Package 'statforbiology'

October 21, 2024

Type Package

Title Tools for Data Analyses in Biology

Version 0.9.9

Description Contains several tools for nonlinear regression analyses and general data analysis in biology and agriculture. Contains also datasets for practicing and teaching purposes. Supports the blog: Onofri (2024) ``Fixing the bridge between biologists and statisticians" <https://www.statforbiology.com> and the book: Onofri (2024) ``Experimental Methods in Agriculture" <https://www.statforbiology.com/_statbookeng/>. The blog is a collection of short articles aimed at improving the efficiency of communication between biologists and statisticians, as pointed out in Kozak (2016) <doi:10.1590/0103-9016-2015-0399>, spreading a better awareness of the potential usefulness, beauty and limitations of biostatistic.

License GPL-3

URL https://github.com/OnofriAndreaPG/statforbiology

BugReports https://github.com/OnofriAndreaPG/statforbiology/issues

Depends R (>= 3.5), drc

Imports ggplot2, tidyr, car, nlme, emmeans, methods, MASS, stats, drcte, multcomp, utils, multcompView

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

Author Andrea Onofri [aut, cre]

Maintainer Andrea Onofri <andrea.onofri@unipg.it>

Repository CRAN

Date/Publication 2024-10-21 12:00:08 UTC

Contents

AMMI *AMMI analysis for multienvironment genotype experiments*

Description

This function performs the AMMI (Addittive Main effects Multiplicative Interaction) analysis, according to Zobel et al (1988). The function has been described in Onofri and Ciriciofolo (2007).

Usage

```
AMMI(yield, genotype, environment, block = NULL, PC = 2,
     MSE = NULL, dfr = NULL)
```

Arguments

yield	a vector containing yield levels
genotype	a vector containing genotype codings
environment	a vector containing environment codings
block	a vector containing block codes for each environment
PC	the number of PCs that one wants to extract
MSE	Mean Squared Error
dfr	Residual Degrees of Freedom

Value

Returns a list of class 'AMMIobject' with the following components

genotype_means	The overall least squares genotype means
environment_means	The overall least squares environment means
interaction_means	The least squares means for the genotype by environment combinations
interaction_effect	a two-way table of interaction effects
additive_ANOVA	an ANOVA table for the additive model
mult_Interaction	an ANOVA table for multiplicative model
MSE	Mean Square Error
dfr	Degrees of freedom for the MSE
environment_scores	a table of environment scores
genotype_scores	a table of genotype scores
stability	AMMI stability value (ASV; Mohammadi and Amri, 2008)

Author(s)

Andrea Onofri

References

Mohammadi, R., Amri, A., 2008. Comparison of parametric and non-parametric methods for selecting stable and adapted durum wheat genotypes in varibale environments. Euphytica 159, 419–432.

Onofri, A., Ciriciofolo, E., 2007. Using R to perform the AMMI analysis on agriculture variety trials. R NEWS 7, 14–19.

Zobel, R. W., Wright, M.J., and Gauch, H. G., 1988. Statistical analysis of a yield trial. Agronomy Journal, 388-393.

Examples

```
WinterWheat <- getAgroData("WinterWheat")
tab <- with(WinterWheat, AMMI(Yield, Genotype, Year, Block, PC = 2))
tab
```

---

angularTransform *Angular transformation for percentages*

Description

Angular transformation for percentages

Usage

```
angularTransform(percentage)
```

Arguments

percentage	A number.

Value

A number.

Examples

```
angularTransform(25)
angularTransform(35.2)
```

anova.aovlist *Prints an ANOVA table for an 'aovList' object*

Description

A wrapper for the 'summary' method for objects of class 'aovlist'.

Usage

```
## S3 method for class 'aovlist'
anova(object, ...)
```

Arguments

object	An object of class "aovList"
...	Other additional arguments

Value

An object of class "summary.aovlist". It is a list (one object per error stratum) of ANOVA tables (class "anova") with a row for each term in the model, plus one for "Residuals" if there are any.

See Also

aov, summary, model.tables, TukeyHSD

Examples

```
# A split-plot design
data(oats)
oats.aov <- aov(Y ~ N*V + Error(B/V), data = oats)
anova(oats.aov)
emmeans::emmeans(oats.aov, ~N)
```

asymReg *Asymptotic functions*

Description

These functions provide the asymptotic regression model (asymReg), the asymptotic regression function with self-starter for the nls function and the asymptotic regression function with self-starter for the drm function in the drc package. The 'DRC.SSasymp()' function provides the self-starter for drc, to fit the same function as in the 'SSasymp()' function in the 'nlme' package. The asymptotic regression model is also known as the Mitscherlich law in agriculture and as the von Bertalanffy law in fisheries research.

Usage

```
asymReg.fun(predictor, init, m, plateau)
NLS.asymReg(predictor, init, m, plateau)
DRC.asymReg(fixed = c(NA, NA, NA), names = c("init", "m", "plateau"))
DRC.SSasymp(fixed = c(NA, NA, NA), names = c("Asym", "R0", "lrc"))
```

Arguments

predictor	a numeric vector of values at which to evaluate the model
init	model parameter
m	model parameter
plateau	model parameter
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The asymptotic model is given by the following function:

$$f(x) = \text{plateau} - (\text{plateau} - \text{init}) \cdot \exp(-mx)$$

A similar parameterisation where 'm' is replaced by exp(lrc) is provided in the 'nlme' package, with self-starter for the 'nls' function. Here, we provide the self-starter for the 'drm' function in the 'drc' package:

$$f(x) = \text{Asym} + (\text{R0} - \text{Asym}) \cdot \exp[-exp(lrc)x]$$

Value

asymReg.fun and NLS. asymReg return a numeric value, while DRC.asymReg and DRC.SSasymp return a list containing the nonlinear function, the self starter function and the parameter names.

Note

DRC.asymReg and DRC.SSasymp are for use with the function drm.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
X <- c(1, 3, 5, 7, 9, 11, 13, 20)
Y <- c(8.22, 14.0, 17.2, 16.9, 19.2, 19.6, 19.4, 19.6)

# nls fit
model <- nls(Y ~ NLS.asymReg(X, init, m, plateau) )

# drm fit
model <- drm(Y ~ X, fct = DRC.asymReg())
summary(model)

model2 <- drm(Y ~ X, fct = DRC.SSasymp())
summary(model2)
```

---

beetGrowth *Growth of sugarbeet in weed-infested and weed-free conditions*

Description

A data frame with 18 observations on the following 3 variables:

Usage

```
data("beetGrowth")
```

Format

A data frame with 18 rows and 3 variables

Details

- DAE. numeric: Days After Emergence
- weightInf. numeric: weight of crop biomass on the weed infested plots
- weightFree. numeric: weight of crop biomass on the weed free plots

References

Covarelli, G., Onofri, A., 1998. Effects of timing of weed removal and emergence in sugarbeet, in: Proceedings 6th EWRS Mediterranean Symposium, Montpellier, 13-15 May 1998. EWRS, pp. 65–72.

Examples

```
data(beetGrowth)
mod3 <- nls(weightInf ~ NLS.L3(DAE, b, c, d), data = beetGrowth)
```

`biplot.AMMIobj` *Biplots for AMMI and GGE analyses of multi-environment genotype experiments*

Description

This method is reserved for the classes 'AMMIobject' and 'GGEobject' as obtained from the 'AMMI()' and 'GGE()' functions in the 'aomisc' package. It draws swift biplots of two types: type 1 plots PC1 against the average yield of genotypes across environments, while type 2 plots PC2 against PC1. The former type of biplot is reserved for 'AMMI' objects, while the second one is for AMMI and GGE objects.

Usage

