteed**: see `predict.gam`
- **na.action**: see `predict.gam`
- **unconditional**: see `predict.gam`
- **iterms.type**: see `predict.gam`
- **interval**: either ‘none’, ‘confidence’ or ‘prediction’.
- **level**: probability level for the interval (default 0.95)
- **tvalue**: t-value statistic used for constructing the intervals
- **...**: additional arguments to be passed to `predict.gam`.

**Value**

numeric vector of the same length as the fitted object when interval is equal to ‘none’. Otherwise, a data.frame with columns named (for a 0.95 level) ‘Estimate’, ‘Est.Error’, ‘Q2.5’ and ‘Q97.5’

**Note**

this is a very simple wrapper for `predict.gam`.

**See Also**

`predict.lm`, `predict.nls`, `predict.gam`, `simulate_nls`, `simulate_gam`

**Examples**

```r
require(ggplot2)
require(mgcv)
data(barley)

fm.G <- gam(yield ~ s(NF, k = 6), data = barley)

## confidence and prediction intervals
cis <- predict_gam(fm.G, interval = "conf")
pis <- predict_gam(fm.G, interval = "pred")

barleyA.ci <- cbind(barley, cis)
barleyA.pi <- cbind(barley, pis)
```
ggplot() +
  geom_point(data = barleyA.ci, aes(x = NF, y = yield)) +
  geom_line(data = barleyA.ci, aes(x = NF, y = Estimate)) +
  geom_ribbon(data = barleyA.ci, aes(x = NF, ymin = Q2.5, ymax = Q97.5),
              color = "red", alpha = 0.3) +
  geom_ribbon(data = barleyA.pi, aes(x = NF, ymin = Q2.5, ymax = Q97.5),
              color = "blue", alpha = 0.3) +
  ggtitle("95% confidence and prediction bands")

---

`predict_nlme`  
Average predictions from several (non)linear models based on IC weights

**Description**

Computes weights based on AIC, AICc, or BIC and it generates weighted predictions by the relative value of the IC values

- predict function for objects of class `lme`
- predict function for objects of class `gnls`
- predict function for objects of class `gls`

**Usage**

```r
predict_nlme(
  ..., 
  criteria = c("AIC", "AICc", "BIC"),
  interval = c("none", "confidence", "prediction", "new-prediction"),
  level = 0.95,
  nsim = 1000,
  plevel = 0,
  newdata = NULL
)
```

```r
predict_lme(
  ..., 
  criteria = c("AIC", "AICc", "BIC"),
  interval = c("none", "confidence", "prediction", "new-prediction"),
  level = 0.95,
  nsim = 1000,
  plevel = 0,
  newdata = NULL
)
```

```r
predict_gnls(
  ...
)
```
...,
criteria = c("AIC", "AICc", "BIC"),
interval = c("none", "confidence", "prediction", "new-prediction"),
level = 0.95,
nsim = 1000,
plevel = 0,
newdata = NULL
)
predict_gls(
...

criteria = c("AIC", "AICc", "BIC"),
interval = c("none", "confidence", "prediction", "new-prediction"),
level = 0.95,
nsim = 1000,
plevel = 0,
newdata = NULL
)

Arguments

... nlme, lme, gls or gnls objects.
criteria either 'AIC', 'AICc' or 'BIC'.
interval either 'none', 'confidence' or 'prediction'. It is also possible to choose 'new-
prediction', which is a prediction that resamples the random effects (only rele-
vant for 'lme' or 'nlme' objects.)
level probability level for the interval (default 0.95)
nsim number of simulations to perform for intervals. Default 1000.
plevel parameter level prediction to be passed to predicition functions.
newdata new data frame for predictions

Value
numeric vector of the same length as the fitted object.

Note
all the objects should be fitted to the same data. The weights are based on the IC value.

See Also
predict.nlme predict.lme predict.gnls

Examples

## Example
require(ggplot2)
require(nlme)
data(Orange)

## All models should be fitted using Maximum Likelihood
fm.L <- nlme(circumference ~ SSlogis(age, Asym, xmid, scal),
             random = pdDiag(Asym + xmid + scal ~ 1),
             method = "ML", data = Orange)
fm.G <- nlme(circumference ~ SSgompertz(age, Asym, b2, b3),
             random = pdDiag(Asym + b2 + b3 ~ 1),
             method = "ML", data = Orange)
fm.F <- nlme(circumference ~ SSfpl(age, A, B, xmid, scal),
             random = pdDiag(A + B + xmid + scal ~ 1),
             method = "ML", data = Orange)
fm.B <- nlme(circumference ~ SSbg4rp(age, w.max, lt.e, ldtm, ldtb),
             random = pdDiag(w.max + lt.e + ldtm + ldtb ~ 1),
             method = "ML", data = Orange)

