Package ‘IPEC’

June 27, 2020

Type Package
Title Root Mean Square Curvature Calculation
Version 0.1.3
Date 2020-06-26
Author Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li
Maintainer Peijian Shi <peijianshi@gmail.com>
Imports numDeriv (>= 2016.8-1), MASS
Description Calculates the RMS intrinsic and parameter-effects curvatures of a nonlinear regression model.
Depends R (>= 3.5.0)
License GPL (>= 2)
NeedsCompilation no
Repository CRAN
Date/Publication 2020-06-27 04:20:02 UTC

R topics documented:

aic ................................................................. 2
biasIPEC .......................................................... 4
bic ................................................................. 8
bootIPEC ......................................................... 10
curvIPEC ....................................................... 16
derivIPEC ....................................................... 24
fitIPEC ........................................................... 27
IPEC ............................................................. 31
isom ............................................................. 38
leaves ............................................................ 39
shoots ............................................................ 40
skewIPEC ....................................................... 41

Index 47
Description

Calculates the AIC value(s) of the object(s) obtained from using the `fitIPEC` function.

Usage

```r
aic( object, ... )
```

Arguments

- `object`: A fitted model object for which there exists the sample size (`n`), estimate(s) of model parameter(s) (`par`), and residual sum of squares (`RSS`)
- `...`: Optionally more fitted model objects

Details

\[ \text{AIC} = 2p - 2\ln(L) \]

where \( p \) represents the number of model parameter(s) plus 1 for the error, and \( \ln(L) \) represents the maximum log-likelihood of the estimated model (Spiess and Neumeyer, 2010).

Value

There is an AIC value corresponding to one object, and there is a vector of AIC values corresponding to the multiple objects.

Note

With the sample size increasing, the number of model parameter(s) has a weaker influence on the value of AIC assuming that \( \ln(RSS/n) \) is a constant.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also

`bic`, `AIC` in package `stats`, and `BIC` in package `stats`
Examples

```r
# Choose a geographical population (see Table S1 in Wang et al. [2018] for details)
# 1: AJ; 2: HN; 3: HW; 4: HZ; 5: JD;
# 6: JS; 7: SC; 8: TC; 9: TT; 10: TX
ind <- 1
L <- Length[PopuCode == ind]
W <- Width[PopuCode == ind]
A <- Area[PopuCode == ind]

# Define a model y = a*(x1*x2), where a is a parameter to be estimated
propor <- function(theta, x){
a <- theta[1]
x1 <- x[,1]
x2 <- x[,2]
a*x1*x2
}

# Define a model y = a*(x1^b)*(x2^c), where a, b and c are parameters to be estimated
threepar <- function(theta, x){
a <- theta[1]
b <- theta[2]
c <- theta[3]
x1 <- x[,1]
x2 <- x[,2]
a*x1^b*x2^c
}

# Define a model y = a*x^b, where a and b are parameters to be estimated
twopar <- function(theta, x){
a <- theta[1]
b <- theta[2]
a*x^b
}

## Not run:
A1 <- fitIPEC(propor, x=cbind(L, W), y=A, fig.opt=FALSE, ini.val=list(seq(0.1, 1.5, by=0.1)))
B1 <- curvIPEC(propor, theta=A1$par, x=cbind(L, W), y=A)
A2 <- fitIPEC(threepar, x=cbind(L, W), y=A, fig.opt=FALSE, ini.val=list(A1$par, seq(0.5, 1.5, by=0.1), seq(0.5, 1.5, by=0.1)))
B2 <- curvIPEC(threepar, theta=A2$par, x=cbind(L, W), y=A)
A3 <- fitIPEC(twopar, x=L, y=A, fig.opt=FALSE, ini.val=list(1, seq(0.5, 1.5, by=0.05)))
B3 <- curvIPEC(twopar, theta=A3$par, x=L, y=A)
A4 <- fitIPEC(twopar, x=W, y=A, fig.opt=FALSE, ini.val=list(1, seq(0.5, 1.5, by=0.05)))
B4 <- curvIPEC(twopar, theta=A4$par, x=W, y=A)
```
### biasIPEC

#### Bias Calculation Function

**Description**

Calculates the bias in the estimates of the parameters of a given nonlinear regression model.

**Usage**

```r
biasIPEC(expr, theta, x, y, tol = .Machine$double.eps, method = "Richardson", method.args = list(eps = 1e-04, d = 0.11, zero.tol = sqrt(.Machine$double.eps/7e-07), r = 6, v = 2, show.details = FALSE), side = NULL)
```

**Arguments**

- `expr`: A given nonlinear regression model
- `theta`: Vector of parameters of the model
- `x`: Vector or matrix of observations of independent variable(s)
- `y`: Vector of observations of response variable
- `tol`: The tolerance for detecting linear dependencies in the columns of a matrix for calculating its inverse. See the input argument of `tol` of the `solve` function in package `base`
- `method`: It is the same as the input argument of `method` of the `hessian` function in package `numDeriv`
- `method.args`: It is the same as the input argument of `method.args` of the `hessian` function in package `numDeriv`
- `side`: It is the same as the input argument of `side` of the `jacobian` function in package `numDeriv`

**Details**

The defined model should have two input arguments: a parameter vector and an independent variable vector or matrix, e.g. `myfun <- function(P, x){...}`, where `P` represents the parameter vector and `x` represents the independent variable vector or matrix.

An absolute value of `percent.bias` (see below) in excess of 1% appears to be a good rule of thumb for indicating nonlinear behavior (Ratkowsky 1983).
### Value

- **bias**  
  Calculated bias

- **percent.bias**  
  Calculated percentage bias that is equal to bias/estimate * 100%

### Note

The current function can be applicable to nonlinear models with multiple independent variables.

### Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

### References


### See Also

- `derivIPEC`, `hessian` in package `numDeriv`, `jacobian` in package `numDeriv`

### Examples

#### Example 1

```r
# The velocity of the reaction (counts/min^2) under different substrate concentrations  
# in parts per million (ppm) (Page 269 of Bates and Watts 1988)
x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten (MM) model
MM <- function(theta, x){
  theta[1]*x / ( theta[2] + x )
}

par1 <- c(212.68490865, 0.06412421)
res3 <- biasIPEC(MM, theta=par1, x=x1, y=y1, tol= 1e-20)
res3
```

#### Example 2

```r
# Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and
```
# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)

\[ x2 \leftarrow \text{seq}(15, 37, \text{by}=1) \]
\[ D2 \leftarrow (41.24, 37.16, 32.47, 26.22, 22.71, 19.01, 16.79, 15.63, 14.27, 12.48, 11.3, 10.56, 9.69, 9.14, 8.24, 8.02, 7.43, 7.27, 7.35, 7.49, 7.63, 7.9, 10.03) \]
\[ y2 \leftarrow 1/D2 \]
\[ y2 \leftarrow \sqrt{y2} \]

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
\[
\text{sqrt.LRF} \leftarrow \text{function}(P, x) \{
  \text{ropt} \leftarrow P[1]
  \text{Topt} \leftarrow P[2]
  \text{Tmin} \leftarrow P[3]
  \text{Tmax} \leftarrow P[4]
  \text{fun0} \leftarrow \text{function}(z) \{
    z[z < \text{Tmin}] \leftarrow \text{Tmin}
    z[z > \text{Tmax}] \leftarrow \text{Tmax}
    \text{return}(z)
  \}
  x \leftarrow \text{fun0}(x)
  \sqrt{(\text{ropt} \times (x-\text{Tmax}) \times (x-\text{Tmin})^2/((\text{Topt}-\text{Tmin}) \times ((\text{Topt}-\text{Tmin}) \times (\text{Topt}+\text{Tmin}-2 \times x))))}
\}
\]
\[
\text{myfun} \leftarrow \text{sqrt.LRF}
\]
\[
\text{par2} \leftarrow (0.1382926, 33.4575663, 5.5841244, 38.8282021)
\]

# To calculate bias
\[
\text{resu3} \leftarrow \text{biasIPEC(myfun, theta=par2, x=x2, y=y2, tol=1e-20)}
\]
\[
\text{resu3}
\]

#### Example 3
# Weight of cut grass data (Pattinson 1981)
# References:
# Clarke, G.P.Y. (1987) Approximate confidence limits for a parameter function in nonlinear
# Gebremariam, B. (2014) Is nonlinear regression throwing you a curve?
# Response to Herbage on Offer. unpublished M.Sc. thesis, University
# of Natal, Pietermaritzburg, South Africa, Department of Grassland Science.