```
## S3 method for class 'AMMIobject'
biplot(x, biplot = 1, xlim=NULL, ylim=NULL, elabels=NULL,
               glabels=NULL, quad=FALSE, cexG=0.9,
               cexE=0.9,
               xlab=NULL, ylab=NULL, font=1, ...)
## S3 method for class 'GGEobject'
biplot(x, biplot = 1, xlim=NULL, ylim=NULL, elabels=NULL,
               glabels=NULL, quad=FALSE, cexG=0.9,
               cexE=0.9,
               xlab=NULL, ylab=NULL, font=1, ...)
```

Arguments

`x`	a 'AMMIobject' or 'GGEobject' objects
`biplot`	Numeric: either 1 or 2, to request on of the two available types of biplots (see description).
`xlim`	graphical parameter as in the 'plot' method
`ylim`	graphical parameter as in the 'plot' method
`elabels`	labels for the environments
`glabels`	labels for the genotypes
`quad`	logical. If TRUE, plots the axes
`cexG`	graphical parameter: the 'cex' for the genotype labels
`cexE`	graphical parameter: the 'cex' for the environment labels
`xlab`	graphical parameter as in the 'plot' method
`ylab`	graphical parameter as in the 'plot' method
`font`	graphical parameter as in the 'plot' method. Relating to the genotype labels
`...`	Other additional arguments

Value

It only returns a graph

Author(s)

Andrea Onofri

Examples

```
WinterWheat <- getAgroData("WinterWheat")
tab <- with(WinterWheat, AMMI(Yield, Genotype, Year, Block, PC = 2))
biplot(tab)
```

---

boxcox.nls *Transform-both-sides (TBS) method for nonlinear regression*

Description

Finds the optimal Box-Cox transformation for non-linear regression models.

Usage

```
  ## S3 method for class 'nls'
boxcox(object, lambda = seq(-2, 2, 1/10), plotit = TRUE,
             start, eps = 1/50, bcAdd = 0, level = 0.95,
             xlab = expression(lambda), ylab = "log-likelihood", ...)
  ## S3 method for class 'nlsbc'
summary(object, ...)
```

Arguments

object	object of class nls. For bcSummary the nls fit should have been obtained using boxcox.nls
lambda	numeric vector of lambda values; the default is (-2, 2) in steps of 0.1.
plotit	logical which controls whether the result should be plotted.
start	a list of starting values (optional).
eps	numeric value: the tolerance for lambda = 0; defaults to 0.02.
bcAdd	numeric value specifying the constant to be added on both sides prior to Box-Cox transformation. The default is 0.
level	numeric value: the confidence level required.
xlab	character string: the label on the x axis, defaults to "lambda".
ylab	character string: the label on the y axis, defaults to "log-likelihood".
...	additional graphical parameters.

Details

boxcox.nls is very similar to the boxcox in its arguments. The optimal lambda value is determined using a profile likelihood approach: For each lambda value the non-linear regression model is fitted and the lambda value resulting in thre largest value of the log likelihood function is picked. If a self starter model was used in the model fit, then gradient information will be used in the profiling.

Value

An object of class nls (returned invisibly). If plotit = TRUE a plot of loglik vs lambda is shown indicating a confidence interval (by default 95%) about the optimal lambda value.

Author(s)

Christian Ritz, modified by Andrea Onofri

References

Carroll, R. J. and Ruppert, D. (1988) *Transformation and Weighting in Regression*, New York: Chapman and Hall (Chapter 4).

See Also

For linear regression the analogue is boxcox.

Examples

```
## Fitting log-logistic model without transformation
ryegrass.m1 <- nls(rootl ~ NLS.L4(conc, b, c, d, e),
                   data=ryegrass)
summary(ryegrass.m1)

## Fitting the same model with optimal Box-Cox transformation
ryegrass.m2 <- boxcox(ryegrass.m1, plotit = TRUE)
summary(ryegrass.m2)
```

---

check.hom *Check linear models for homoscedasticity*

Description

This function takes a linear model object as an argument and checks whether the residuals are homoscedastic, in relation to a stratification variable or covariate, that is given as an argument.

Usage

```
check.hom(obj, var, alternative)
```

Arguments

obj	a linear model object fitted with lm()
var	a vector containing a stratification variable. If missing, the fitted values from obj are taken as a covariate
alternative	The null (homogeneous variances) is tested against the alternative, that is (i) different variances for each level of the stratification variable given as argument ("varIdent"), (ii) variance is a power function of a covariate, that is given as argument ("varPower"; the fitted values are taken as the covariate, in case the argument 'var' is missing and (iii) the variance is an exponential function of a covariate, that is given as argument ("varExp"; the fitted values are taken as the covariate, in case the argument 'var' is missing)

Details

The function fits a gls with same structure as the input model, together with a heteroscedastic gls, where residuals are allowed to be heteroscedastic, according to the 'alternative' argument. The two models are compared by a LRT

Value

LRT	the value of LRT
LRT	the P-value of LRT
aovtable	a summary table for the LRT
modHet	the gls object containing the heteroscedastic fit

Author(s)

Andrea Onofri

Examples

```
fileName <- "https://www.casaonofri.it/_datasets/FGP_rape.csv"
library(statforbiology)
dataset <- read.csv(fileName)
dataset[,1:5] <- lapply(dataset[,1:5], factor)
mod <- lm(FGP ~ Genotype * Run, data = dataset)
check <- check.hom(mod, Run)
check$aovtable
```

compCoefs *Pairwise comparisons of model parameters for nls objects*

Description

With models containing covariates and factors, we may be interested in fitting nonlinear regression models in a groupwise fashion, according to the levels of the experimental factor(s). This function permits pairwise comparison procedures of model parameters; parameters can be compared by means of either ratios or differences. It works very much like the function 'compParm' in the package 'drc', but it is to be used with 'nls' objects.

Usage

```
compCoefs(object, parameterNames, operator = "-", display = "pairwise",
      adjust = "holm", reversed = FALSE, level = 0.05, Letters = c(letters, LETTERS))
```

Arguments

object	an object of class 'nls'
parameterNames	The name of parameters to compare. It must corresponds to the names of parameters in coefs(object)
operator	a character. If equal to "/" parameter ratios are compared. If equal to "-" (default) parameter differences are compared.
display	a character. If equal to "pairwise" (default), pairwise comparisons are displayed. Otherwise, a compact letter display is returned, according to the argument in 'level'.
adjust	A multiplicity adjustment method as in p.adjust. Defaults to "holm".
reversed	logical. Should the order of means/letters be decreasing? Defaults to FALSE, which means that the means and letters are in increasing order.
level	Protection level for compact letter display
Letters	Vector of distinct characters (or character strings) used to connect levels that are not significantly different. They should be recogizable when concatenated. The default behaviour is to use the small letters, followed by the capital letters. See help for 'multcompView::multcompLetters()' for futher detail)

Value

returns a dataframe, containing the results

Author(s)

Andrea Onofri

Examples

```
data(metamitron)
modNlin <- nls(Conc ~ A[Herbicide] * exp(-k[Herbicide] * Time),
               start=list(A=rep(100, 4), k=rep(0.06, 4)),
               data=metamitron)
tab <- summary(modNlin)
pn <- c("k1", "k2", "k3", "k4")
compCoefs(modNlin, pn) # Holm multiplicity adjustment
compCoefs(modNlin, pn, adjust = "none")

# Displaying letters (P = 0.05 is default)
compCoefs(modNlin, pn, adjust = "none",
        display = "cld")

# Calculate ratios
compCoefs(modNlin, pn, adjust = "none",
        display = "pairwise", operator = "/")
```

`compCurves` *Compare regression curves in a pairwise fashion*

Description

For regression models containing grouping factors and fitted with the 'drm' function in the 'drc' package, this function compares the different curves for each factor level in a pairwise fashion, according to a series of F tests for the extra-sum-of-squares. Works only with 'drc' objects

Usage

```
compCurves(
  obj,
 adjusted = c("none", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr")
)
```

Arguments

`obj`	a drc object
`adjusted`	a character string, for selecting the method of multiplicity correction. Must be one of: c("none", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr")

Value

returns a list with two slots: (i) Pairs: the list of peirwise comparisons, with F values and P-values; (ii) Letters: letter display for the different curves

Author(s)

Andrea Onofri

Examples

```
metamitron <- getAgroData("metamitron")
head(metamitron)
tail(metamitron)

mod1 <- drm(Conc ~ Time, fct = DRC.expoDecay(),
               curveid = Herbicide,
               data = metamitron)
summary(mod1)
compCurves(mod1, adjusted = "bonferroni")
```

`contr.Tukey` *Pairwise contrast matrix*

Description

Returns a matrix of Tukey-type contrasts (all-pairwise)

Usage