## Print the table with weights

## Each model prediction is weighted according to their AIC values

ggplot(data = Orange, aes(x = age, y = circumference)) +
  geom_point() +
  geom_line(aes(y = predict(fm.L, level = 0), color = "Logistic")) +
  geom_line(aes(y = predict(fm.G, level = 0), color = "Gompertz")) +
  geom_line(aes(y = predict(fm.F, level = 0), color = "4P-Logistic")) +
  geom_line(aes(y = predict(fm.B, level = 0), color = "Beta")) +
  geom_line(aes(y = prd, color = "Avg. Model"), size = 1.2)

---

**predict_nls**

**Average predictions from several (non)linear models based on IC weights**

**Description**

Computes weights based on AIC, AICc, or BIC and it generates weighted predictions by the relative value of the IC values

**Usage**

```
predict_nls(
    ...,  
    criteria = c("AIC", "AICc", "BIC"),
    interval = c("none", "confidence", "prediction"),
    level = 0.95,
)```
nsim = 1000,
resid.type = c("none", "resample", "normal", "wild"),
newdata = NULL
)
predict2_gam(  
...,  
criteria = c("AIC", "AICc", "BIC"),  
interval = c("none", "confidence", "prediction"),  
level = 0.95,  
nsim = 1000,  
resid.type = c("none", "resample", "normal", "wild"),  
newdata = NULL
)

Arguments

... 'nls' or 'lm' objects ('glm' and 'gam' objects inherit 'lm').
criteria either 'AIC', 'AICc' or 'BIC'.
interval either 'none', 'confidence' or 'prediction'.
level probability level for the interval (default 0.95)
nsim number of simulations to perform for intervals. Default 1000.
resid.type either 'none', "resample", "normal" or "wild".
newdata new data frame for predictions

Value

numeric vector of the same length as the fitted object when interval is equal to 'none'. Otherwise, a
data.frame with columns named (for a 0.95 level) 'Estimate', 'Est.Error', 'Q2.5' and 'Q97.5'

Note

all the objects should be fitted to the same data. Weights are based on the chosen IC value (exp(-0.5 * delta IC)). For models of class gam there is very limited support.

See Also

predict_lm, predict_nls, predict_gam, simulate_nls, simulate_gam

Examples

## Example
require(ggplot2)
require(mgcv)
data(barley, package = "nlraa")

fm.L <- lm(yield ~ NF, data = barley)
R2M <- lm(yield ~ NF + I(NF^2), data = barley)
fm.A <- nls(yield ~ SSasymp(NF, Asym, R0, lrc), data = barley)
fm.LP <- nls(yield ~ SSlinp(NF, a, b, xs), data = barley)
fm.QP <- nls(yield ~ SSquadp3(NF, a, b, c), data = barley)
fm.BL <- nls(yield ~ SSblin(NF, a, b, xs, c), data = barley)
fm.G <- gam(yield ~ s(NF, k = 6), data = barley)

## Print the table with weights

## Each model prediction is weighted according to their AIC values

ggplot(data = barley, aes(x = NF, y = yield)) +
  geom_point() +
  geom_line(aes(y = fitted(fm.L), color = "Linear")) +
  geom_line(aes(y = fitted(fm.Q), color = "Quadratic")) +
  geom_line(aes(y = fitted(fm.A), color = "Asymptotic")) +
  geom_line(aes(y = fitted(fm.LP), color = "Linear-plateau")) +
  geom_line(aes(y = fitted(fm.QP), color = "Quadratic-plateau")) +
  geom_line(aes(y = fitted(fm.BL), color = "Bi-linear")) +
  geom_line(aes(y = fitted(fm.G), color = "GAM")) +
  geom_line(aes(y = prd, color = "Avg. Model"), size = 1.2)

---

**R2M**

*R-squared for nonlinear mixed models*

**Description**

R-squared ‘modified’ for nonlinear (mixed) models

**Usage**

R2M(x, ...)

## S3 method for class 'nls'
R2M(x, ...)

## S3 method for class 'lm'
R2M(x, ...)

## S3 method for class 'gls'
R2M(x, ...)

## S3 method for class 'gnls'
R2M(x, ...)

## S3 method for class 'lme'
R2M(x, ...)  
## S3 method for class 'nlme'
R2M(x, ...)

**Arguments**

- **x**: object of class ‘lm’, ‘nls’, ‘gls’, ‘gnls’, ‘lme’ or ‘nlme’.
- **...**: additional arguments (none use at the moment).

**Details**

I have read some papers about computing an R-squared for (non)linear (mixed) models and I am not sure that it makes sense at all. However, here they are and the method is general enough that it extends to nonlinear mixed models. What do these numbers mean and why would you want to compute them are good questions to ponder...

**Recommended reading**:

https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faq-what-are-pseudo-r-squareds/


**Value**

it returns a list with the following structure:
- for an object of class ‘lm’, ‘nls’, ‘gls’ or ‘gnls’,
  - R2: R-squared
  - var.comp: variance components with var.fixed and var.resid
  - var.perc: variance components percents (should add up to 100)
- for an object of class ‘lme’ or ‘nlme’ in addition it also returns:
  - R2.marginal: marginal R2 which only includes the fixed effects
  - R2.conditional: conditional R2 which includes both the fixed and random effects
  - var.random: the variance contribution of the random effects

**Note**

The references here strongly discourage the use of R-squared in anything but linear models.
simulate_gam

Simulate responses from a generalized additive linear model \texttt{gam}

Description

By sampling from the vector of coefficients it is possible to simulate data from a \textquote{gam} model.

Usage

\begin{verbatim}
simulate_gam(
  object,
  nsim = 1,
  psim = 1,
  resid.type = c("none", "resample", "normal", "wild"),
  value = c("matrix", "data.frame"),
  ...
)
\end{verbatim}

Arguments

- **object**: object of class \texttt{gam} or \texttt{glm}.
- **nsim**: number of simulations to perform.
- **psim**: parameter simulation level (an integer, 0, 1, 2 or 3).
- **resid.type**: type of residual to use. \textquote{none}, \textquote{resample}, \textquote{normal} or \textquote{wild}.
- **value**: either \textquote{matrix} or \textquote{data.frame}.
- **...**: additional arguments (none used at the moment).
Details

This function is probably redundant. Simply using simulate generates data from the correct distribution for objects which inherit class lm. The difference is that I'm trying to add the uncertainty in the parameter estimates.

These are the options that control the parameter simulation level

- \( \text{psim} = 0 \) returns the fitted values
- \( \text{psim} = 1 \) simulates from a beta vector (mean response)
- \( \text{psim} = 2 \) simulates observations according to the residual type (similar to observed data)
- \( \text{psim} = 3 \) simulates a beta vector, considers uncertainty in the variance covariance matrix of beta and adds residuals (prediction)

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

Note

\( \text{psim} = 3 \) is not implemented at the moment.

The purpose of this function is to make it compatible with other functions in this package. It has some limitations compared to the functions in the ‘see also’ section.

References


See Also

predict, predict.gam, simulate and simulate.lm.

Examples