# 'x4' is the vector of weeks after commencement of grazing in a pasture
# 'y4' is the vector of weight of cut grass from 10 randomly sited quadrants
\[ x4 \leftarrow 1:13 \]
\[ y4 \leftarrow c(3.183, 3.059, 2.871, 2.622, 2.541, 2.184, 2.110, 2.075, 2.018, 1.903, 1.770, 1.762, 1.550) \]
# Define the first case of Mitscherlich equation
MitA <- function(P1, x){
  P1[3] + P1[2]*exp(P1[1]*x)
}

# Define the second case of Mitscherlich equation
MitB <- function(P2, x){
}

# Define the third case of Mitscherlich equation
MitC <- function(P3, x, x1=1, x2=13){
  theta1 <- P3[1]
  beta2 <- P3[2]
  beta3 <- P3[3]
  theta2 <- (beta3 - beta2)/(exp(theta1*x2)-exp(theta1*x1))
  theta3 <- beta2/(1-exp(theta1*(x1-x2))) - beta3/(exp(theta1*(x2-x1))-1)
  theta3 + theta2*exp(theta1*x)
}

ini.val3 <- c(-0.1, 2.5, 1)
ini.val4 <- c(exp(-0.1), log(2.5), 1)
ini.val6 <- c(-0.15, 2.52, 1.09)

r0 <- fitIPEC( MitA, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parA <- r0$par
r3 <- biasIPEC( MitA, theta=parA, x=x4, y=y4, tol=1e-20 )

r0 <- fitIPEC( MitB, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parB <- r0$par
R3 <- biasIPEC( MitB, theta=parB, x=x4, y=y4, tol=1e-20 )

r0 <- fitIPEC( MitC, x=x4, y=y4, ini.val=ini.val6, xlim=NULL, ylim=NULL, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parC <- r0$par
RES3 <- biasIPEC( MitC, theta=parC, x=x4, y=y4, tol=1e-20 )

#################################################################################################

#### Example 4 ###################################################################################
# Data on biochemical oxygen demand (BOD; Marske 1967)
# References
# Pages 56, 255 and 271 in Bates and Watts (1988)
# 52, 391-396.
data(isom)
Y <- isom[,1]
X <- isom[,2:4]

# There are three independent variables saved in matrix 'X' and one response variable (Y)
# The first column of 'X' is the vector of partial pressure of hydrogen
# The second column of 'X' is the vector of partial pressure of n-pentane
# The third column of 'X' is the vector of partial pressure of isopentane
# Y is the vector of experimental reaction rate (in 1/hr)

isom.fun <- function(theta, x){
  x1  <- x[,1]
  x2  <- x[,2]
  x3  <- x[,3]
  theta1 <- theta[1]
  theta2 <- theta[2]
  theta3 <- theta[3]
  theta4 <- theta[4]
  theta1*theta3*(x2-x3/1.632) / ( 1 + theta2*x1 + theta3*x2 + theta4*x3 )
}

par8  <- c(35.92831619, 0.07084811, 0.03772270, 0.16718384)
cons3 <- biasIPEC( isom.fun, theta=par8, x=X, y=Y, tol= 1e-20 )
cons3

bic  

Bayesian Information Criterion (BIC) Calculation Function

Description
Calculates the BIC value(s) of the object(s) obtained from using the fitIPEC function.

Usage
bic( object, ... )

Arguments

object           A fitted model object for which there exists the sample size (n), estimate(s) of model parameter(s) (par), and residual sum of squares (RSS)

...            Optionally more fitted model objects

Details
BIC = p ln(n) - 2 ln(L), where p represents the number of model parameter(s) plus 1 for the error, n represents the sample size, and ln(L) represents the maximum log-likelihood of the estimated model (Spiess and Neumeyer, 2010).
Value

There is a BIC value corresponding to one object, and there is a vector of BIC values corresponding to the multiple objects.

Note

The BIC gives a higher penalty on the number of model parameters than the AIC.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also

aic, AIC in package stats, and BIC in package stats

Examples

```r
### Example ##############################################################################
data(leaves)
attach(leaves)
# Choose a geographical population (see Table S1 in Wang et al. [2018] for details)
# 1: AJ; 2: HN; 3: HW; 4: HZ; 5: JD;
# 6: JS; 7: SC; 8: TC; 9: TT; 10: TX
ind <- 1
L <- Length[PopuCode == ind]
W <- Width[PopuCode == ind]
A <- Area[PopuCode == ind]

# Define a model y = a*(x1*x2), where a is a parameter to be estimated
proportion <- function(theta, x){
a <- theta[1]
x1 <- x[,1]
x2 <- x[,2]
a*x1*x2
}

# Define a model y = a*(x1^b)*(x2^c), where a, b and c are parameters to be estimated
threepar <- function(theta, x){
a <- theta[1]
b <- theta[2]
c <- theta[3]
x1 <- x[,1]
}
```
x2 <- x[,2]
a*x1^b*x2^c
}

# Define a model y = a*x*b, where a and b are parameters to be estimated
twopar <- function(theta, x){
a <- theta[1]
b <- theta[2]
a*x^b
}

## Not run:
A1 <- fitIPEC(propor, x=cbind(L, W), y=A, fig.opt=FALSE, ini.val=list(seq(0.1, 1.5, by=0.1)))
B1 <- curvIPEC(propor, theta=A1$par, x=cbind(L, W), y=A)
A2 <- fitIPEC(threepar, x=cbind(L, W), y=A, fig.opt=FALSE, ini.val=list(A1$par, seq(0.5, 1.5, by=0.1), seq(0.5, 1.5, by=0.1)))
B2 <- curvIPEC(threepar, theta=A2$par, x=cbind(L, W), y=A)
A3 <- fitIPEC(twopar, x=L, y=A, fig.opt=FALSE, ini.val=list(1, seq(0.5, 1.5, by=0.05)))
B3 <- curvIPEC(twopar, theta=A3$par, x=L, y=A)
A4 <- fitIPEC(twopar, x=W, y=A, fig.opt=FALSE, ini.val=list(1, seq(0.5, 1.5, by=0.05)))
B4 <- curvIPEC(twopar, theta=A4$par, x=W, y=A)
aic(A1, A2, A3, A4)
bic(A1, A2, A3, A4)

## End(Not run)

 booIPEC

Bootstrap Function for Nonlinear Regression

Description

Generates the density distributions, standard deviations, confidence intervals, covariance matrices and correlation matrices of parameters based on bootstrap replications.

Usage

bootIPEC(expr, x, y, ini.val, target.fun = "RSS", control = list(), nboot = 200, CI = 0.95, fig.opt = TRUE, fold = 3.5, seed = NULL, unique.num = 2, prog.opt = TRUE)

Arguments

expr A given parametric model
x Vector or matrix of observations of independent variable(s)
y Vector of observations of response variable
init.val: A vector or list of initial values of model parameters.

target.fun: Objective function, “RSS” and “chi.sq”, for the optimization.

control: A list of control parameters for using the `optim` function in package `stats`.

nboot: The number of bootstrap replications.

CI: The confidence level(s) of the required interval(s).

fig.opt: Option of drawing figures of the distributions of bootstrap values of parameters and figures of pairwise comparisons of bootstrap values.

fold: Parameter reducing the extreme bootstrap values of parameters.

seed: The number to specify a seed for generating a set of random numbers.

unique.num: The least number of sampled non-overlapping data points for carrying out a bootstrap nonlinear regression.

prog.opt: Option of showing the running progress of bootstrap.

Details

`ini.val` can be a vector or a list that has saved initial values for model parameters, e.g. $y = \beta_0 + \beta_1 \times x + \beta_2 \times x^2$,

`ini.val = list(\beta_0=seq(5,15,len=2),\beta_1=seq(0.1,1,len=9),\beta_2=seq(0.01,0.05,len=5))`, which is similar to the usage of the input argument of `start` of `nls` in package `stats`.

`CI` determines the width of confidence intervals.

`fold` is used to delete the data whose differences from the median exceed a certain `fold` of the difference between 3/4 and 1/4 quantiles of the bootstrap values of a model parameter.

`seed` is used in the `set.seed` function in package `base`. If `seed` is assigned to be a specific integer, the users can obtain the same bootstrap values (standard deviation, median, mean, confidence interval) for repeating this function.

The default of `unique.num` is 2. That is, at least two non-overlapping data points randomly sampled from $(x, y)$ are needed for carrying out a bootstrap nonlinear regression.

Value

`M`: The matrix saving the fitted results of all `nboot` bootstrap values of model parameters and goodness of fit.

`perc.ci.mat`: The matrix saving the estimate, standard deviation, median, mean, and the calculated lower and upper limits of confidence interval based on the bootstrap percentile method.

`bc.a.ci.mat`: The matrix saving the estimate, standard deviation, median, mean, and the calculated lower and upper limits of confidence interval based on the bootstrap $BC_a$ method.

`covar.mat`: The covariance matrix of parameters based on the bootstrap values when `nboot` > 1.

`cor.mat`: The correlation matrix of parameters based on the bootstrap values when `nboot` > 1.
Note
To obtain reliable confidence intervals of model parameters, more than 2000 bootstrap replications are recommended; whereas to obtain a reliable standard deviation of the estimate of a parameter, more than 30 bootstrap replications are sufficient (Efron and Tibshirani 1993). bca.ci.mat is recommended to show better confidence intervals of parameters than those in perc.ci.mat.

The outputs of model parameters will all be represented by $\theta_i$, i from 1 to p, where p represents the number of model parameters. The letters of model parameters defined by users such as $\beta_i$ will be automatically replaced by $\theta_i$.