```
contr.Tukey(n, names = NULL)
```

Arguments

`n`	an integer: the number of levels, corresponding to the number of columns in the output matrix
`names`	a vector of names with same length as n

Details

This functions is used for creating contrast matrices for use with glht or other methods for fitting linear contrasts. The rows of the resulting matrices contain the coefficients of contrasts relating to a factor with n levels. The names of the levels can be optionally given with the argument 'names', other wise they default to the numbers from 1 to n.

Value

A matrix with n columns and n(n-1)/2 rows

Author(s)

Andrea Onofri

Examples

```
contr.Tukey(8, LETTERS[1:8])
```

CVA *Canonical variate analysis for multienvironment and multitrait genotype experiments*

Description

This function performs canonical variate analysis as a descriptive visualisation tool. It is close to the 'lda()' function in the MASS package but it is not meant to be used for discriminant analyses.

Usage

```
CVA(dataset, groups, scale = TRUE, constraint = 3)
```

Arguments

dataset	dataset is a multidimensional matrix of observations
groups	groups is a vector coding for groupings
scale	whether the data needs to be standardised prior to analysis. Defaults to TRUE
constraint	It is the type of scaling for eigenvectors, so that canonical variates have: 1 = unit within-group standard deviations (most common); 2 = unit total standard deviations; 3 = unit within group norms; 4 = unit total norms. It defaults to 3

Details

More detail can be found in a blog page, at 'https://www.statforbiology.com/2023/stat_multivar_cva/'. Please, note that preliminary data transformations (e.g.: standardisation) are left to the user and must be performed prior to analyses (see example below).

Value

TOT	matrix of total variances-covariances
B	matrix of 'between-groups' variances-covariances
W	matrix of 'within-group' variances-covariances
B/W	matrix of W^{-1} B
eigenvalues	vector of eigenvalues
eigenvectors	matrix of eigenvectors
proportion	a vector containing the proportion of total discriminating ability captured by each canonical variate
correlation	vector of canonical correlations
squared.canonical.correlation	vector of squared canonical correlations
coefficients	matrix of canonical coefficients
scores	matrix of canonical scores

`centroids` matrix of scores for centroids

`total.structure` matrix of total canonical structure

`between.structure` matrix of between-groups canonical structure

`within.structure` matrix of within-groups canonical structure

`class.fun` matrix of classifications functions

`class.val` matrix of classification values

`within.structure` matrix of within-groups canonical structure

`class` vector of predicted classes

Author(s)

Andrea Onofri

References

https://www.statforbiology.com/2023/stat_multivar_cva/

Examples

```
dataset <- getAgroData("WheatQuality4years")
dataset$Year <- factor(dataset$Year)
head(dataset)

# Standardise the data
groups <- dataset$Genotype
Z <- apply(dataset[,3:6], 2, scale, center = TRUE, scale = TRUE)
head(Z)

# Performs CVA
cvaobj <- CVA(Z, groups)
cvaobj
```

---

degradation *Soil degradation kinetic for a herbicide*

Description

A data frame with 24 observations on the following 2 variables:

Usage

```
data("degradation")
```

Format

A data frame with 24 rows and 2 variables

Details

- Time. numeric: Days from start of incubation
- Conc. numeric: residual concentration

Examples

```
data("degradation")
degradation
```

---

deviance.drc *Residual deviance for a non-linear regression fit.*

Description

Calculate the sum of squared residuals from a non-linear regression fit with the 'drm()' function in the 'drc()' package.

Usage

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

Arguments

object	a 'drc' object, for which the deviance is required
...	Other additional arguments, if necessary

Value

The value of the deviance extracted from the object object.

Author(s)

Andrea Onofri

Examples

```
X <- c(1, 3, 5, 7, 9, 11, 13, 20)
Y <- c(8.22, 14.0, 17.2, 16.9, 19.2, 19.6, 19.4, 19.6)

# nls fit
model <- nls(Y ~ NLS.asymReg(X, init, m, plateau) )
deviance(model)

# drm fit
model2 <- drm(Y ~ X, fct = DRC.asymReg())
deviance(model2)
```

`expoDecay` *Exponential decay function*

Description

These functions provide the exponential decay function (expoDecay), the exponential decay function with self-starter for the `nls` function and the exponential decay function with self-starter for the `drm` function in the drc package.

Usage

```
expoDecay.fun(predictor, C0, k)
NLS.expoDecay(predictor, C0, k)
DRC.expoDecay(fixed = c(NA, NA), names = c("C0", "k"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model.
`C0`	model parameter
`k`	model parameter
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The exponential decay is given by the following function:

$$f(x) = \mathrm{C0} \cdot \exp(-kx)\}$$

Value

expoDecay.fun and NLS.expoDecay return a numeric value, while DRC.expoDecay returns a list containing the nonlinear function, the self starter function and the parameter names.

Note

DRC.expoDecay is for use with the function drm.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
data("degradation")
model <- drm(Conc ~ Time, fct = DRC.expoDecay(),
             data = degradation)
summary(model)
model2 <- nls(Conc ~ NLS.expoDecay(Time, a, b),
              data = degradation)
summary(model2)
```

---

expoGrowth *Exponential growth function*

Description

These functions provide the exponential growth equation (expoGrowth), the exponential growth equation with self-starter for the nls function and the exponential growth equation with self-starter for the drm function in the drc package.

Usage

```
expoGrowth.fun(predictor, init, k)
NLS.expoGrowth(predictor, init, k)
DRC.expoGrowth(fixed = c(NA, NA), names = c("init", "k"))
```

Arguments

predictor	a numeric vector of values at which to evaluate the model.
init	model parameter: response at predictor = 0
k	model parameter: growth rate
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The exponential growth is given by the following function:

$$f(x) = \mathrm{C0} \cdot \exp(kx)\}$$

Value

expoGrowth.fun() and NLS.expoGrowth() return a numeric value, while DRC.expoGrowth() returns a list containing the nonlinear function, the self starter function and the parameter names.

Note

DRC.expoGrowth() is for use with the function drm.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
library(statforbiology)
Time <- 1:20
Resp <- c(0.18, 0.64, 1.14, 0.67, 0.32, 0.86, 0.70, 0.73, 0.89, 0.48,
    2.20, 1.03, 1.14, 2.14, 1.31, 2.08, 1.85, 1.47, 1.98, 1.30)
model <- drm(Resp ~ Time, fct = DRC.expoGrowth())
summary(model)
model2 <- nls(Resp ~ NLS.expoGrowth(Time, a, b))
summary(model2)
```

---

getAgroData *Get one of the available datasets*

Description

This function loads and returns a dataset available in an external repository and stored as '.csv' or other type of text files. all the examples.

Usage

```
getAgroData(
  fileName,
  where = "https://www.casaonofri.it/_datasets/",
  type = "csv"
)
```

Arguments

fileName	character: the name of the file (with no extension)
where	character: the name of the web repository
type	character: the extension of the web file

Value

returns a data.frame

Author(s)

Andrea Onofri

Examples

```
getAgroData("rimsulfuron")
```

---

getPlotData *Get the data for plotting with ggplot()*

Description

This method works on a model object and retrieves the data for plotting the observed values (average or all data) and predictions from model fit. It is mainly meant to be used with ggplot()

Usage

```
getPlotData(obj, ...)

## S3 method for class 'drc'
getPlotData(obj, xlim = NULL, gridsize = 100, type = c("average", "all"), ...)

## S3 method for class 'nls'
getPlotData(obj, xlim = NULL, gridsize = 100, type = c("average", "all"), ...)

## S3 method for class 'drcte'
getPlotData(
  obj,
```

```
  xlim = NULL,
  gridsize = 100,
  npmle.type = c("interpolation", "midpoint", "right", "left", "none"),
  ...
)
```

Arguments

obj	A fitted model object. Methods are provided for drc, drcte and nls objects
...	Other additional arguments.
xlim	a vector. The interval for the predictor in which predictions are to be obtained
gridsize	numeric. Number of points in the grid (within xlim) used for predictions.
type	a character string specifying whether all the observed points should be plotted (type = "all") or whether the response should be averaged over the levels of the predictor (type = "all"). It is disregarded for drcte objects
npmle.type	a character string. For drcte objects, the NPMLE of the cumulative density function is only specified at the end of each inspection interval, while it is not unique within each interval. This argument specifies how the CDF increases within each interval: possible values are "interpolation" (it is assumed that the CDF increases progressively), "left" (the CDF increases at the beginning of each interval), "right" (the CDF increases at the end of each interval; it is very common in survival analysis) and "midpoint" (the CDF increases in the middle of each interval; it is very common in survival analysis). This argument is neglected with nls and drc objects.