```r
require(ggplot2)
require(mgcv)
## These count data are from GAM book by Simon Wood (pg. 132) - see reference
y <- c(12, 14, 33, 50, 67, 74, 123, 141, 165, 204, 253, 246, 240)
t <- 1:13
dat <- data.frame(y = y, t = t)
fit <- gam(y ~ t + I(t^2), family = poisson, data = dat)
sims <- simulate_gam(fit, nsim = 100, value = "data.frame")
```
**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. This generates just one simulation from these type of models. To generate multiple simulations use **simulate_lme**

### Usage

```r
simulate_gls(
  object,
  psim = 1,
  na.action = na.fail,
  naPattern = NULL,
  data = 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**
- **data**: the data argument is needed when using this function inside user defined functions. It should be identical to the data used to fit the model.
- **...**: 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.
simulate_gnls

**Value**

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

**See Also**

predict.gls simulate_lme

**Examples**

```r
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**

```r
simulate_gnls(
  object, 
  psim = 1, 
  na.action = na.fail, 
  naPattern = NULL, 
  data = 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.
**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**

```r
simulate_lm(
  object,
  psim = 1,
  nsim = 1,
  resid.type = c("none", "resample", "normal", "wild"),
)```

**Details**

This function is based on `predict.gnls` function. It uses function `mvtnorm` 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**

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

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

- **object**: object of class `lm`
- **psim**: parameter simulation level (an integer, 0, 1, 2, 3 or 4).
- **nsim**: number of simulations to perform
- **resid.type**: type of residual to include (none, resample, normal or wild)
- **value**: either ‘matrix’ or ‘data.frame’
- **data**: the data argument might be needed when using this function inside user defined functions. At least it is expected to be safer.
- **...**: 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. When psim = 2 and resid.type = none, simulate is used instead. ‘wild’ does not assume constant variance, but it assumes symmetry.

Value

matrix or data.frame with responses

References

Examples

```r
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, and there is experimental code for considering the ‘correlation’ structure.

**Usage**

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

**Arguments**

- **object** object of class lme or gls
- **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
- **data** the data argument is needed when using this function inside user defined functions.
- **...** additional arguments (it is possible to supply a newdata this way)
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.

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

I find the `simulate.merMod` in the lme4 package 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` and `simulate.merMod` in the 'lme4' package.