Author(s)
Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also
fitIPEC

Examples
### Example 1

```r
# The velocity of the reaction (counts/min^2) under different substrate concentrations
# in parts per million (ppm) (Page 269 of Bates and Watts 1988)
x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten (MM) model
MM <- function(theta, x){
  theta[1]*x / ( theta[2] + x )
}

## Not run:
res4 <- bootIPEC( MM, x=x1, y=y1, ini.val=c(200, 0.05), target.fun = "RSS",
  control=list(reltol=1e-20, maxit=40000), nboot=2000, CI=0.95,
  fig.opt=TRUE, seed=123 )
res4
## End(Not run)
```

### Example 2
graphics.off()
# Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and

# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)

x2 <- seq(15, 37, by=1)
D2 <- c(41.24,37.16,32.47,26.22,22.71,19.01,16.79,15.63,14.27,12.48,
      11.3,10.56,9.69,9.14,8.24,8.02,7.43,7.27,7.35,7.49,7.63,7.9,10.03)
y2 <- 1/D2
y2 <- sqrt( y2 )
ini.val1 <- c(0.14, 30, 10, 40)

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x){
  ropt <- P[1]
  Topt <- P[2]
  Tmin <- P[3]
  Tmax <- P[4]
  fun0 <- function(z){
    z[z < Tmin] <- Tmin
    z[z > Tmax] <- Tmax
    return(z)
  }
  x <- fun0(x)
  sqrt( ropt*(x-Tmax)*(x-Tmin)^2/((Topt-Tmin)*((Topt-Tmin)
      *(Topt-Tmax)-(Topt+Tmin-2*x))))
}

myfun <- sqrt.LRF
## Not run:
resu4 <- bootIPEC( myfun, x=x2, y=y2, ini.val=ini.val1, target.fun="RSS",
                  nboot=2000, CI=0.95, fig.opt=TRUE, seed=123 )
resu4
## End(Not run)

#################################################################################################
#### Example 3 ##################################################################################

graphics.off()
# Height growth data of four species of bamboos (Gramineae: Bambusoideae)
# Reference(s):
# Ecol. Model. 349, 1-10.

data(shoots)
attach(shoots)

# Choose a species
# 1: Phyllostachys iridescens; 2: Phyllostachys mannii;
# 3: Sinobambusa tootsik; 4: Pleioblastus maculatus

# 'x3' is the vector of the observation times from a specific starting time of growth
# 'y3' is the vector of the aboveground height values of bamboo shoots at 'x3'

ind <- 2
x3 <- time[code == ind]
y3 <- height[code == ind]

# Define the beta sigmoid model (bsm)
bsm <- function(P, x){
  P <- cbind(P)
  if(length(P) != 4) {stop("The number of parameters should be 4!")}
  ropt <- P[1]
  topt <- P[2]
  tmin <- P[3]
  tmax <- P[4]
  tailor.fun <- function(x){
    x[x < tmin] <- tmin
    x[x > tmax] <- tmax
    return(x)
  }
  x <- tailor.fun(x)
  return(ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*
           (x-tmin)/(topt-tmin))^((topt-tmin)/(tmax-topt))
}

# Define the simplified beta sigmoid model (simp.bsm)
simp.bsm <- function(P, x, tmin=0){
  P <- cbind(P)
  ropt <- P[1]
  topt <- P[2]
  tmax <- P[3]
  tailor.fun <- function(x){
    x[x < tmin] <- tmin
    x[x > tmax] <- tmax
    return(x)
  }
  x <- tailor.fun(x)
  return(ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*
           (x-tmin)/(topt-tmin))^((topt-tmin)/(tmax-topt))
}

# For the original beta sigmoid model
ini.val2 <- c(40, 30, 5, 50)
xlab2 <- "Time (d)"
ylab2 <- "Height (cm)"
```r
## Not run:
re4 <- bootIPEC( bsm, x=x3, y=y3, ini.val=ini.val2, target.fun = "RSS",
                control=list(trace=FALSE, reltol=1e-20, maxit=50000),
                nboot=2000, CI=0.95, fig.opt=TRUE, fold=10, seed=123 )
re4
## End(Not run)

# For the simplified beta sigmoid model (in comparison with the original beta sigmoid model)
ini.val7 <- c(40, 30, 50)
## Not run:
RESU4 <- bootIPEC( simp.bsm, x=x3, y=y3, ini.val=ini.val7, target.fun = "RSS",
              control=list(trace=FALSE, reltol=1e-20, maxit=50000),
              nboot=2000, CI=0.95, fig.opt=TRUE, fold=10, seed=123 )
RESU4
## End(Not run)
```