Value

This function returns a list of two elements: 'plotPoints' is a dataframe containing the observed data, 'plotFits' is a dataframe containing model predictions

Author(s)

Andrea Onofri

Examples

```
fileName <- "https://www.casaonofri.it/_datasets/metamitron.csv"
metamitron <- read.csv(fileName)
mod <- drm(Conc ~ Time, fct = EXD.2(),
           data = metamitron, curveid = Herbicide)
retVal <- getPlotData(mod)
library(ggplot2)
ggplot() +
  geom_point(data = retVal$plotPoints, mapping = aes(x = Time, y = Conc)) +
  geom_line(data = retVal$plotFits, mapping = aes(x = Time, y = Conc)) +
  facet_wrap(~ Herbicide)
```

GGE *GGE analysis for multienvironment genotype experiments*

Description

This function performs the GGE (Genotype plus Genotype by Environment interaction) analysis, according to Yan et al., 2000.

Usage

```
GGE(yield, genotype, environment, block, PC = 2)
```

Arguments

yield	a vector containing yield levels
genotype	a vector containing genotype codings
environment	a vector containing environment codings
block	a vector containing block codes for each environment
PC	the number of PCs that one wants to extract

Value

Returns a list of class 'GGEobject' with the following components

genotype_means	The overall least squares genotype means
environment_means	The overall least squares environment means
interaction_means	The least squares means for the genotype by environment combinations
ge_effect	a two-way table of 'interaction effects'gge' effects
additive_ANOVA	an ANOVA table for the additive model
GGE_summary	a summary table for GGE analysis
environment_scores	a table of environment scores
genotype_scores	a table of genotype scores

Author(s)

Andrea Onofri

References

Yan, W., Hunt, L.A., Sheng, Q., Szlavnics, Z., 2000. Cultivar Evaluation and Mega-Environment Investigation Based on the GGE Biplot. Crop Science 40, 597–605.

Examples

```
WinterWheat <- getAgroData("WinterWheat")
tab <- with(WinterWheat, GGE(Yield, Genotype, Year, Block, PC = 2))
tab
```

---

gnlht *Linear/nonlinear contrasts of model parameters*

Description

This function calculates linear/nonlinear contrasts of model parameters and returns their estimates with delta standard errors.

Usage

```
## S3 method for class 'numeric'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
## S3 method for class 'lm'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
## S3 method for class 'nls'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
## S3 method for class 'lme'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
## S3 method for class 'nlme'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
## S3 method for class 'drc'
gnlht(object, func, const, vcov.,
            parameterNames, dfr, ...)
```

Arguments

object	a named vector of parameter estimates, or a model object for which there are coef and vcov methods. The estimates are assumed as asymptotically normally distributed with covariance matrix returned by vcov. or passed as an argument
func	a list of functions or quoted strings that are the functions of the parameter estimates to be evaluated
const	If necessary, a dataframe whose columns are the constants to be used in the calculations above. For each row of this dataframe, the functions above are evaluated
vcov.	a variance-covariance matrix, or a function to calculate it from the model object
parameterNames	a character vector with namings for the parameters to be combined

dfr	number of degrees of freedom
...	Additional arguments

Details

Methods are given for several types of model fitting objects (lm, nls, lme, nlme, drc), from where model coefficients and a variance-covariance matrix are automatically retrieved. For other cases, a named vector of model parameters and a variance-covariance matrix can be provided as arguments to the 'gnlht()' function.

Value

Returns a data.frame

Author(s)

Andrea Onofri

Examples

```
data(metamitron)
#Fit nls grouped model
modNlin <- nls(Conc ~ A[Herbicide] * exp(-k[Herbicide] * Time),
               start=list(A=rep(100, 4), k=rep(0.06, 4)),
               data=metamitron)
summary(modNlin)


# Compare parameters
funList <- list(~k1 - k2, ~k1 - k3, ~k1 - k4)
gnlht(modNlin, funList)

# Combine parameters
funList <- list(~ -log(0.5)/k1, ~ -log(0.5)/k2,
                ~ -log(0.5)/k3, ~ -log(0.5)/k4)
gnlht(modNlin, funList)

# Combine more flexibly
funList <- list(~ -log(prop)/k1, ~ -log(prop)/k2,
                ~ -log(prop)/k3, ~ -log(prop)/k4)
gnlht(modNlin, funList, const = data.frame(prop = 0.5))

funList <- list(~ -log(prop)/k1, ~ -log(prop)/k2,
                ~ -log(prop)/k3, ~ -log(prop)/k4)
gnlht(modNlin, funList, const = data.frame(prop = c(0.7, 0.5, 0.3)))

# Other possibilities
funList <- list(~ (k2 - k1)/(k1 * k2) * log(prop),
                ~ (k3 - k1)/(k1 * k3) * log(prop),
                ~ (k4 - k1)/(k1 * k4) * log(prop))
gnlht(modNlin, funList, const = data.frame(prop = c(0.7, 0.5, 0.3)))
```

```
# Predictions
funList <- list(~ A1 * exp (- k1 * Time), ~ A2 * exp (- k2 * Time),
                ~ A3 * exp (- k3 * Time), ~ A4 * exp (- k4 * Time))
propdF <- data.frame(Time = seq(0, 67, 1))
func <- funList
const <- propdF
pred <- gnlht(modNlin, funList, const = propdF)
head(pred)
tail(pred)
```

`linear` *Simple linear regression functions*

Description

These functions provide the simple linear regression model (linear), the linear regression model with self-starter for the `nls` function (NLS.linear) and the simple linear regression function with self-starter for the `drm` function in the drc package (DRC.linear). For the 'nls' function, we also provide function and self starter for the simple linear regression through origin (NLS.linearOrigin). Obviously, fitting linear functions with nonlinear least square regression is sub-optimal, but it might be useful for comparing alternative models.

Usage

```
linear.fun(predictor, a, b)
NLS.linear(predictor, a, b)
NLS.linearOrigin(predictor, b)
DRC.linear(fixed = c(NA, NA), names = c("a", "b"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model.
`a`	numeric. The response when the predict is equal to 0.
`b`	numeric. The slope.
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The simple linear regression model is given by the following equation:

$$f(x) = a + bx$$

Value

linear.fun, NLS.linear and NLS.linearOrigin return a numeric value, while DRC.linear returns a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
# Simple linear regression
X <- seq(5, 50, 5)
Y <- 10 + 0.5*X + rnorm(10, 0, 0.5)

model1 <- nls(Y ~ NLS.linear(X, a, b))
model2 <- nls(Y ~ NLS.linearOrigin(X, b)) # force through origin
summary(model1); summary(model2)

model1 <- drm(Y ~ X, fct = DRC.linear())
model2 <- drm(Y ~ X, fct = DRC.linear(fixed = c(0, NA)))
summary(model1); summary(model2)
```

---

logCurve *Logarithmic curve*

Description

These functions provide the logarithmic model (logCurve) with self-starter for the `nls` function and for the `drm` function in the drc package.

Usage

```
logCurve.fun(predictor, a, b)
NLS.logCurve(predictor, a, b)
NLS.logCurveNI(predictor, b)
DRC.logCurve(fixed = c(NA, NA), names = c("a", "b"))
```

Arguments

predictor	a numeric vector of values at which to evaluate the model.
a	model parameter
b	model parameter
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The logarithmic curve is given by the following function:

$$f(x) = a + b\log(X)$$

This curve crosses the X axis at X = a. We can force it through the origin by setting a = 0; this is possible by setting 'fixed = c(=, NA), while, in the 'nls()' function, we need to use the NLS.logCurveNI()' function.

Value

logCurve.fun, NLS.logCurve and NLS.logCurveNI return a numeric value, while DRC.logCurve returns a list containing the nonlinear function, the self starter function and the parameter names.