Examples

```r
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

Simulate multiple samples from a nonlinear model

**Usage**

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

  data = NULL,
  ...
)

Arguments

  object       object of class gnls or nlme
  nsim         number of samples, default 1
  psim         simulation level for fixed and random parameters see simulate_nlme_one for
               more details.
  value        whether to return a matrix (default) or an augmented data frame
  data         the data argument is needed when using this function inside user defined func-
               tions.
  ...          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 further plotting more convenient.

Examples

  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))

  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,
  data = 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. Currently, working on psim = 3, which will simulate a new set of random
          effects. This can be useful when computing prediction intervals at the subject-
          level.
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 optional missing value action. See predict.nlme
naPattern optional missing value pattern. See predict.nlme
data      the data argument is needed when using this function inside user defined func-
          tions.
...       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
**simulate_nls**

Simulate fitted values from an object of class **nls**

**Description**

Simulate values from an object of class **nls**.

**Usage**

```r
simulate_nls(
  object,
  nsim = 1,
  psim = 1,
  resid.type = c("none", "resample", "normal", "wild"),
  value = c("matrix", "data.frame"),
  data = NULL,
  ...
)
```

**Arguments**

- `object`: object of class **nls**
- `nsim`: number of simulations to perform
- `psim`: parameter simulation level, 0: for fitted values, 1: for simulation from fixed parameters (assuming a fixed vcov matrix), 2: simulation from sampling both from the parameters and the residuals, 3: for simulation considering the uncertainty in the residual standard error only (sigma) and fixing the parameter estimates at their original value; this will result in simulations similar to the observed values.
- `resid.type`: either 'none', "resample", "normal" or "wild".
- `value`: either 'matrix' or 'data.frame'
- `data`: the data argument is needed when using this function inside user defined functions.
- `...`: additional arguments (it is possible to supply a newdata this way)

**Details**

This function is based on **predict.gnls** function

It uses function **mvnorm** 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**

This function should return a vector with the same dimensions as the original data, unless newdata is provided.
Value

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

Note

The default behavior is that simulations are performed for the mean function only. When ‘psim = 2’ this function will silently choose ‘resample’ as the ‘resid.type’. This is not ideal design for this function, but I made this choice for compatibility with other types of simulation originating from glm and gam.

See Also

predict.gnls, predict_nls

Examples

```
data(barley, package = "nlraa")
fit <- nls(yield ~ SSlinp(NF, a, b, xs), data = barley)
sim <- simulate_nls(fit, nsim = 100)
```

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.


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


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

```r
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)))
```

SSbeta5

**self start for Beta 5-parameter function**

Description

Self starter for Beta 5-parameter function

Usage

```r
beta5(temp, mu, tb, a, tc, b)
SSbeta5(temp, mu, tb, a, tc, b)
```

Arguments

- `temp` input vector which is normally ‘temperature’
- `mu` mu parameter (see equation)
- `tb` base (low) temperature at which no expansion occurs
- `a` parameter describing the initial increasing phase
- `tc` critical (high) temperature at which no expansion occurs
- `b` parameter describing the decreasing phase

Details

For details see the publication by Yin et al. (1995) “A nonlinear model for crop development as a function of temperature”. Agricultural and Forest Meteorology 77 (1995) 1-16

The form of the equation is:

\[
exp(mu) \ast (temp - tb)^a \ast (tc - temp)^b
\]
Value

\textit{beta5}: vector of the same length as \textit{x} (temp) using the \textit{beta5} function

Examples

```r
require(minpack.lm)
require(ggplot2)
## Temperature response example
data(maizeleafext)
## Fit model
fit <- nlsLM(rate ~ SSbeta5(temp, mu, tb, a, tc, b), data = maizeleafext)
## Visualize
ndat <- data.frame(temp = 0:45)
ndat$rate <- predict(fit, newdata = ndat)
ggplot() +
  geom_point(data = maizeleafext, aes(temp, rate)) +
  geom_line(data = ndat, aes(x = temp, y = rate))
```

\textit{SSbg4rp} \hspace{1cm} \textit{self start for the reparameterized Beta growth function with four parameters}

Description

Self starter for Beta Growth function with parameters \textit{w.max}, \textit{lt.e}, \textit{ldtm}, \textit{ldtb}

Usage

```r
bg4rp(time, w.max, lt.e, ldtm, ldtb)
SSbg4rp(time, w.max, lt.e, ldtm, ldtb)
```

Arguments

- \textit{time}: input vector (\textit{x}) which is normally \textit{‘time’}, the smallest value should be close to zero.
- \textit{w.max}: value of weight or mass at its peak
- \textit{lt.e}: log of the time at which the maximum weight or mass has been reached.
- \textit{ldtm}: log of the difference between time at which the weight or mass reaches its peak and half its peak.
- \textit{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:

\[
\text{w.max} \times (1 + \frac{\exp(lt.e) - \text{time}}{\exp(ldtm)}) \times \frac{((\text{time} - (\exp(lt.e) - \exp(ldtb)))}{\exp(ldtb)\times\exp(ldtm)}
\]

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