### Example 4 ##################################################################################
```
# Weight of cut grass data (Pattinson 1981)
# References:
# Clarke, G.P.Y. (1987) Approximate confidence limits for a parameter function in nonlinear
# Gebremariam, B. (2014) Is nonlinear regression throwing you a curve?
# Response to Herbage on Offer. unpublished M.Sc. thesis, University
# of Natal, Pietermaritzburg, South Africa, Department of Grassland Science.

# 'x4' is the vector of weeks after commencement of grazing in a pasture
# 'y4' is the vector of weight of cut grass from 10 randomly sited quadrants

x4 <- 1:13
y4 <- c( 3.183, 3.059, 2.871, 2.622, 2.541, 2.184,
         2.110, 2.075, 2.018, 1.903, 1.770, 1.762, 1.550 )

# Define the first case of Mitscherlich equation
MitA <- function(P1, x){
  P1[3] + P1[2]*exp(P1[1]*x)
}

# Define the second case of Mitscherlich equation
MitB <- function(P2, x){
}

# Define the third case of Mitscherlich equation
MitC <- function(P3, x, x1=1, x2=13){
```
```
theta1 <- P3[1]
beta2 <- P3[2]
beta3 <- P3[3]
theta2 <- (beta3 - beta2)/(exp(theta1*x2)-exp(theta1*x1))
theta3 <- beta2/(1-exp(theta1*(x1-x2))) - beta3/(exp(theta1*(x2-x1))-1)
theta3 + theta2*exp(theta1*x)

## Not run:
ini.val3 <- c(-0.1, 2.5, 1.0)
r4 <- bootIPEC( MitA, x=x4, y=y4, ini.val=ini.val3, target.fun="RSS",
nboot=2000, CI=0.95, fig.opt=TRUE, seed=123)
r4

ini.val4 <- c(exp(-0.1), log(2.5), 1)
R4 <- bootIPEC( MitB, x=x4, y=y4, ini.val=ini.val4, target.fun="RSS",
nboot=2000, CI=0.95, fig.opt=TRUE, seed=123)
R4

# ini.val6 <- c(-0.15, 2.52, 1.09)
iv.list2 <- list(seq(-2, -0.05, len=5), seq(1, 4, len=8), seq(0.05, 3, by=0.5))
RES0 <- fitIPEC( MitC, x=x4, y=y4, ini.val=iv.list2, target.fun="RSS",
control=list(trace=FALSE, reltol=1e-10, maxit=5000))
RES0$par
RES4 <- bootIPEC( MitC, x=x4, y=y4, ini.val=iv.list2, target.fun="RSS",
control=list(trace=FALSE, reltol=1e-10, maxit=5000),
nboot=5000, CI=0.95, fig.opt=TRUE, fold=3.5, seed=123, unique.num=2)
RES4

## End(Not run)

## curvIPEC
RMS Curvature Calculation Function

Description
Calculates the root mean square curvatures (intrinsic and parameter-effects curvatures) of a nonlin-
er regression model.

Usage
curvIPEC(expr, theta, x, y, tol = 1e-16, alpha = 0.05, method = "Richardson",
method.args = list(eps = 1e-04, d = 0.11,
zero.tol = sqrt(.Machine$double.eps/7e-07),
r = 6, v = 2, show.details = FALSE), side = NULL)
Arguments

- **expr**: A given parametric model
- **theta**: Vector of parameters of the model
- **x**: Vector or matrix of observations of independent variable(s)
- **y**: Vector of observations of response variable
- **tol**: The tolerance for detecting linear dependencies in the columns of a matrix in the QR decomposition. See the input argument of tol of the qr function in package base
- **alpha**: Parameter controlling the significance level for testing the significance of a curvature
- **method**: It is the same as the input argument of method of the hessian function in package numDeriv
- **method.args**: It is the same as the input argument of method.args of the hessian function in package numDeriv
- **side**: It is the same as the input argument of side of the jacobian function in package numDeriv

Details

This function was built based on the hessian and jacobian functions in package numDeriv, with reference to the rms.curv function in package MASS. However, it is more general without being limited by the deriv3 function in package stats and nls class like the rms.curv function in package MASS. It mainly relies on package numDeriv. The users only need provide the defined model, the fitted parameter vector, and the observations of independent and response variables, they will obtain the curvatures. The input argument theta can be obtained using the fitIPEC function in the current package, and it also can be obtained using the other nonlinear regression functions.

Value

- **rms.ic**: The root mean square intrinsic curvature
- **rms.pec**: The root mean square parameter-effects curvature
- **critical.c**: Critical curvature value

Note

The calculation precision of curvature mainly depends on the setting of method.args. The two important default values in the list of method.args are d = 0.11, and r = 6.

This function cannot be used to calculate the maximum intrinsic and parameter-effects curvatures.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li
References


See Also

derivIPEC, hessian in package numDeriv, jacobian in package numDeriv, rms.curv in package MASS

Examples

#### Example 1
# The velocity of the reaction (counts/min^2) under different substrate concentrations
# in parts per million (ppm) (Pages 255 and 269 of Bates and Watts 1988)

x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten model
MM <- function(theta, x){
  theta[1]*x / ( theta[2] + x )
}

par1 <- c(212.68490865, 0.06412421)

# To calculate curvatures
res2 <- curvIPEC(MM, theta=par1, x=x1, y=y1, alpha=0.05, method="Richardson",
  method.args=list(eps=1e-4, d=0.11, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
res2

#### Example 2
# Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and

# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)
x2 <- seq(15, 37, by=1)
D2 <- c( 41.24, 37.16, 32.47, 26.22, 22.71, 19.01, 16.79, 15.63, 14.27, 12.48,
       11.3, 10.56, 9.69, 9.14, 8.24, 8.02, 7.43, 7.27, 7.35, 7.49, 7.63, 7.9, 10.03 )
y2 <- 1/D2
y2 <- sqrt(y2)

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x){
  ropt <- P[1]
  Topt <- P[2]
  Tmin <- P[3]
  Tmax <- P[4]
  fun0 <- function(z){
    z[z < Tmin] <- Tmin
    z[z > Tmax] <- Tmax
    return(z)
  }
  x <- fun0(x)
  sqrt(ropt*(x-Tmax)*(x-Tmin)^2/((Topt-Tmin)*((Topt-Tmin)
     *(x-Topt)-(Topt-Tmax)*(Topt+Tmin-2*x))) )
}

myfun <- sqrt.LRF
par2 <- c(0.1382926, 33.4575663, 5.5841244, 38.8282021)

# To calculate curvatures
resu2 <- curvIPEC( myfun, theta=par2, x=x2, y=y2, alpha=0.05, method="Richardson",
                   method.args=list(eps=1e-4, d=0.11, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
resu2

# Example 3
# Height growth data of four species of bamboos (Gramineae: Bambusoideae)
# Reference(s):
# Ecol. Model. 349, 1-10.

data(shoots)
attach(shoots)

# Choose a species
# 1: Phyllostachys iridescens; 2: Phyllostachys mannii;
# 3: Sinobambusa tootsik; 4: Pleioblastus maculatus
# 'x3' is the vector of the observation times (in d) from a specific starting time of growth
# 'y3' is the vector of the aboveground height values (in cm) of bamboo shoots at 'x3'

ind <- 3
x3 <- time[code == ind]
y3 <- height[code == ind]
# Define the beta sigmoid model (bsm)
bsm <- function(P, x){
  P <- cbind(P)
  if(length(P) != 4) {stop("The number of parameters should be 4!")}
  ropt <- P[1]
  topt <- P[2]
  tmin <- P[3]
  tmax <- P[4]
  tailor.fun <- function(x){
    x[x < tmin] <- tmin
    x[x > tmax] <- tmax
    return(x)
  }
  x <- tailor.fun(x)
  ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*((x-tmin)/(topt-tmin))^{((topt-tmin)/(tmax-topt))}
}

# Define the simplified beta sigmoid model (simp.bsm)
simp.bsm <- function(P, x, tmin=0){
  P <- cbind(P)
  ropt <- P[1]
  topt <- P[2]
  tmax <- P[3]
  tailor.fun <- function(x){
    x[x < tmin] <- tmin
    x[x > tmax] <- tmax
    return(x)
  }
  x <- tailor.fun(x)
  ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*((x-tmin)/(topt-tmin))^{((topt-tmin)/(tmax-topt))}
}

# For the original beta sigmoid model
ini.val2 <- c(40, 30, 5, 50)
xlab2 <- "Time (d)"
ylab2 <- "Height (cm)"
re0 <- fitIPEC(bsm, x=x3, y=y3, ini.val=ini.val2, xlim=NULL, ylim=NULL, xlab=xlab2, ylab=ylab2, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000))
par3 <- re0$par
par3
re1 <- derivIPEC(bsm, theta=par3, x3[20], method="Richardson", method.args=list(eps=1e-4, d=0.11, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
re1
re2 <- curvIPEC(bsm, theta=par3, x=x3, y=y3, alpha=0.05, method="Richardson", method.args=list(eps=1e-4, d=0.11, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
re2

# For the simplified beta sigmoid model (in comparison with the original beta sigmoid model)
ini.val7 <- c(40, 30, 50)

RESU0 <- fitIPEC( simp.bsm, x=x3, y=y3, ini.val=ini.val7,
               xlim=NULL, ylim=NULL, xlab=xlab2, ylab=ylab2,
               fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )

par7 <- RESU0$par

RESU2 <- curvIPEC( simp.bsm, theta=par7, x=x3, y=y3, alpha=0.05, method="Richardson",
               method.args=list(eps=1e-4, d=0.11,
               zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )

RESU2

#####################################################

#### Example 4 ####################################################################################
# Weight of cut grass data (Pattinson 1981)
# References:
# Clarke, G.P.Y. (1987) Approximate confidence limits for a parameter function in nonlinear
# Gebremariam, B. (2014) Is nonlinear regression throwing you a curve?
# Response to Herbage on Offer. unpublished M.Sc. thesis, University
# of Natal, Pietermaritzburg, South Africa, Department of Grassland Science.
#
# 'x4' is the vector of weeks after commencement of grazing in a pasture
# 'y4' is the vector of weight of cut grass from 10 randomly sited quadrants

x4 <- 1:13
y4 <- c(3.183, 3.059, 2.871, 2.622, 2.541, 2.184, 2.110, 2.075, 2.018, 1.903, 1.770, 1.762, 1.550)

# Define the first case of Mitscherlich equation
MitA <- function(P1, x){
    P1[3] + P1[2]*exp(P1[1]*x)
}

# Define the second case of Mitscherlich equation
MitB <- function(P2, x){
}

# Define the third case of Mitscherlich equation
MitC <- function(P3, x, x1=1, x2=13){
    theta1 <- P3[1]
    beta2 <- P3[2]
    beta3 <- P3[3]
    theta2 <- (beta3 - beta2)/((exp(theta1*x2)-exp(theta1*x1))
    theta3 <- beta2/(1-exp(theta1*(x1-x2))) - beta3/((exp(theta1*(x2-x1))-1)
    theta3 + theta2*exp(theta1*x)
}
ini.val3 <- c(-0.1, 2.5, 1)
r0 <- fitIPEC(MitA, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL,
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000))
parA <- r0$par
parA
r2 <- curvIPEC(MitA, theta=parA, x=x4, y=y4, alpha=0.05, method="Richardson",
method.args=list(eps=1e-4, d=0.11,
zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
r2

ini.val4 <- c(exp(-0.1), log(2.5), 1)
R0 <- fitIPEC(MitB, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL,
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000))
parB <- R0$par
parB
R2 <- curvIPEC(MitB, theta=parB, x=x4, y=y4, alpha=0.05, method="Richardson",
method.args=list(eps=1e-4, d=0.11,
zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
R2

ini.val6 <- c(-0.15, 2.52, 1.09)
RES0 <- fitIPEC(MitC, x=x4, y=y4, ini.val=ini.val6, xlim=NULL, ylim=NULL,
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000))
parC <- RES0$par
parC
RES2 <- curvIPEC(MitC, theta=parC, x=x4, y=y4,
tol=1e-20, alpha=0.05, method="Richardson",
method.args=list(eps=1e-4, d=0.11,
zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
RES2

##################################################################################################
#### Example 5 ###################################################################################
# Conductance of a thermistor (y5) as a function of temperature (x5) (Meyer and Roth, 1972)
# References:
# Page 120 in Ratkowsky (1983)
# Meyer, R.R. and Roth P.M. (1972) Modified damped least squares:
x5 <- seq(50, 125, by=5)
y5 <- c(34780, 28610, 23650, 19630, 16370, 13720, 11540, 9744,
8261, 7030, 6005, 5147, 4427, 3820, 3307, 2872)
y5 <- log(y5)

conduct.fun <- function(P, x){
-P[1]+P[2]/(x+P[3])
}


### Example 6 ###################################################################################

# Data on biochemical oxygen demand (BOD; Marske 1967)
# References:
# Pages 255 and 270 in Bates and Watts (1988)

# 'x6' is a vector of time (in d)
# 'y6' is a vector of biochemical oxygen demand (mg/l)

x6 <- c(1, 2, 3, 4, 5, 7)
y6 <- c(8.3, 10.3, 19.0, 16.0, 15.6, 19.8)

BOD.fun <- function(P, x)
  P[1]*(1-exp(P[2]*x))
}

ini.val7 <- c(210, 0.06)
consq0 <- fitIPEC( BOD.fun, x=x6, y=y6, ini.val=ini.val7, xlim=NULL, ylim=NULL, fig.opt=TRUE, control=list( trace=FALSE, reltol=1e-20, maxit=50000 ) )
par7 <- consq0$par

consq2 <- curvIPEC( BOD.fun, theta=par7, x=x6, y=y6, alpha=0.05, method="Richardson", method.args=list(eps=1e-4, d=0.11, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )

consq2

### Example 7 ###################################################################################

# Data on biochemical oxygen demand (BOD; Marske 1967)
# References:
# Pages 56, 255 and 271 in Bates and Watts (1988)
# 52, 391-396.

data(isom)
Y <- isom[,1]
# There are three independent variables saved in matrix 'X' and one response variable (Y)
# The first column of 'X' is the vector of partial pressure of hydrogen
# The second column of 'X' is the vector of partial pressure of n-pentane
# The third column of 'X' is the vector of partial pressure of isopentane
# Y is the vector of experimental reaction rate (in 1/hr)

isom.fun <- function(theta, x){
  x1 <- x[,1]
  x2 <- x[,2]
  x3 <- x[,3]
  thetal <- theta[1]
  theta2 <- theta[2]
  theta3 <- theta[3]
  theta4 <- theta[4]
  thetal*theta3*(x2-x3/1.632) / ( 1 + theta2*x1 + theta3*x2 + theta4*x3 )
}

par8 <- c(35.92831619, 0.07084811, 0.03772270, 0.16718384)
cons2 <- curvIPEC( isom.fun, theta=par8, x=X, y=Y, alpha=0.05, method="Richardson",
  method.args=list(eps=1e-4, d=0.11,
    zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )

cons2

#derivIPEC

derivIPEC

Derivative Calculation Function

Description

Calculates the Jacobian and Hessian matrices of model parameters at a vector z.

Usage

derivIPEC(expr, theta, z, method = "Richardson",
  method.args = list(eps = 1e-04, d = 0.11,
    zero.tol = sqrt(.Machine$double.eps/7e-07), r = 6, v = 2,
    show.details = FALSE), side = NULL)

Arguments

expr A given parametric model
theta Vector of parameters of the model
z Vector where the derivatives are calculated
method It is the same as the input argument of method of the hessian function in package numDeriv
method.args It is the same as the input argument of method.args of the hessian function in package numDeriv
side

It is the same as the input argument of side of the `jacobian` function in package `numDeriv`.

Details

The Hessian and Jacobian matrices are calculated at a vector \( z \), which represents a value of a single independent variable or a combination of different values of multiple independent variables.

Value

- **Jacobian**: The Jacobian matrix of parameters at \( z \)
- **Hessian**: The Hessian matrix of parameters at \( z \)

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also

- `biasIPEC`, `skewIPEC`, `curvIPEC`, `hessian` in package `numDeriv`, `jacobian` in package `numDeriv`

Examples