Note

DRC.logCurve() is for use with the function drm.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
X <- c(1,2,4,5,7,12)
Y <- c(1.97, 2.32, 2.67, 2.71, 2.86, 3.09)

# lm fit
model <- lm(Y ~ log(X) )

# nls fit
model <- nls(Y ~ NLS.logCurve(X, a, b) )

# drm fit
model <- drm(Y ~ X, fct = DRC.logCurve() )
```

`ma` *Moving average for a vector*

Description

This is a wrapper for the 'filter()' function that uses the convolution method to calculate the moving averages of the terms in a vector, i.e. a series of averages of different subsets of the full vector

Usage

```
ma(x, n = 5, sides = 2)
```

Arguments

`x`	a numeric vector representing a time series
`n`	an integer, representing the number of values that compose each subset to be averaged
`sides`	can be either 1 or 2. If sides = 1 the n values to be averaged are taken before the preent values; If sides = 2 the n values are taken before and after the present value; If n is odd, (n - 1)/2 values are taken before the present value and (n - 1)/2 are taken after the present value, while, if n is even, more values are taken after the present value

Value

This function returns a vector of moving averages

Author(s)

Andrea Onofri

Examples

```
series <- c(319, 317, 332, 271, 301, 292, 351, 358, 259, 270)
ma(series, n = 4, sides = 2)
```

`metamitron` *Degradation of metamitron in soil with co-applied herbicides*

Description

The dataset describes the degradation of metamitron in soil at 20°C with several co-applied herbicides. It is a synthetic dataset, that was generated by Monte Carlo methods, starting from the observed data in Vischetti et al., 1996. A data frame with 96 observations on the following 3 variables:

Usage

```
data("metamitron")
```

Format

A data frame with 96 rows and 3 variables

Details

- Time. numeric: Days from start of incubation
- Herbicide. factor: M is 'metamitron', MP is 'metamitron+phenmedipham', MC is 'metamitron+chloridazon' and MPC is 'metamitron+phenmedipham+chloridazon'
- Conc. numeric: residual concentration of metamitron

References

Vischetti, C., Marini, M., Businelli, M., Onofri, A., 1996. The effect of temperature and co-applied herbicides on the degradation rate of phenmedipham, chloridazon and metamitron in a clay loam soil in the laboratory, in: Re, A.D., Capri, E., Evans, S.P., Trevisan, M. (Eds.), "The Environmental Phate of Xenobiotics", Proceedings X Symposium on Pesticide Chemistry, Piacenza. La Goliardica Pavese, Piacenza, pp. 287–294.

Examples

```
data("metamitron")
metamitron
```

mixture *Efficacy of the mixture of two herbicides*

Description

Two herbicides are used alone and in mixture. It is a data frame with 16 observations on the following 2 variables:

Usage

```
data("mixture")
```

Format

A data frame with 24 rows and 2 variables

Details

- Treat. Factor: the treatment levels
- Weight. numeric: the weight of weeds

Examples

```
data("degradation")
degradation
```

negExp *Negative exponential functions*

Description

These functions provide the negative exponential model (negExp.fun) with the related self-starters for the nls function (NLS.negExp) drm function in the 'drc' package (DRC.negExp) and the exponential cumulative distribution function (negExpDist.fun), with self-starters for both 'nls' (NLS.negExpDist) and 'drc' (DRC.negExpDist).

Usage

```
negExp.fun(predictor, a, c)
negExpDist.fun(predictor, c)
NLS.negExp(predictor, a, c)
DRC.negExp(fixed = c(NA, NA), names = c("a", "c"))
NLS.negExpDist(predictor, c)
DRC.negExpDist(fixed = NA, names = c("c"))
```

Arguments

predictor	a numeric vector of values at which to evaluate the model.
a	a numeric parameter representing the higher asymptote
c	a numeric parameter that is proportional to the relative rate of increase of the fitted function
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The negative exponential model is given by the following function:

$$f(x) = \text{a}\{1 - \exp[-\exp(cx)]\}$$

while the exponential CDF is obtained by setting a = 1:

$$f(x) = 1 - \exp[-\exp(cx)]$$

The 'drc' package contains also the function AR.2(), where c is replaced by e = 1/c. The 'nlme' package also contains an alternative parameterisation named 'SSasympOrig()', where c is replaced by phi3 = log(c).

Value

negExp.fun and negExpDist.fun return a numeric value, while the self-starters return a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc. Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
X <- c(1, 3, 5, 7, 9, 11, 13, 20)
Y <- c(4.5, 12.0, 16.1, 16.4, 18.9, 19.5, 19.3, 19.6)
model <- drm(Y ~ X, fct = DRC.negExp())
summary(model)
```

pairComp *Pairwise comparisons between the numeric elements of a vector*

Description

This function provides pairwise comparisons between the element of a vector, as long as a variance-covariance matrix is also provided

Usage

```
pairComp(parm, vcovMat, nams = NULL, dfr = NULL, adjust = "none",
         level = 0.05, Letters = c(letters, LETTERS, "."),
         reversed = FALSE)
```

Arguments

parm	A (possibly named) vector of estimates
vcovMat	Variance-covariance matrix for the estimates
nams	A character vector of parameter names (optional). If it is not provided and if 'parm' is not a named vectors, numbers '1:length(parm)' are used.
dfr	An optional number of residual degrees of freedom (defaults to Inf)
adjust	A multiplicity adjustment method as in 'multcomp'. Defaults to "none".
level	Protection level for compact letter display
Letters	Vector of distinct characters (or character strings) used to connect levels that are not significantly different. They should be recogizable when concatenated. The default behaviour is to use the small letters, followed by the capital letters. See help for 'multcompView::multcompLetters()' for futher detail)
reversed	logical. Should the order of means/letters be decreasing? Defaults to FALSE, which means that the means and letters are in increasing order)

Value

Returns a list with the following elements

Pairs	A dataframe of pairwise comparisons
Letters	A dataframe with compact letter display

Author(s)

Andrea Onofri

References

Onofri A. (2020) The broken bridge between biologists and statisticians: a blog and R package, Statforbiology, IT, web: https://www.statforbiology.com

Examples

```
# library(devtools)
# install_github("OnofriAndreaPG/aomisc")
library(statforbiology)
data(metamitron)

#Fit nls grouped model
modNlin <- nls(Conc ~ A[Herbicide] * exp(-k[Herbicide] * Time),
               start=list(A=rep(100, 4), k=rep(0.06, 4)),
               data=metamitron)
tab <- summary(modNlin)
tab

# Retreive infos and make comparisons
coefs <- coef(modNlin)[5:8]
vcovMat <- vcov(modNlin)[5:8, 5:8]
cp <- pairComp(coefs, vcovMat, dfr = tab$df[2],
               adjust = "none", reversed = FALSE)
cp$Letters
cp$pairs
```

`plotnls` *Plotting diagnostics for an* `nls` *object*

Description

This function is aimed at providing some types of plots to assess the goodness of fit for the selected model. Three plots (selectable by the argument 'which') are currently available: a plot of residuals against fitted values (which = 1), a Normal Q-Q plot (which = 2) and a plot of predicted against expected (line) and observed (symbols). By default, type = 3 is provided. As for the third graph, we can either plot all the data (type= "all") or the group means (type = "means"; the default)

Usage

```
plotnls(x, type = c("average", "all"),
                   xlim = NULL, gridsize = 100,
                   which = 3, ...)
```

Arguments

`x`	an object of class 'nls'
`type`	it can be either "means" or "all". In the first case, the group means are plotted for the third graph. It is only considered when which = 3
`xlim`	The limits for the x-axis (x1, x2)
`gridsize`	For 'which = 3', it sets the resolution of the fitted line
`which`	The type of graph: can be 1, 2 or 3 (see description). It defaults to 3.
`...`	additional graphical arguments

Details

It mimicks the behaviour of the function plot.lm()

Value

No return value, it produces a plot

Author(s)

Andrea Onofri

Examples

```
library(statforbiology)
degradation <- read.csv("https://www.casaonofri.it/_datasets/degradation.csv")
mod <- nls(Conc ~ A*exp(-k*Time),
               start=list(A=100, k=0.05),
               data=degradation)
plotnls(mod, which = 3)
```

`poly2` *Simple polynomial regression functions*

Description

These functions provide the simple polynomial (second order) regression model (poly2), the polynomial regression model with self-starter for the `nls` function (NLS.poly2) and the polynomial regression function with self-starter for the `drm` function in the drc package (DRC.poly2). Fitting linear functions with nonlinear least square regression is sub-optimal, but it might be useful for comparing alternative models.