```r
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**

\[
\text{bfg}(\text{time}, \text{w.max}, \text{t.e}, \text{t.m})
\]

\[
\text{SSbfg}(\text{time}, \text{w.max}, \text{t.e}, \text{t.m})
\]

\[
\text{bfg2}(\text{time}, \text{w.max}, \text{w.b}, \text{t.e}, \text{t.m}, \text{t.b})
\]

**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:

\[
\text{w.max} \ast \left(1 + \frac{(\text{t.e} - \text{time})}{(\text{t.e} - \text{t.m})}\right) \ast \left(\text{time}/\text{t.e}\right)^{\text{t.e}/(\text{t.e} - \text{t.m})}
\]

. Given this function weight is expected to decay and reach zero again at \(2 \ast \text{t.e} - \text{t.m}\).

**Value**

- **bfg**: vector of the same length as x (time) using the beta growth function
- **bfg2**: a numeric vector of the same length as x (time) containing parameter estimates for equation specified

**Examples**

```r
## See extended example in vignette 'nlraa-AgronJ-paper'
x <- seq(0, 17, by = 0.25)
y <- bfg(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**

```r
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**

```r
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:

\[ w.max \times \left(1 + \frac{\exp(lt.e) - time}{\exp(ldt)}\right) \times \left(\frac{time}{\exp(lt.e)}\right)\frac{1}{\exp(ldt)} \]

Given this function weight is expected to decay and reach zero again at \(2 \times ldt\). 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

```r
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
**SSdlf**  

*self start for Declining Logistic Function*

**Description**

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

**Usage**

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

**Arguments**

- **time**: input vector (x) which is normally ‘time’, the smallest 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 been reached.
- **scal**: scale parameter which controls the spread also interpreted in terms of time to go from xmid to approx. 0.63 asym

**Examples**

```r
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)))
```
Details

Response function:

\[ y = \frac{(\text{asym} - a_2)}{1 + \exp((x_{mid} - \text{time})/\text{scal}))} + a_2 \]

- \text{asym}: upper asymptote
- \text{xmid}: time when \( y \) is midway between \( w \) and \( a \)
- \text{scal}: controls the spread
- \text{a2}: lower asymptote

The four parameter logistic \text{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
- \text{dlf}: vector of the same length as \( x \) (time) using the declining logistic function

Examples

```r
## Extended example in the vignette 'n1raa-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

```r
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 is the exponential function

\[ y = a \ast \exp(c \ast x) \]


### Value

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

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

### Examples

```r
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

```r
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 is the exponential-plateau function, where ‘xs’ is the break-point

\[(x < xs) \ast a \ast exp(c \ast x) + (x \geq xs) \ast (a \ast exp(c \ast xs))\]


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

```r
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

```r
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


The equation is:

\[(cm/rm) \ast \log(1 + \exp(rm \ast (t - tb)))\]

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

```r
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)
hil13(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:

- **hil1**: 
  \[ \frac{1}{1 + \left( \frac{Ka}{x} \right)} \]

- **hil2**: 
  \[ \frac{1}{1 + \left( \frac{Ka}{x} \right)^n} \]

- **hil3**: 
  \[ a \frac{1}{1 + \left( \frac{Ka}{x} \right)^n} \]

**Value**

- **hil1**: vector of the same length as x (time) using the Hill 1 function
- **hil2**: vector of the same length as x (time) using the Hill 2 function
- **hil3**: vector of the same length as x (time) using the Hill 3 function

**Note**

Zero values are not allowed.

**Examples**

```r
require(ggplot2)
## Example for hil1
set.seed(1234)
x <- 1:20
y <- hil1(x, 10) + rnorm(20, sd = 0.03)
dat1 <- data.frame(x = x, y = y)
fit1 <- nls(y ~ SShill1(x, Ka), data = dat1)
```
SSlinp

## Example for hill2
```r
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
```r
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
```r
linp(x, a, b, xs)
```

```r
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 $x < xs : (a + b \times x)$ and flat ($asymptote = a + b \times xs$) when $x \geq xs$.