```r
#### Example 1
# Define the Michaelis-Menten model
MM <- function(theta, x){
  theta[1]*x / ( theta[2] + x )
}
par1 <- c(212.68490865, 0.06412421)
res1 <- derivIPEC(MM, theta=par1, z=0.02, method="Richardson",
                 method.args=list(eps=1e-4, d=0.1, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
res1

#### Example 2
# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x){
  ropt <- P[1]
  Topt <- P[2]
  Tmin <- P[3]
  ...
```

```
Tmax <- P[4]
fun0 <- function(z){
  z[z < Tmin] <- Tmin
  z[z > Tmax] <- Tmax
  return(z)
}
x <- fun0(x)
sqrt(ropt*(x-Tmax)*(x-Tmin)^2/((Topt-Tmin)*(Topt-Tmin)
  *(x-Topt)-(Topt-Tmax)*(Topt+Tmin-2*x)))
}
myfun <- sqrt.LRF
par2 <- c(0.1382926, 33.4575663, 5.5841244, 38.8282021)
resul1 <- derivIPEC(myfun, theta=par2, z=15, method="Richardson",
method.args=list(eps=1e-4, d=0.1, zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
resul1

### Example 3 ####################################################################################################
#### Example 3 #####################################################################################
# Weight of cut grass data (Pattinson 1981)
# References:
# Clarke, G.P.Y. (1987) Approximate confidence limits for a parameter function in nonlinear
# Gebremariam, B. (2014) Is nonlinear regression throwing you a curve?
# Response to Herbage on Offer. unpublished M.Sc. thesis, University
# of Natal, Pietermaritzburg, South Africa, Department of Grassland Science.

# 'x4' is the vector of weeks after commencement of grazing in a pasture
# 'y4' is the vector of weight of cut grass from 10 randomly sited quadrants
x4 <- 1:13
y4 <- c(3.183, 3.059, 2.871, 2.622, 2.541, 2.184, 2.110, 2.075, 2.018, 1.903, 1.770, 1.762, 1.550)

# Define the third case of Mitscherlich equation
MitC <- function(P3, x){
  theta1 <- P3[1]
  beta2 <- P3[2]
  beta3 <- P3[3]
  x1 <- 1
  x2 <- 13
  theta2 <- (beta3 - beta2)/(exp(theta1*x2)-exp(theta1*x1))
  theta3 <- beta2/(1-exp(theta1*(x1-x2))) - beta3/(exp(theta1*(x2-x1))-1)
  theta3 + theta2*exp(theta1*x)
}
ini.val6 <- c(-0.15, 2.52, 1.09)
RES0 <- fitIPEC(MitC, x=x4, y=y4, ini.val=ini.val6, xlim=NULL, ylim=NULL, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000))
parC <- RES0$par
parC
RES1 <- derivIPEC( MitC, theta=parC, z=2, method="Richardson",
                   method.args=list(eps=1e-4, d=0.11,
                                    zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
RES1

---

**fitIPEC**  
*Nonlinear Fitting Function*

**Description**

Estimates the parameters of a given parametric model using the `optim` function in package `stats`.

**Usage**

```r
fitIPEC( expr, x, y, ini.val, target.fun = "RSS", control = list(),
          fig.opt = TRUE, xlim = NULL, ylim = NULL, xlab = NULL, ylab = NULL )
```

**Arguments**

- `expr` A given parametric model
- `x` Vector or matrix of observations of independent variable(s)
- `y` Vector of observations of response variable
- `ini.val` A vector or list of initial values of model parameters
- `target.fun` Objective function, “RSS” and “chi.sq”, for the optimization
- `control` A list of control parameters for using the `optim` function in package `stats`
- `fig.opt` An option to determine whether to draw the fitted curve
- `xlim` The shown range of the x-axis
- `ylim` The shown range of the y-axis
- `xlab` The label of the x-axis
- `ylab` The label of the y-axis

**Details**

The Nelder-Mead algorithm is the default in the `optim` function in package `stats`. The user can accurately estimate the model parameters by setting smaller relative convergence tolerance and larger maximum number of iterations in the input argument of `control`, e.g. `control=list(trace=FALSE,reltol=1e-20,maxit=50000)`, at the expense of the running speed.

`ini.val` can be a vector or a list that has saved initial values for model parameters, e.g.  
\[
y = \beta_0 + \beta_1 x + \beta_2 x^2,
\]

ini.val = list(beta0=seq(5,15,len=2),beta1=seq(0.1,1,len=9),beta2=seq(0.01,0.05,len=5)),
which is similar to the usage of the input argument of start of `nls` in package `stats`. 
Value

<table>
<thead>
<tr>
<th>expr</th>
<th>The formula used</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>Vector of estimates of parameters</td>
</tr>
<tr>
<td>RSS</td>
<td>Calculated residual sum of squares</td>
</tr>
<tr>
<td>chi.sq</td>
<td>Calculated $\chi^2$</td>
</tr>
<tr>
<td>R.sq</td>
<td>Calculated $R^2$</td>
</tr>
<tr>
<td>n</td>
<td>The number of data points, namely the sample size</td>
</tr>
</tbody>
</table>

Note

This function can be applicable to a nonlinear parametric model with a single independent variable or with multiple independent variables.

$R.sq$ is only used to help users intuitively judge whether the fitted curve seriously deviates from the actual observations. However, it should NOT be used to decide which of several competing models is the most appropriate (Pages 44-45 in Ratkowsky 1990). RSS and curvatures are among the suitable candidates to answer such a question.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also

`bootIPEC`, `optim` in package `stats`

Examples

```r
#### Example 1 ###################################################################################
graphics.off()
# The velocity of the reaction (counts/min^2) under different substrate concentrations
# in parts per million (ppm) (Page 269 of Bates and Watts 1988)
x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten model
MM <- function(theta, x){
  theta[1]*x / (theta[2] + x)
}

res0 <- fitIPEC(MM, x=x1, y=y1, ini.val=c(200, 0.05),
                xlim=c(0, 1.5), ylim=c(0, 250), fig.opt=TRUE)
par1 <- res0$par
par1
```
# The input names of parameters will not affect the fitted results.
# We can use other names to replace theta1 and theta2.
iv.list1 <- list( theta1=seq(100, 300, by=50), theta2=seq(10, 100, by=10) )
result0 <- fitIPEC( MM, x=x1, y=y1, ini.val=iv.list1, xlim=c(0, 1.5), ylim=c(0, 250),
fig.opt=FALSE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
param1 <- result0$par

### Not run:

#### Example 2

# Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and

# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)

dx2 <- seq(15, 37, by=1)
D2 <- c(41.24,37.16,32.47,26.22,22.71,19.01,16.79,15.63,14.27,12.48,
11.3,10.56,9.69,9.14,8.24,8.02,7.43,7.27,7.35,7.49,7.63,7.9,10.03)
y2 <- 1/D2
y2 <- sqrt( y2 )
ini.val1 <- c(0.14, 30, 10, 40)