Usage

```
poly2.fun(predictor, a, b, c)
NLS.poly2(predictor, a, b, c)
DRC.poly2(fixed = c(NA, NA, NA), names = c("a", "b", "c"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`a`	numeric. The response when the predictor is equal to 0.
`b`	numeric. The slope at X = 0
`c`	numeric. Regression parameter
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	a vector of character strings giving the names of the parameters. The default is reasonable.

Details

The simple polynomial (second order) regression model is given by the following equation:

$$f(x) = a + bx + cx^2$$

Value

poly2.fun and NLS.poly2 return a numeric value, while DRC.poly2 returns a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
# Polynomial regression
X <- seq(5, 50, 5)
Y <- c(12.6, 74.1, 157.6, 225.5, 303.4, 462.8,
       669.9, 805.3, 964.2, 1169)

model <- nls(Y ~ NLS.poly2(X, a, b, c))
summary(model)
model <- drm(Y ~ X, fct = DRC.poly2())
summary(model)
```

R2nls *Goodness of fit for nonlinear regression*

Description

This function calculates measures of goodness of fit for nonlinear regression. It works with both 'nls' and 'drc' objects

Usage

```
R2nls(object)
```

Arguments

object	A nonlinear regression fit object. It can be either a 'nls' fit or 'drm' fit.

Value

A list with the following slots:

R2	Traditional coefficient of determination, calculated as the ratio of model SS to total SS. Formula as in Schabenberger and Pierce, 5.23, pag 211.
PseudoR2	Pseudo-R2, more useful for nonlinear regression with no-intercept-models. Formula Formula as in Schabenberger and Pierce, 5.24, pag 212.
R2adj	Adjusted R2, similar to R2 above, but penalised for higher number of parameters.
MSE	Mean Squared Error
RMSE	Root Means Squared Error
RRMSE	Relative Root Means Squared Error

Author(s)

Andrea Onofri

References

Schabenberger, O., Pierce, F.J., 2002. Contemporary statistical models for the plant and soil sciences. Taylor & Francis, CRC Press, Books.

Examples

```
data(beetGrowth)
mod3 <- nls(weightInf ~ NLS.L3(DAE, b, c, d), data = beetGrowth)
R2nls(mod3)

mod4 <- drm(weightInf ~ DAE, fct = L.3(), data = beetGrowth)
R2nls(mod4)
```

SSbeta *Beta equation*

Description

These functions provide the beta equation, a threshold model that was derived from the beta density function and it was adapted to describe phenomena taking place only within a minimum and a maximum threshold value (threshold model), for example to describe the germination rate (GR, i.e. the inverse of germination time) as a function of temperature. These functions provide the beta equation (beta.fun), the self-starters for the nls function (NLS.beta) and the self-starters for the drm function in the drc package (DRC.beta)

Usage

```
beta.fun(X, b, d, Xb, Xo, Xc)
NLS.beta(X, b, d, Xb, Xo, Xc)
DRC.beta()
```

Arguments

X	a numeric vector of values at which to evaluate the model
b	model parameter
d	model parameter
Xb	model parameter (base threshold level)
Xo	model parameter (optimal threshold level)
Xc	model parameter (ceiling threshold level)

Details

This equation is parameterised as:

$$f(x) = max\left(d\left\{\left(\frac{X-Xb}{Xo-Xb}\right)\left(\frac{Xc-X}{Xc-Xo}\right)^{\frac{Xc-Xo}{Xo-Xb}}\right\}^{b}, 0\right)$$

It depicts a curve that is equal to 0 for X < Xb, grows up to a maximum, that is attained at X = Xo and decreases down to 0, that is attained at X = Xc and mantained for X > Xc.

Value

beta.fun, NLS.beta return a numeric value, while DRC.beta returns a list containing the nonlinear function and the self starter function

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
X <- c(1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50)
Y <- c(0, 0, 0, 7.7, 12.3, 19.7, 22.4, 20.3, 6.6, 0, 0)

model <- nls(Y ~ NLS.beta(X, b, d, Xb, Xo, Xc))
summary(model)
modelb <- drm(Y ~ X, fct = DRC.beta())
summary(modelb)
plot(modelb, log = "")
```

SSbragg *Bragg's Equation*

Description

These functions provide the Bragg's equations, that is based on the normal (Gaussian) distribution and it supports a maximum, a minimum and inflection points. These functions provide the equations with 4 (bragg.4.fun) and 3 (bragg.3.fun) parameters with self-starters for the `nls` function (NLS.bragg.4, NLS.bragg.3) and the self-starters for the `drm` function in the drc package (DRC.bragg.4, DRC.bragg.3)

Usage

```
bragg.4.fun(X, b, c, d, e)
bragg.3.fun(X, b, d, e)
NLS.bragg.4(X, b, c, d, e)
NLS.bragg.3(X, b, d, e)
DRC.bragg.4()
DRC.bragg.3()
```

Arguments

X	a numeric vector of values at which to evaluate the model
b	model parameter (relates to slope at inflection point)
d	model parameter (maximum value)
e	model parameter (abscissa at maximum value)
c	model parameter (lower asymptote)

Details

The Bragg's equation is parameterised as:

$$f(x) = c + (d - c)\ \exp(-b \cdot (X - e)^2)$$

for the 4-parameter model. For the 3-parameter model, c is equal to 0. It depicts a bell-shaped curve

Value

bragg.4.fun, bragg.3.fun, NLS.bragg.4 and NLS.bragg.3 return a numeric value, while DRC.bragg.4 and DRC.bragg.3 return a list containing the nonlinear function and the self starter function

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
library(statforbiology)
X <- c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50)
Y1 <- c(0.1, 2, 5.7, 9.3, 19.7, 28.4, 20.3, 6.6, 1.3, 0.1)
Y2 <- Y1 + 2

# nls fit
mod.nls <- nls(Y1 ~ NLS.bragg.3(X, b, d, e) )
mod.nls2 <- nls(Y2 ~ NLS.bragg.4(X, b, c, d, e) )

# drm fit
mod.drc <- drm(Y1 ~ X, fct = DRC.bragg.3() )
mod.drc2 <- drm(Y2 ~ X, fct = DRC.bragg.4() )
plot(mod.drc, ylim = c(0, 30), log = "")
plot(mod.drc2, add = TRUE, col = "red")
```

SScousens85 *Rectangular hyperbola for yield/weed density relationship*

Description

These functions provide the rectangula hyperbola that was devided by Cousens (1985) for modelling the relationship between crop yield and weed density. The function was derived from the yield-loss function, and contains parameters that are revelant for competition studies. These functions provide the equation (cousens85.fun), the self-starters for the `nls` function (NLS.cousens85) and the self-starters for the `drm` function in the drc package (DRC.cousens85)

Usage

```
cousens85.fun(predictor, Ywf, i, A)
NLS.cousens85(predictor, Ywf, i, A)
DRC.cousens85(fixed = c(NA, NA, NA), names = c("Ywf", "i", "A"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`Ywf`	model parameter (Weed-free yield)
`i`	model parameter (initial slope)

A	model parameter (maximum percentage yield loss)
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is usually reasonable.

Details

This equation is parameterised as:

$$f(x) = Ywf \frac{(1 - (ipredictor)}{(100\,(1 + i\,predictor/A)))}$$

It depicts a decreasing curve with no inflection point. The curve is equal to 'Ywf' when x = 0 and the lower asymptote is at 'A' multiplied by 'Ywf/100'

Value

cousens85.fun, NLS.cousens85 return a numeric value, while DRC.cousens85 return a list containing the nonlinear function and the self starter function

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Cousens, R., 1985. A simple model relating yield loss to weed density. Annals of Applied Biology 107, 239–252. https://doi.org/10.1111/j.1744-7348.1985.tb01567.x

Examples

```
library(statforbiology)
dataset <- getAgroData("Sinapis")