### 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

```r
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**

```r
logis5(x, asym1, asym2, xmid, iscal, theta)
SSlogis5(x, asym1, asym2, xmid, iscal, theta)
```

**Arguments**

- `x`: input vector (x)
- `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 (only when theta = 1)
- `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**

The equation for this function is:

\[
 f(x) = asym2 + (asym1 - asym2)/(1 + exp(iscal * (log(x) - log(xmid))))^\theta
\]

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

```r
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, iscal, 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)
```

`SSmoh`  
*Self start for modified hyperbola (photosynthesis)*

Description

Self starter for modified Hyperbola with parameters: asymp, xmin and k

Usage

- `moh(x, asym, xmin, k)`
- `SSmoh(x, asym, xmin, k)`

Arguments

- x: input vector (x) which is normally a controlling variable such as nitrogen
- asym: asymptotic value when x tends to infinity
- xmin: value of x for which y equals zero
- k: curvature parameter

Details

This function is described in Archontoulis and Miguez (2015) - (doi:10.2134/agronj2012.0506). See Table S3 (Eq. 3.8)
**Value**

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

moh: vector of the same length as x (time) using the modified hyperbola

**Examples**

```r
require(ggplot2)
set.seed(1234)
x <- seq(3, 30)
y <- moh(x, 35, 3, 0.83) + rnorm(length(x), 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SS:moh(x, asym, xmin, k), data = dat)
## Visualize observed and simulated
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Testing predict function
prd <- predict_nls(fit, interval = "confidence")
datA <- cbind(dat, prd)
## Plotting
ggplot(data = datA, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit))) +
  geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5),
              fill = "purple", alpha = 0.3)

x <- seq(0, 20)
y <- moh(x, 30, 3, 0.9)
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**

```r
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

```r
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)
## Visualize observed and simulated
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit)))
## Testing predict function
prd <- predict_nls(fit, interval = "confidence")
datA <- cbind(dat, prd)
## Plotting
ggplot(data = datA, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit))) +
  geom_ribbon(aes(ymin = Q2.5, ymax = Q97.5),
  fill = "purple", alpha = 0.3)
x <- seq(0, 2000)
y <- nrh(x, 30, 0.04, 0.85, 2)
plot(x, y)
```
Description
Self starter for a plateau-exponential function

Usage
\[ \text{pexpf}(x, a, xs, c) \]
\[ \text{SSpexpf}(x, a, xs, c) \]

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: \( \text{for } x < xs: y = a \) \( \text{and } x \geq xs: a \times \exp(c \times (x - xs)) \).

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

Examples
```r
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

**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 \( x \geq xs \) : \( y = a + b \times (x - xs) \).

**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**

```r
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**

```r
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**


**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**

```r
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)
```
SSpquad3

self start for plateau-quadratic function

Description
Self starter for plateau-quadratic function with (three) parameters a (intercept), b (slope), c (quadratic)

Usage
pquad3(x, a, b, c)

SSpquad3(x, a, b, c)

Arguments
x input vector
a the intercept
b the slope
c quadratic term

Details

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

```r
require(ggplot2)
require(minpack.lm)
set.seed(123)
x <- 1:30
y <- pquad3(x, 20.5, 0.36, -0.012) + rnorm(30, 0, 0.3)
dat <- data.frame(x = x, y = y)
fit <- nlsLM(y ~ SSpquad3(x, a, b, c), data = dat)
## plot
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(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**

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

```r
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**


**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**

```r
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
```
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


Value

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

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)
```
SSquadp3  

**Description**

Self starter for quadratic plateau function with (three) parameters a (intercept), b (slope), c (quadratic)

**Usage**

quadp3(x, a, b, c)

SSquadp3(x, a, b, c)

**Arguments**

- **x**: input vector
- **a**: the intercept
- **b**: the slope
- **c**: quadratic term

**Details**

The equation is, for a response (y) and a predictor (x):

\[ y \begin{cases} \begin{align*} x & \leq xs \quad & (a + b \times x + c \times x^2) \\ x & > xs \quad & (a + (-b^2)/(4 \times c)) \end{align*} \end{cases} \]

where the break-point (xs) is \(-0.5\times b/c\)

and the asymptote is \((a + (-b^2)/(4 \times c))\)

**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

```r
require(ggplot2)
set.seed(123)
x <- 1:30
y <- quadp3(x, 5, 1.7, -0.04) + rnorm(30, 0, 0.6)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSquadp3(x, a, b, c), data = dat)
## 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

```r
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:

\[ a \times x^c / (1 + b \times x^d) \]

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](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](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

```
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

SSsharp

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

```r
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)))
```

SSsharp

**self start for temperature response**

Description

Self starter for temperature response function

Usage

```r
sharp(temp, r_tref, e, el, tl, eh, th, tref = 25)
SSsharp(temp, r_tref, e, el, tl, eh, th, tref = 25)
```

Arguments

- `temp`: input vector (x) which is normally 'temperature'.
- `r_tref`: rate at the standardised temperature, tref
- `e`: activation energy (eV)
- `el`: low temperature de-activation energy (eV)
- `tl`: temperature at which the enzyme is half active and half suppressed due to low temperatures
- `eh`: high temperature de-activation energy (eV)
- `th`: temperature at which enzyme is half active and half suppressed due to high temperatures
- `tref`: standardisation temperature in degrees centigrade. Temperature at which rates are not inactivated by either high or low temperatures. Typically, 25 degrees.
Details

Value
sharp: vector of the same length as x using a sharp function

Note
I do not recommend using this function.