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x){
  ropt <- P[1]
  Topt <- P[2]
  Tmin <- P[3]
  Tmax <- P[4]
  fun0 <- function(z){
    z[z < Tmin] <- Tmin
    z[z > Tmax] <- Tmax
    return(z)
  }
  x <- fun0(x)
  sqrt((ropt*(x-Topt)^2+((Topt-Tmin)*((Topt-Tmin)
  *(x-Topt)-(Topt-Tmax)*(Topt+Tmin-2*x))))
}
myfun <- sqrt.LRF
xlab1 <- expression( paste("Temperature (", degree, "C)" , sep="" ) )
ylab1 <- expression( paste("Developmental rate"^(1/2), " (", d^"-1", ")", sep="" ) )
resu0 <- fitIPEC( myfun, x=x2, y=y2, ini.val=ini.val1, xlim=NULL, ylim=NULL, xlab=xlab1, ylab=ylab1, fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
par2 <- resu0$par
par2
resu0

## End(Not run)

### Example 3 ###################################################################################
graphics.off()
data(shoots)
attach(shoots)

# Choose a species
# 1: Phyllostachys iridescens; 2: Phyllostachys mannii;
# 3: Sinobambusa tootsik; 4: Pleioblastus maculatus
# 'x3' is the vector of observation times from a specific starting time of growth
# 'y3' is the vector of the aboveground height values of bamboo shoots at 'x3'
ind <- 2
x3 <- time[code == ind]
y3 <- height[code == ind]

# Define the beta sigmoid model (bsm)
bsm <- function(P, x){
P <- cbind(P)
if(length(P) !=4 ) {stop(" The number of parameters should be 4!")}
ropt <- P[1]
topt <- P[2]
tmin <- P[3]
tmax <- P[4]
tailor.fun <- function(x){
x[x < tmin] <- tmin
x[x > tmax] <- tmax
return(x)
}
x <- tailor.fun(x)
ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*(
(x-tmin)/(topt-tmin))^((topt-tmin)/(tmax-topt))
}
ini.val2 <- c(40, 30, 5, 50)
xlab2 <- "Time (d)"
ylab2 <- "Height (cm)"

re0 <- fitIPEC( bsm, x=x3, y=y3, ini.val=ini.val2, 
                xlim=NULL, ylim=NULL, xlab=xlab2, ylab=ylab2, 
                fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
par3 <- re0$par

##################################################################################################
# Example 4 #########################################################################################
# Data on biochemical oxygen demand (BOD; Marske 1967)
# References:
# Pages 56, 255 and 271 in Bates and Watts (1988)
# 52, 391-396.

data(isom)
Y <- isom[,1]
X <- isom[,2:4]

# There are three independent variables saved in matrix 'X' and one response variable (Y)
# The first column of 'X' is the vector of partial pressure of hydrogen
# The second column of 'X' is the vector of partial pressure of n-pentane
# The third column of 'X' is the vector of partial pressure of isopentane
# Y is the vector of experimental reaction rate (in 1/hr)

isom.fun <- function(theta, x){
  x1 <- x[,1]
  x2 <- x[,2]
  x3 <- x[,3]
  theta1 <- theta[1]
  theta2 <- theta[2]
  theta3 <- theta[3]
  theta4 <- theta[4]
  theta1*theta3*(x2-x3/1.632) / ( 1 + theta2*x1 + theta3*x2 + theta4*x3 )
}

ini.val8 <- c(35, 0.1, 0.05, 0.2)
cons1 <- fitIPEC( isom.fun, x=X, y=Y, ini.val=ini.val8, control=list(
                  trace=FALSE, reltol=1e-20, maxit=50000) )
par8 <- cons1$par

###############################################################################

IPEC

IPEC

Root Mean Square Curvature Calculation

Description

Calculates the RMS intrinsic and parameter-effects curvatures of a nonlinear regression model.
Details

The DESCRIPTION file:

Package: IPEC
Type: Package
Title: Root Mean Square Curvature Calculation
Version: 0.1.3
Date: 2020-06-26
Author: Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li
Maintainer: Peijian Shi <peijianshi@gmail.com>
Imports: numDeriv (>= 2016.8-1), MASS
Description: Calculates the RMS intrinsic and parameter-effects curvatures of a nonlinear regression model.
Depends: R (>= 3.5.0)
License: GPL (>= 2)
NeedsCompilation: no

Index of help topics:

IPEC Root Mean Square Curvature Calculation
aic Akaike Information Criterion (AIC) Calculation Function
biasIPEC Bias Calculation Function
bic Bayesian Information Criterion (BIC) Calculation Function
bootIPEC Bootstrap Function for Nonlinear Regression
curvIPEC RMS Curvature Calculation Function
derivIPEC Derivative Calculation Function
fitIPEC Nonlinear Fitting Function
isom Data on Biochemical Oxygen Demand
leaves Leaf Data of _Parrotia subaequalis_ (Hamamelidaceae)
shoots Height Growth Data of Bamboo Shoots
skewIPEC Skewness Calculation Function

Note

We are deeply thankful to Drs. Paul Gilbert and Jinlong Zhang for their invaluable help during creating this package.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li
Maintainer: Peijian Shi <peijianshi@gmail.com>

References


See Also

`hessian` in package `numDeriv`, `jacobian` in package `numDeriv`, `rms.curv` in package `MASS`

Examples

```r
### Example 1
# The velocity of the reaction (counts/min^2) under different substrate concentrations
# in parts per million (ppm) (Page 269 of Bates and Watts 1988)
x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten model
MM <- function(theta, x){
  theta[1]*x / ( theta[2] + x )
}

res0 <- fitIPEC( MM, x=x1, y=y1, ini.val=c(200, 0.05),
  xlim=c( 0, 1.5 ), ylim=c(0, 250), fig.opt=TRUE )
par1 <- res0$par

res1 <- derivIPEC( MM, theta=par1, z=x1[1], method="Richardson",
  method.args=list(eps=1e-4, d=0.11,
    zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )

res2 <- curvIPEC( MM, theta=par1, x=x1, y=y1, alpha=0.05, method="Richardson",
  method.args=list(eps=1e-4, d=0.11,
    zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )

res3 <- biasIPEC(MM, theta=par1, x=x1, y=y1, tol= 1e-20)

## Not run:
res4 <- bootIPEC( MM, x=x1, y=y1, ini.val=par1, target.fun = "RSS",
  control=list(reltol=1e-20, maxit=40000),
  nboot=2000, CI=0.95, fig.opt=TRUE, seed=123 )

## End(Not run)

# To calculate skewness
```

IPEC

res5 <- skewIPEC(MM, theta=par1, x=x1, y=y1, tol=1e-20)
res5

#################################################################################################
## Not run:
### Example 2 #################################################################################################

# Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and

# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)

x2 <- seq(15, 37, by=1)
D2 <- c(41.24, 37.16, 32.47, 26.22, 22.71, 19.01, 16.79, 15.63, 14.27, 12.48,
       11.35, 10.56, 9.69, 9.14, 8.24, 8.02, 7.43, 7.27, 7.35, 7.49, 7.63, 7.90, 10.03)
y2 <- 1/D2
y2 <- sqrt( y2 )

ini.val1 <- c(0.14, 30, 10, 40)

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x){
  ropt <- P[1]
  T0pt <- P[2]
  Tmin <- P[3]
  Tmax <- P[4]
  fun0 <- function(z){
    z[z < Tmin] <- Tmin
    z[z > Tmax] <- Tmax
    return(z)
  }
  x <- fun0(x)
  sqrt( ropt*(x-Tmax)*(x-Tmin)^2/((T0pt-Tmin)*(T0pt-Tmin-
       (T0pt-Tmax)*(T0pt+Tmin-2*x))) )
}

myfun <- sqrt.LRF
xlab1 <- expression( paste("Temperature (", degree, "C)", se="")
ylab1 <- expression( paste("Developmental rate"^{1/2}, " (", d{"-1"}, ")", se="")
resu0 <- fitIPEC( myfun, x=x2, y=y2, ini.val=ini.val1, xlim=NULL, ylim=NULL,
                  xlab=xlab1, ylab=ylab1, fig.opt=TRUE,
                  control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
par2 <- resu0$par
par2
resu1 <- derivIPEC( myfun, theta=par2, z=x2[1], method="Richardson", 
               method.args=list(eps=1e-4, d=0.11, 
               zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
resu1

# To calculate curvatures
resu2 <- curvIPEC( myfun, theta=par2, x=x2, y=y2, alpha=0.05, method="Richardson", 
               method.args=list(eps=1e-4, d=0.11, 
               zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
resu2

# To calculate bias
resu3 <- biasIPEC(myfun, theta=par2, x=x2, y=y2, tol=1e-20)
resu3

resu4 <- bootIPEC( myfun, x=x2, y=y2, ini.val=ini.val1, target.fun = "RSS", 
               nboot=2000, CI=0.95, fig.opt=TRUE, seed=123 )
resu4

# To calculate skewness
resu5 <- skewIPEC(myfun, theta=par2, x=x2, y=y2, tol=1e-20)
resu5

# End(Not run)

### Example 3

data(shoots)
attach(shoots)

# Choose a species
# 1: Phyllostachys iridescens; 2: Phyllostachys mannii;
# 3: Sinobambusa tootsik; 4: Pleioblastus maculatus
# 'x3' is the vector of the observation times from a specific starting time of growth
# 'y3' is the vector of the aboveground height values of bamboo shoots at 'x3'

ind <- 3
x3 <- time[code == ind]
y3 <- height[code == ind]