# nls fit
mod.nls <- nls(yield ~ NLS.cousens85(density, Ywf, i, A),
               data = dataset )
summary(mod.nls)
mod.nls2 <- drm(yield ~ density, fct = DRC.cousens85(), data = dataset )
summary(mod.nls2)
plot(mod.nls2)
```

SSE *Modified Gompertz equations*

Description

These functions provide the modified Gompertz equations with 4 (E4.fun), 3 (E3.fun) and 2 (E2.fun) parameters with self-starter for the `nls` function (NLS.E4, NLS.E3 and NLS.E2) and for the `drm` function in the 'drc' package (DRC.E4, DRC.E3 and DRC.E2).

Usage

```
E4.fun(predictor, b, c, d, e)
E3.fun(predictor, b, d, e)
E2.fun(predictor, b, e)
NLS.E4(predictor, b, c, d, e)
NLS.E3(predictor, b, d, e)
NLS.E2(predictor, b, e)
DRC.E4(fixed = c(NA, NA, NA, NA), names = c("b", "c", "d", "e"))
DRC.E3(fixed = c(NA, NA, NA), names = c("b", "d", "e"))
DRC.E2(fixed = c(NA, NA), names = c("b", "e"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`b`	model parameter (slope at inflection point)
`c`	model parameter (lower asymptote)
`d`	model parameter (higher asymptote)
`e`	model parameter (abscissa at inflection point)
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	names. A vector of character strings giving the names of the parameters. The default is reasonable.

Details

The modified Gompertz equation is parameterised as:

$$f(x) = c + (d - c)\,(1 - \exp\,[-exp(b(x - e)))]$$

It is a sygmoidally shaped curve and it is asymmetric about its inflection point, but the type of asymmetry is different from the Gompertz equation. For the 3- and 2-parameter model c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

E4.fun, E3.fun, E2.fun, NLS.E4, NLS.E3 and NLS.E2 return a numeric value, while DRC.E4, DRC.E3 and DRC.E2 return a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

Examples

```
data(beetGrowth)
mod3 <- nls(weightInf ~ NLS.E3(DAE, b, c, d), data = beetGrowth)
summary(mod3)
plot(mod3)
```

SSGompertz *Gompertz equations*

Description

These functions provide the Gompertz equations with 4 (G4.fun), 3 (G3.fun) and 2 (G2.fun) parameters with self-starter for the `nls` function (NLS.G4, NLS.G3 and NLS.G2).

Usage

```
G4.fun(predictor, b, c, d, e)
G3.fun(predictor, b, d, e)
G2.fun(predictor, b, e)
NLS.G4(predictor, b, c, d, e)
NLS.G3(predictor, b, d, e)
NLS.G2(predictor, b, e)
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`b`	model parameter (slope at inflection point)
`c`	model parameter (lower asymptote)
`d`	model parameter (higher asymptote)
`e`	model parameter (abscissa at inflection point)

Details

The Gompertz equation is parameterised as:

$$f(x) = c + (d - c)\ \exp\left[-exp(-b(x - e))\right]$$

It is a sygmoidally shaped curve and it is asymmetric about its inflection point. For the 3- and 2-parameter model c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

All these functions return a numeric value.

Author(s)

Andrea Onofri

Examples

```
data(beetGrowth)
mod3 <- nls(weightInf ~ NLS.G3(DAE, b, c, d), data = beetGrowth)
summary(mod3)
plot(mod3)
```

SSL *Logistic equations*

Description

These functions provide the logistic equations with 4 (L4.fun), 3 (L3.fun) and 2 (L2.fun) parameters with self-starter for the `nls` function (NLS.L4, NLS.L3 and NLS.L2) and the self-starter for logistic function with two parameters for the `drm` function in the drc package (DRC.L2).

Usage

```
L4.fun(predictor, b, c, d, e)
L3.fun(predictor, b, d, e)
L2.fun(predictor, b, e)
NLS.L4(predictor, b, c, d, e)
NLS.L3(predictor, b, d, e)
NLS.L2(predictor, b, e)
DRC.L2(upper = 1, fixed = c(NA, NA), names = c("b", "e"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`b`	model parameter (slope at inflection point)
`c`	model parameter (lower asymptote)
`d`	model parameter (higher asymptote)
`e`	model parameter (abscissa at inflection point)
`upper`	numeric. For L.2, a upper asymptote different from 1 can be specified.
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	names. A vector of character strings giving the names of the parameters. The default is reasonable.

Details

The logistic equation is parameterised as:

$$f(x) = c + \frac{d - c}{1 + exp\,[-b(x - e)]}$$

for the 3- and 2-parameter model c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

L4.fun, L3.fun, L2.fun, NLS.L4, NLS.L3 and NLS.L2 return a numeric value, while DRC.L2 returns a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

Examples

```
data(beetGrowth)
mod3 <- nls(weightInf ~ NLS.L3(DAE, b, c, d), data = beetGrowth)
mod3b <- drm(weightInf ~ DAE, fct=DRC.L2(upper = 25), data = beetGrowth)
```

SSLL *Log-logistic equation*

Description

These functions provide the loglogistic equation, that has a symmetric sygmoidal shape over the logarithm of time and it has been used for bioassay work. These functions provide the 4-, 3- and 2-parameter equations (LL4.fun(), LL3.fun() and LL2.fun()) as well as the self-starters for the `nls` function (NLS.LL4(), NLS.LL3() and NLS.LL2())

Usage

```
LL4.fun(predictor, b, c, d, e)
LL3.fun(predictor, b, d, e)
LL2.fun(predictor, b, e)
NLS.LL4(predictor, b, c, d, e)
NLS.LL3(predictor, b, d, e)
NLS.LL2(predictor, b, e)
```

Arguments

predictor	a numeric vector of values at which to evaluate the model
b	model parameter (slope at inflection point)
c	model parameter (lower asymptote)
d	model parameter (higher asymptote)
e	model parameter (abscissa at inflection point)

Details

These functions provide the log-logistic equation for bioassay work This equation (4-parameters) is parameterised as:

$$f(x) = c + \frac{d - c}{\exp(1 + \exp(-b\,(\log(x) - \log(e))))}$$

For the 3- and 2-parameters model, c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

All these functions return a numeric value

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Ritz, C., Jensen, S.M., Gerhard, D., Streibig, J.C., 2019. Dose-response analysis using R, CRC Press. ed. USA.

Examples

```
dataset <- getAgroData("brassica")

model <- nls(FW ~ NLS.LL4(Dose, b, c, d, e), data = dataset)
model <- nls(FW ~ NLS.LL3(Dose, b, d, e), data = dataset)
model <- nls(FW/max(FW) ~ NLS.LL2(Dose, b, e), data = dataset)
summary(model)
```

SSlorentz *Lorentz equation*

Description

These functions provide the Lorentz equation with 3 and 4 parameters ('lorentz.3.fun()' and 'lorentz.4.fun()'), as well as the self-starters for the nls function ('NLS.lorentz.3()' and 'NLS.lorentz.4()') and for the drm function in the 'drc' package ('DRC.lorentz.3()' and 'DRC.lorentz.4()')

Usage

```
lorentz.3.fun(X, b, d, e)
lorentz.4.fun(X, b, c, d, e)
NLS.lorentz.3(X, b, d, e)
NLS.lorentz.4(X, b, c, d, e)
DRC.lorentz.3()
DRC.lorentz.4()
```

Arguments

X	a numeric vector of values at which to evaluate the model
b	model parameter
d	model parameter
e	model parameter
c	model parameter

Details

These functions provide the Lorentz equation, that is a bell-shaped curve similar to a gaussian density function. It is parameterised as:

$$f(x) = c + \frac{d - c}{(1 + b\,(X - e)^2)}$$

The parameter 'e' represents the abscissa of the maximum value, while c is the minimum (asymptotic) response value and d is the maximum response value. The parameter 'b' relates to the slope at inflection point. For the 3-parameters curve, c is equal to 0.