Examples

```r
require(ggplot2)
require(minpack.lm)

temp <- 0:45
rate <- sharp(temp, 1, 0.03, 1.44, 28, 19, 44) + rnorm(length(temp), 0, 0.05)
dat <- data.frame(temp = temp, rate = rate)
## Fit model
fit <- nlsLM(rate ~ SSsharp(temp, r_tref, e, el, tl, eh, th, tref = 20), data = dat)
## Visualize
ggplot(data = dat, aes(temp, rate)) + geom_point() + geom_line(aes(y = fitted(fit)))
```

SSTemp3  

self start for Collatz temperature response

Description
Self starter for Collatz temperature response function

Usage

```r
temp3(x, t.m, t.l, t.h)
SSTemp3(x, t.m, t.l, t.h)
```

Arguments

- **x**: input vector (x) which is normally 'temperature'.
- **t.m**: medium temperature
- **t.l**: low temperature
- **t.h**: high temperature
SStrlin

Details


Value

temp3: vector of the same length as x using a temp function

Examples

```r
## A temperature response function
require(ggplot2)
set.seed(1234)
x <- 1:50
y <- temp3(x, 25, 13, 36) + rnorm(length(x), sd = 0.05)
dat1 <- data.frame(x = x, y = y)
fit1 <- nls(y ~ SStemp3(x, t.m, t.l, t.h), data = dat1)

ggplot(data = dat1, aes(x, y)) +
  geom_point() +
  geom_line(aes(y = fitted(fit1)))
```

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

```r
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

```r
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))
```

---

**summary_simulate**  
Summarize a matrix of simulations by their mean (median), sd (mad), and quantiles
Description

Utility function to summarize the output from `simulate` functions in this package

Usage

```r
summary_simulate(
  object,
  probs = c(0.025, 0.975),
  robust = FALSE,
  data,
  by,
  ...
)
```

Arguments

- **object**: nobs x nsim matrix where nobs are the number of observations in the dataset and nsim are the number of simulations.
- **probs**: the percentiles to be computed by the quantile function.
- **robust**: If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead.
- **data**: the original data.frame used to fit the model. A data.frame will be returned instead of a matrix in this case.
- **by**: optionally aggregate the results by some factor in the data.frame. It will be coerced to a formula. If robust is FALSE, the mean will be used. Otherwise, the median.
- **...**: additional arguments to be passed. (none used at the moment)

Value

By default it returns a matrix unless the data argument is present and then it will return a data.frame

Examples

```r
data(barley, package = "nlraa")
fit <- nls(yield ~ SSlinp(NF, a, b, xs), data = barley)
sim <- simulate_nls(fit, nsim = 100)
sims <- summary_simulate(sim)