# Define the beta sigmoid model (bsm)
bsm <- function(P, x){
P <- cbind(P)
if(length(P) !< 4 ) {stop("The number of parameters should be 4!")}
ropt <- P[1]
topt <- P[2]
tmin <- P[3]
tmax <- P[4]
tailor.fun <- function(x){
  x[x < tmin] <- tmin
  x[x > tmax] <- tmax
  return(x)
}
x <- tailor.fun(x)
ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*
  ((x-tmin)/(topt-tmin))^((topt-tmin)/(tmax-topt))

# Define the simplified beta sigmoid model (simp.bsm)
simp.bsm <- function(P, x, tmin=0){
P <- cbind(P)
ropt <- P[1]
topt <- P[2]
tmax <- P[3]
tailor.fun <- function(x){
  x[x < tmin] <- tmin
  x[x > tmax] <- tmax
  return(x)
}
x <- tailor.fun(x)
ropt*(x-tmin)*(x-2*tmax+topt)/(topt+tmin-2*tmax)*
  ((x-tmin)/(topt-tmin))^((topt-tmin)/(tmax-topt))
}

# For the original beta sigmoid model
ini.val2 <- c(40, 30, 5, 50)
xlab2 <- "Time (d)"
ylab2 <- "Height (cm)"
re0 <- fitIPEC( bsm, x=x3, y=y3, ini.val=ini.val2, xlim=NULL, ylim=NULL,
xlab=xlab2, ylab=ylab2, fig.opt=TRUE,
control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
par3 <- re0$par
par3
re1 <- derivIPEC( bsm, theta=par3, x3[15], method="Richardson",
  method.args=list(eps=1e-4, d=0.11, zero.tol= sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
re1
re2 <- curvIPEC( bsm, theta=par3, x=x3, y=y3, alpha=0.05, method="Richardson",
  method.args=list(eps=1e-4, d=0.11, zero.tol= sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
re2
re3 <- biasIPEC( bsm, theta=par3, x=x3, y=y3, tol= 1e-20 )
re3
## Not run:
re4 <- bootIPEC( bsm, x=x3, y=y3, ini.val=ini.val2, target.fun = "RSS", 
control=list(trace=FALSE, reltol=1e-20, maxit=50000), 
nboot=2000, CI=0.95, fig.opt=TRUE, fold=3.5, seed=123 )
re4

## End(Not run)

re5 <- skewIPEC( bsm, theta=par3, x=x3, y=y3, tol= 1e-20 )
re5

# For the simplified beta sigmoid model
# (in comparison with the original beta sigmoid model)
ini.val7 <- c(40, 30, 50)
RESU0 <- fitIPEC( simp.bsm, x=x3, y=y3, ini.val=ini.val7, 
xlim=NULL, ylim=NULL, xlab=xlab2, ylab=ylab2, 
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
par7 <- RESU0$par
par7
RESU1 <- derivIPEC( simp.bsm, theta=par7, x3[15], method="Richardson", 
method.args=list(eps=1e-4, d=0.11, 
zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
RESU1
RESU2 <- curvIPEC( simp.bsm, theta=par7, x=x3, y=y3, alpha=0.05, method="Richardson", 
method.args=list(eps=1e-4, d=0.11, 
zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2) )
RESU2
RESU3 <- biasIPEC( simp.bsm, theta=par7, x=x3, y=y3, tol= 1e-20 )
RESU3

## Not run:
RESU4 <- bootIPEC( simp.bsm, x=x3, y=y3, ini.val=ini.val7, target.fun = "RSS", 
control=list(trace=FALSE, reltol=1e-20, maxit=50000), 
nboot=2000, CI=0.95, fig.opt=TRUE, fold=3.5, seed=123 )
RESU4

## End(Not run)

RESU5 <- skewIPEC( simp.bsm, theta=par7, x=x3, y=y3, tol= 1e-20 )
RESU5

###############################################
#### Example 4 ###################################################################################
# Data on biochemical oxygen demand (BOD; Marske 1967)
# References:
# Pages 56, 255 and 271 in Bates and Watts (1988)
# 52, 391-396.

.......

### Example 4

...
```r
data(isom)
Y <- isom[,1]
X <- isom[2:4]

# There are three independent variables saved in matrix 'X' and one response variable (Y)
# The first column of 'X' is the vector of partial pressure of hydrogen
# The second column of 'X' is the vector of partial pressure of n-pentane
# The third column of 'X' is the vector of partial pressure of isopentane
# Y is the vector of experimental reaction rate (in 1/hr)

isom.fun <- function(theta, x){
  x1 <- x[,1]
  x2 <- x[,2]
  x3 <- x[,3]
  theta1 <- theta[1]
  theta2 <- theta[2]
  theta3 <- theta[3]
  theta4 <- theta[4]
  theta1*theta3*(x2-x3/1.632) / ( 1 + theta2*x1 + theta3*x2 + theta4*x3 )
}

ini.val8 <- c(35, 0.1, 0.05, 0.2)
cons1 <- fitIPEC( isom.fun, x=X, y=Y, ini.val=ini.val8, control=list( trace=FALSE, reltol=1e-20, maxit=50000 )
par8  <- cons1$par
cons2 <- curvIPEC( isom.fun, theta=par8, x=X, y=Y, alpha=0.05, method="Richardson",
                   method.args=list(eps=1e-4, d=0.11,
                                    zero.tol=sqrt(.Machine$double.eps/7e-7), r=6, v=2))
cons2
cons3 <- biasIPEC( isom.fun, theta=par8, x=X, y=Y, tol= 1e-20 )
cons3

## Not run:
cons4 <- bootIPEC( isom.fun, x=X, y=Y, ini.val=ini.val8, target.fun = "RSS",
                   control=list(trace=FALSE, reltol=1e-20, maxit=50000),
                   nboot=2000, CI=0.95, fig.opt=TRUE, foldId=10000, seed=123 )
cons4

## End(Not run)

cons5 <- skewIPEC( isom.fun, theta=par8, x=X, y=Y, tol= 1e-20 )
```

---

**isom**

*Data on Biochemical Oxygen Demand*
Description
Data on the reaction rate of the catalytic isomerization of n-pentane to isopentane versus the partial pressures of hydrogen, n-pentane, and isopentane.

Usage
data(isom)

Details
There are four columns in the data set:
'y' is the vector of experimental reaction rate (in 1/hr);
'x1' is the vector of partial pressure of hydrogen;
'x2' is the vector of partial pressure of n-pentane;
'x3' is the vector of partial pressure of isopentane.

Note
There were errors about the definitions of 'x2' and 'x3' in page 272 in Bates and Watts (1988). Here, we redefined them according to the paper of Carr (1960).

References

Examples
data(isom)
isom
table(isom[,1])
X <- isom[,2:4]
X
Y

leaves

Leaf Data of *Parrotia subaequalis* (*Hamamelidaceae*)

Description
The data consist of the area, length and width of the leaves of 10 geographical populations of *P. subaequalis* collected in Southern China from July to September, 2016.

Usage
data(leaves)
Details

In the data set, there are four variables: PopuCode, Length, Width and Area. PopuCode is used to save the number codes of different geographical populations; Length is used to save the scanned leaf length data (cm); Width is used to save the scanned leaf width data (cm); Area is used to save the scanned leaf area data (cm squared).

References


Examples
data(leaves)
attach(leaves)
# Choose a geographical population (see Table S1 in Wang et al. [2018] for details)
# 1: AJ; 2: HN; 3: HW; 4: HZ; 5: JD;
# 6: JS; 7: SC; 8: TC; 9: TT; 10: TX
ind <- 1
L <- Length[PopuCode == ind]
W <- Width[PopuCode == ind]
A <- Area[PopuCode == ind]
x <- L*W
fit <- lm(A ~ x-1)
summary(fit)
# Show the leaf areas of the 10 geographical populations
dev.new()
par(mar=c(5,6,2,2))
boxplot(Area~PopuCode, cex=1.5, cex.lab=1.5, cex.axis=1.5,
col="grey70", xlab=expression(bold("Population code")),
ylab=expression(bold(paste("Leaf area (cm"^"2",",",sep=""))),
ylim=c(0, 50), xaxs="i", yaxs="i", las=1)

shoots  Height Growth Data of Bamboo Shoots

Description

The height growth data of four species of bamboos in the Nanjing Forestry University campus in 2016.

Usage
data(shoots)
Details

In the data set, there are four variables: code, species, time and height. code is used to save the number codes of different bamboo species; species is used to save the Latin names of different bamboo species; time is used to save the observation times (d) from a specific starting time of growth, and every bamboo has a different starting time of growth; height is used to save the measured aboveground height values (cm).

code = 1 represents Phyllostachys iridescens, and the starting time (namely time = 0) was defined as 12:00, 3rd April, 2016;
code = 2 represents Phyllostachys mannii, and the starting time (namely time = 0) was defined as 12:00, 4th April, 2016;
code = 3 represents Sinobambusa tootsik, and the starting time (namely time = 0) was defined as 12:00, 18th April, 2016;
code = 4 represents Pleioblastus maculatus, and the starting time (namely time = 0) was defined as 12:00, 29th April, 2016.