Value

lorentz.3.fun(), lorentz.4.fun(), NLS.lorentz.3() and NLS.lorentz.4() return a numeric value, while DRC.lorentz.3() and DRC.lorentz.4() returns a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
X <- c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50)
Y1 <- c(0.1, 2, 5.7, 9.3, 19.7, 28.4, 20.3, 6.6, 1.3, 0.1)
Y2 <- Y1 + 2

# nls fit
mod.nls <- nls(Y1 ~ NLS.lorentz.3(X, b, d, e) )
mod.nls2 <- nls(Y2 ~ NLS.lorentz.4(X, b, c, d, e) )

# drm fit
mod.drc <- drm(Y1 ~ X, fct = DRC.lorentz.3() )
mod.drc2 <- drm(Y2 ~ X, fct = DRC.lorentz.4() )
plot(mod.drc, ylim = c(0, 30), log = "")
plot(mod.drc2, add = TRUE, col = "red")
```

SSpowerCurve *Power curve equation*

Description

These functions provide the Power curve equation, that is also known as the Freundlich equation and it is very used in agricultural chemistry, e.g. to model the sorption of xenobiotics in soil. It is also used to model the number of plant species as a function of sampling area (Muller-Dumbois method). These functions provide the equation ('powerCurve.fun()') as well as the self-starters for the `nls` function ('NLS.powerCurve()') and for the `drm` function in the 'drc' package ('DRC.powerCurve()')

Usage

```
powerCurve.fun(predictor, a, b)
NLS.powerCurve(predictor, a, b)
DRC.powerCurve(fixed = c(NA, NA), names = c("a", "b"))
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`a`	model parameter
`b`	model parameter
`fixed`	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
`names`	names. A vector of character strings giving the names of the parameters. The default is reasonable.

Details

These functions provide the Power curve equation, that is parameterised as:

$$f(x) = a\, x^b$$

which is totally equivalent to an exponential curve on the logarithm of X:

$$f(x) = a\, \exp\left[b\, \log(x)\right]$$

We see that both parameters relate to the 'slope' of the curve and b dictates its shape. If 0 < b < 1, the response Y increases as X increases and the curve is convex up. If b < 0 the curve is concave up and Y decreases as X increases. Otherwise, if b > 1, the curve is concave up and Y increases as X increases.

Value

powerCurve.fun() and NLS.powerCurve() return a numeric value, while DRC.powerCurve() returns a list containing the nonlinear function, the self starter function and the parameter names.

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Examples

```
dataset <-getAgroData("speciesArea")

#nls fit
model <- nls(numSpecies ~ NLS.powerCurve(Area, a, b),
             data = dataset)
summary(model)
# drm fit
model <- drm(numSpecies ~ Area, fct = DRC.powerCurve(),
             data = dataset)
summary(model)
```

SSW1 *Weibull equation (Type I)*

Description

These functions provide the Weibull equation (type I), that has an asymmetric sygmoidal shape and it has been used for bioassay work. These functions provide the 4-, 3- and 2-parameter equations (W1.4.fun(), W1.3.fun() and W1.2.fun()) as well as the self-starters for the `nls` function (NLS.W1.4(), NLS.W1.3() and NLS.W1.2()

Usage

```
W1.4.fun(predictor, b, c, d, e)
W1.3.fun(predictor, b, d, e)
W1.2.fun(predictor, b, e)
NLS.W1.4(predictor, b, c, d, e)
NLS.W1.3(predictor, b, d, e)
NLS.W1.2(predictor, b, e)
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`b`	model parameter (slope at inflection point)
`c`	model parameter (lower asymptote)
`d`	model parameter (higher asymptote)
`e`	model parameter (abscissa at inlection point)

Details

These functions provide the Weibull (Type I) equation for bioassay work This equation (4-parameters) is parameterised as:

$$f(x) = c + (d - c)\exp(-\exp(-b\,(\log(x) - \log(e))))$$

For the 3- and 2-parameters model, c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

All these functions return a numeric value

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Ritz, C., Jensen, S.M., Gerhard, D., Streibig, J.C., 2019. Dose-response analysis using R, CRC Press. ed. USA.

Examples

```
library(statforbiology)
dataset <- getAgroData("brassica")
model <- nls(FW ~ NLS.W1.4(Dose, b, c, d, e), data = dataset)
model.2 <- nls(FW ~ NLS.W1.3(Dose, b, d, e), data = dataset)
model.3 <- nls(FW/max(FW) ~ NLS.W1.2(Dose, b, e), data = dataset)
summary(model)
```

SSW2 *Weibull equation (Type II)*

Description

These functions provide the Weibull equation (type II), that has an asymmetric sygmoidal shape and it has been used for bioassay work. These functions provide the 4-, 3- and 2-parameter equations (W2.4.fun(), W2.3.fun() and W2.2.fun()) as well as the self-starters for the `nls` function (NLS.W2.4(), NLS.W2.3() and NLS.W2.2()

Usage

```
W2.4.fun(predictor, b, c, d, e)
W2.3.fun(predictor, b, d, e)
W2.2.fun(predictor, b, e)
NLS.W2.4(predictor, b, c, d, e)
NLS.W2.3(predictor, b, d, e)
NLS.W2.2(predictor, b, e)
```

Arguments

`predictor`	a numeric vector of values at which to evaluate the model
`b`	model parameter (slope at inflection point)
`c`	model parameter (lower asymptote)
`d`	model parameter (higher asymptote)
`e`	model parameter (abscissa at inlection point)

Details

These functions provide the Weibull (Type I) equation for bioassay work This equation (4-parameters) is parameterised as:

$$f(x) = c + (d - c)(1 - \exp(-\exp(b\,(\log(x) - \log(e)))))$$

For the 3- and 2-parameters model, c is equal to 0, while for the 2-parameter model d is equal to 1.

Value

All these functions return a numeric value

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Ritz, C., Jensen, S.M., Gerhard, D., Streibig, J.C., 2019. Dose-response analysis using R, CRC Press. ed. USA.

Examples

```
library(statforbiology)
dataset <- getAgroData("brassica")
model <- nls(FW ~ NLS.W2.4(Dose, b, c, d, e), data = dataset)
model <- nls(FW ~ NLS.W2.3(Dose, b, d, e), data = dataset)
model <- nls(FW/max(FW) ~ NLS.W2.2(Dose, b, e), data = dataset)
summary(model)
```

SSYL *Yield loss equation (Rectangular hyperbola)*

Description

These functions provide the yield loss equation, based on a rectangular hyperbola, supporting a higher asymptote and no inflection points. These functions provide the equation (YL.fun), the equation with self-starters for the `nls` function (NLS.YL) and equation with self-starters for the `drm` function in the drc package (DRC.YL)

Usage

```
YL.fun(predictor, i, A)
NLS.YL(predictor, i, A)
DRC.YL(fixed = c(NA, NA), names = c("i", "A"))
```

Arguments

predictor	a numeric vector of values at which to evaluate the model
i	model parameter (initial slope)
A	model parameter (maximum percentage yield loss)
fixed	numeric vector. Specifies which parameters are fixed and at what value they are fixed. NAs for parameter that are not fixed.
names	a vector of character strings giving the names of the parameters. The default is usually reasonable.

Details

The Yield-loss equation is parameterised as:

$$f(x) = \frac{i\,x}{1 + (i\,x)/A}$$

, it is convex and asymptotically increasing, while the predictor increases. The response is zero when the predictor is also zero and it was mainly used to describe yield losses (in percentage) due to weed competition, expressed as plant density (Cousens, 1985)

Value

YL.fun and NLS.YL return a numeric value, while DRC.YL returns a list containing the nonlinear function and the self starter function

Author(s)

Andrea Onofri

References

Ratkowsky, DA (1990) Handbook of nonlinear regression models. New York (USA): Marcel Dekker Inc.

Onofri, A. (2020). A collection of self-starters for nonlinear regression in R. See: https://www.statforbiology.com/2020/stat_nls_usefulfunctions/

Cousens, R., 1985. A simple model relating yield loss to weed density. Annals of Applied Biology 107, 239–252. https://doi.org/10.1111/j.1744-7348.1985.tb01567.x

Examples

```
library(statforbiology)
WeedDens <- c(0, 5, 10, 20, 25)
YieldLoss <- c(0, 17.9, 21.5, 27.4, 29.5)

# nls fit
mod.nls <- nls(YieldLoss ~ NLS.YL(WeedDens, i, A) )
summary(mod.nls)
# drm fit
mod.drc <- drm(YieldLoss ~ WeedDens, fct = DRC.YL() )
summary(mod.drc)
```

Index