## If we want to combine the data.frame
simd <- summary_simulate(sim, data = barley)

## If we also want to aggregate by nitrogen rate
simda <- summary_simulate(sim, data = barley, by = "NF")
```
swpg  

*Water limitations for Soybean growth*

**Description**

Simulated data based on observed 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**


**Source**

Simulated 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**

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

object object which inherits class \texttt{lm}, \texttt{gls} or \texttt{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)

data optional passing of ‘data’, probably needed when using this function inside other functions.

Details

Variance Covariance matrix for (non)linear mixed models

Value

It returns a \texttt{matrix} or a sparse matrix \texttt{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 \texttt{getVarCov} only works well for \texttt{lme} objects. The equivalence is more or less:

\begin{itemize}
  \item getVarCov type = “random.effects” equivalent to \texttt{var_cov} type = “random”.
  \item getVarCov type = “conditional” equivalent to \texttt{var_cov} type = “residual”.
  \item getVarCov type = “marginal” equivalent to \texttt{var_cov} type = “all”.
\end{itemize}

The difference is that getVarCov has an argument that specifies the individual for which the matrix is being retrieved and \texttt{var_cov} returns the full matrix only.

See Also

\texttt{getVarCov}

Examples

\begin{verbatim}
require(graphics)
require(nlme)
data(ChickWeight)
## First a linear model
flm <- \texttt{lm(weight \sim Time, data = ChickWeight)}
vlm <- \texttt{var_cov(flm)}
## First model with no modeling of the Variance-Covariance
fit0 <- \texttt{gls(weight \sim Time, data = ChickWeight)}
v0 <- \texttt{var_cov(fit0)}
## Only modeling the diagonal (weights)
fit1 <- \texttt{gls(weight \sim Time, data = ChickWeight, weights = varPower())}
v1 <- \texttt{var_cov(fit1)}
## Only the correlation structure is defined and there are no groups
\end{verbatim}
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(log(v4[,ncol(v4):1])))
Index

* datasets
  barley, 3
  fm1.P.at.x.0.4, 8
  fm1.P.bt, 9
  fm1.P.bt.ft, 9
  fm2.Lob.bt, 10
  fmm1.bt, 10
  lfmc, 13
  Lob.bt.pe, 14
  maizeleafext, 14
  nlraa.env, 15
  sm, 38
  swpg, 72

barley, 3
bell (SSbell), 39
beta5 (SSbeta5), 40
bg4rp (SSbg4rp), 41
bgf (SSbgf), 42
bgf2 (SSbgf2), 42
bgf4 (SSbgf4), 44
bgfrp (SSbgfrp), 45
blin (SSblin), 46
Boot, 5, 6, 8
boot, 4–6, 8
boot.gls (boot.lme), 5
boot.gnl (boot.nlme), 6
boot.lm, 4, 8
boot.lme, 5
boot.nlme, 6
boot.nls, 7
dlf (SSdlf), 47

expf (SSexpf), 48
expfp (SSexpfp), 49
explin (SSexplin), 50

fitted, 35
fm1.P.at.x.0.4, 8

fm1.P.bt, 9
fm1.P.bt.ft, 9
fm2.Lob.bt, 10
fmm1.bt, 10
gam, 23, 24, 27, 38
getVarCov, 73
glm, 27, 38
gls, 5, 21, 29, 33, 73
gnls, 6, 21, 30, 35

hill1 (SShill), 51
hill2 (SShill), 51
hill3 (SShill), 51
IA_tab, 11, 27
IC_tab, 12, 13
ICtab, 13

lfmc, 13
linp (SSlinp), 53
lm, 4, 28, 31, 32, 73
lme, 5, 21, 33, 73
Lob.bt.pe, 14
logis5 (SSlogis5), 54

maizeleafext, 14
Matrix, 73

moh (SSmoh), 55
mvnorm, 29, 31, 34, 36, 37

nlsList.formula, 16

nlsList, 16
nlsLM, 16
nlsLMList, 15
nlsLMList.formula, 16
nrh (SSnrh), 56

nlme, 6, 35, 36
nlraa.env, 15
nls, 7, 16, 17, 37
nlsList, 16, 17
nlsLM, 16
nlsLMList, 15
nlsLMList.formula, 16

75
INDEX

pexpf (SSpexpf), 58
plin (SSplin), 59
plot.IA_tab (IA_tab), 11
pquad (SSpquad), 60
pquad3 (SSpquad3), 61
predict, 28
predict.gam, 20, 24, 28
predict.gls, 29, 30
predict.gnls, 22, 31, 37, 38
predict.lm, 20, 24
predict.lme, 22, 34
predict.nlme, 22, 36
predict.nls, 18, 20, 24
predict2_gam (predict_nls), 23
predict2_nls, 17
predict.gam, 19
predict.gls (predict.nlme), 21
predict_gnls (predict.nlme), 21
predict.lme (predict.nlme), 21
predict.nlme, 21
predict.nls, 18, 19, 23, 38
profd (SSprofd), 62
quadp (SSquadp), 63
quadp3 (SSquadp3), 64

R2M, 25
ratio (SSratio), 65
ricker (SSricker), 66

sharp (SSsharp), 67
simulate, 28, 31, 32
simulate_gam, 20, 24, 27
simulate_gls, 5, 29
simulate_gnls, 6, 30, 35
simulate.lm, 4, 28, 31
simulate.lme, 5, 29, 30, 33
simulate.nlme, 34
simulate.nlme_one, 6, 35, 36
simulate.nls, 7, 20, 24, 37
sm, 38
SSbell, 39
SSbeta5, 40
SSbg4rp, 41
SSbgf, 42, 45
SSbgf4, 42, 44
SSbgpr, 44, 45
SSblin, 46
SSdf, 47

SSexpf, 48
SSexpfp, 49
SSexpf, 49
SSexpf, 50
SSfp1, 48
SShill, 51
SShill1 (SShill), 51
SShill2 (SShill), 51
SShill3 (SShill), 51
SSlinp, 53
SSlogis5, 54
SSmoh, 55
SSnrh, 56
SSpexpf, 58
SSplin, 59
SSquad, 60
SSquad3, 61
SSprofs, 62
SSquad, 63
SSquadp3, 64
SSratio, 65
SSricker, 66
SSsharp, 67
SStemp3, 68
SStrlin, 69
summary_simulate, 70
swpg, 72

temp3 (SStemp3), 68
strlin (SStrlin), 69

var_cov, 72
vcov, 29, 31, 34, 36, 37