References


Examples

data(shoots)
attach(shoots)
# Choose a species
# 1: Phyllostachys iridescens; 2: Phyllostachys mannii;
# 3: Sinobambusa tootsik; 4: Pleioblastus maculatus
ind <- 1
x3 <- time[code == ind]
y3 <- height[code == ind]
dev.new()
par(mar=c(5,5,2,2))
plot(x3, y3, cex=1.5, cex.lab=1.5, cex.axis=1.5, xlab="Time (d)", ylab="Height (cm)"

skewIPEC

Skewness Calculation Function

Description

Calculates the skewness in the estimates of the parameters of a given model.

Usage

skewIPEC( expr, theta, x, y, tol = .Machine$double.eps, method = "Richardson",
method.args = list(eps = 1e-04, d = 0.11,
zero.tol = sqrt(.Machine$double.eps/7e-07), r = 6, v = 2,
show.details = FALSE), side = NULL )
Arguments

- expr: A given parametric model
- theta: Vector of parameters of the model
- x: Vector or matrix of observations of independent variable(s)
- y: Vector of observations of response variable
- tol: The tolerance for detecting linear dependencies in the columns of a matrix for calculating its inverse. See the input argument of tol of the solve function in package base
- method: It is the same as the input argument of method of the hessian function in package numDeriv
- method.args: It is the same as the input argument of method.args of the hessian function in package numDeriv
- side: It is the same as the input argument of side of the jacobian function in package numDeriv

Details

The defined model should have two input arguments: a parameter vector and an independent variable vector or matrix, e.g. myfun <- function(P,x){...}, where P represents the parameter vector and x represents the independent variable vector or matrix.

Let $|g_{1i}|$ be a measure of the skewness of the estimate of the $i$-th parameter. If $|g_{1i}| < 0.1$, the estimator $\hat{\theta}_i$ of parameter $\theta_i$ is very close-to-linear in behavior; if $0.1 \leq |g_{1i}| < 0.25$, the estimator is reasonably close-to-linear; if $|g_{1i}| \geq 0.25$, the skewness is very apparent; if $|g_{1i}| > 1$, the estimator is considerably nonlinear in behavior (Pages 27-28 in Ratkowsky 1990).

Value

- skewness: Calculated skewness

Note

The current function can be applicable to nonlinear models with multiple independent variables.

Author(s)

Peijian Shi, Peter Ridland, David A. Ratkowsky, Yang Li

References


See Also

derivIPEC, hessian in package numDeriv, jacobian in package numDeriv
Examples

#### Example 1
The velocity of the reaction (counts/min^2) under different substrate concentrations
# in parts per million (ppm) (Page 269 of Bates and Watts 1988)
x1 <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56, 1.10, 1.10)
y1 <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)

# Define the Michaelis-Menten (MM) model
MM <- function(theta, x) {
  theta[1]*x / (theta[2] + x)
}

par1 <- c(212.68490865, 0.06412421)
res5 <- skewIPEC( MM, theta=par1, x=x1, y=y1, tol=1e-20)
res5

#### Example 2
Development data of female pupae of cotton bollworm (Wu et al. 2009)
# References:
# properties for describing temperature-dependent developmental rates of insects
# Wu, K.-J., Gong, P.-Y. and Ruan, Y.-M. (2009) Estimating developmental rates of
# Helicoverpa armigera (Lepidoptera: Noctuidae) pupae at constant and

# 'x2' is the vector of temperature (in degrees Celsius)
# 'D2' is the vector of developmental duration (in d)
# 'y2' is the vector of the square root of developmental rate (in 1/d)

x2 <- seq(15, 37, by=1)
D2 <- c(41.24,37.16,32.47,26.22,22.71,19.01,16.79,15.63,14.27,12.48,
       11.3,10.56,9.69,9.14,8.24,8.02,7.43,7.27,7.35,7.49,7.63,7.9,10.03)
y2 <- 1/D2
y2 <- sqrt( y2 )

# Define the square root function of the Lobry-Rosso-Flandrois (LRF) model
sqrt.LRF <- function(P, x) {
  ropt <- P[1]
  Topt <- P[2]
  Tmin <- P[3]
  Tmax <- P[4]
  fun0 <- function(z){
    z[z < Tmin] <- Tmin
    z[z > Tmax] <- Tmax
    return(z)
  }
  x <- fun0(x)
  sqrt( (ropt*(x-Topt)*x*(x-Tmin)^2/((Topt-Tmin)^2*(Topt-Tmax)*(Topt+Tmin-2*x)) )
}

par2 <- c(27.8645858,1.1058746,10.0,29.4278550)
res5 <- skewIPEC( sqrt.LRF, theta=par2, x=x2, y=y2, tol=1e-20)
res5
myfun <- sqrt.LRF
par2 <- c(0.1382926, 33.4575663, 5.5841244, 38.8282021)

# To calculate bias
resu5 <- skewIPEC( myfun, theta=par2, x=x2, y=y2, tol=1e-20 )
resu5

#########################################################################
#### Example 3 ##################################################################
# Weight of cut grass data (Pattinson 1981)
# References:
# Clarke, G.P.Y. (1987) Approximate confidence limits for a parameter function in nonlinear
# Gebremariam, B. (2014) Is nonlinear regression throwing you a curve?
# Response to Herbage on Offer. unpublished M.Sc. thesis, University
# of Natal, Pietermaritzburg, South Africa, Department of Grassland Science.

# 'x4' is the vector of weeks after commencement of grazing in a pasture
# 'y4' is the vector of weight of cut grass from 10 randomly sited quadrants
x4 <- 1:13
y4 <- c(3.183, 3.059, 2.871, 2.622, 2.541, 2.184, 2.110, 2.075, 2.018, 1.903, 1.770, 1.762, 1.550)

# Define the first case of Mitscherlich equation
MitA <- function(P1, x){
  P1[3] + P1[2]*exp(P1[1]*x)
}

# Define the second case of Mitscherlich equation
MitB <- function(P2, x){
}

# Define the third case of Mitscherlich equation
MitC <- function(P3, x, x1=1, x2=13){
  theta1 <- P3[1]
  beta2 <- P3[2]
  beta3 <- P3[3]
  theta2 <- (beta3 - beta2)/(exp(theta1*x2)-exp(theta1*x1))
  theta3 <- beta2/(1-exp(theta1*(x1-x2))) - beta3/(exp(theta1*(x2-x1))-1)
  theta3 + theta2*exp(theta1*x)
}

ini.val3 <- c(-0.1, 2.5, 1)
r0 <- fitIPEC( MitA, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL,
              fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parA <- r0$par
parA
r5 <- skewIPEC(MitA, theta=parA, x=x4, y=y4, tol=1e-20)
R5

ini.val4 <- c(exp(-0.1), log(2.5), 1)
R0 <- fitIPEC(MitB, x=x4, y=y4, ini.val=ini.val3, xlim=NULL, ylim=NULL,
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parB <- R0$par
parB
R5 <- skewIPEC( MitB, theta=parB, x=x4, y=y4, tol=1e-20 )
R5

ini.val6 <- c(-0.15, 2.52, 1.09)
RES0 <- fitIPEC( MitC, x=x4, y=y4, ini.val=ini.val6, xlim=NULL, ylim=NULL,
fig.opt=TRUE, control=list(trace=FALSE, reltol=1e-20, maxit=50000) )
parC <- RES0$par
parC
RES5 <- skewIPEC( MitC, theta=parC, x=x4, y=y4, tol=1e-20 )
RES5

# Example 4 #
# Data on biochemical oxygen demand (BOD; Marske 1967)
# References
# Pages 56, 255 and 271 in Bates and Watts (1988)
# 52, 391-396.

data(isom)
Y <- isom[,1]
X <- isom[,2:4]

# There are three independent variables saved in matrix 'X' and one response variable (Y)
# The first column of 'X' is the vector of partial pressure of hydrogen
# The second column of 'X' is the vector of partial pressure of n-pentane
# The third column of 'X' is the vector of partial pressure of isopentane
# Y is the vector of experimental reaction rate (in 1/hr)

isom.fun <- function(theta, x){
x1 <- x[,1]
x2 <- x[,2]
x3 <- x[,3]
theta1 <- theta[1]
theta2 <- theta[2]
theta3 <- theta[3]
theta4 <- theta[4]
theta1*theta3*(x2-x3/1.632) / ( 1 + theta2*x1 + theta3*x2 + theta4*x3 )
}

par8 <- c(35.92831619, 0.07084811, 0.03772270, 0.16718384)
cons5 <- skewIPEC( isom.fun, theta=par8, x=X, y=Y, tol=1e-20 )
cons5
## Index

*Topic **package**

- IPEC, 31

**AIC**, 2, 9
**aic**, 2, 9
**biasIPEC**, 4, 25
**BIC**, 2, 9
**bic**, 2, 8
**bootIPEC**, 10, 28
**curvIPEC**, 16, 25
**deriv3**, 17
**derivIPEC**, 5, 18, 24, 42
**fitIPEC**, 2, 8, 12, 17, 27
**hessian**, 4, 5, 17, 18, 24, 25, 33, 42
**IPEC**, 31
**isom**, 38
**jacobian**, 4, 5, 17, 18, 25, 33, 42
**leaves**, 39
**nls**, 11, 17, 27
**optim**, 11, 27, 28
**qr**, 17
**rms.curv**, 17, 18, 33
**set.seed**, 11
**shoots**, 40
**skewIPEC**, 25, 41
**solve**, 4, 42