Package ‘OptimModel’

March 12, 2024

Type Package

Version 2.0-1

Date 2024-02-17

Title Perform Nonlinear Regression Using 'optim' as the Optimization Engine

Author Steven Novick [aut, cre]

Maintainer Steven Novick <Steven.Novick@takeda.com>

Depends R (>= 3.0.0),stats, utils, methods, Matrix

Suggests knitr, testthat, rmarkdown, ggplot2

Description A wrapper for 'optim' for nonlinear regression problems; see Nocedal J and Write S (2006, ISBN: 978-0387-30303-1). Performs ordinary least squares (OLS), iterative re-weighted least squares (IRWLS), and maximum likelihood (MLE). Also includes the robust outlier detection (ROUT) algorithm; see Motulsky, H and Brown, R (2006)<doi:10.1186/1471-2105-7-123>.

License GPL-2

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2024-03-12 09:20:02 UTC

R topics documented:

`beta_model` *Beta hook-effect model, gradient, starting values, and back-calculation functions*

Description

Five-parameter hook-effect model for dose-response curve fitting

Usage

```
beta_model(theta, x)
```

Arguments

`theta`	Vector of five parameters: $(e_{\min}, e_{\max}, \log(\delta_1), \log(\delta_2), \log(\delta_3))$. See details.
`x`	Vector of concentrations for the Beta model.

Details

The five-parameter Beta model is given by:

$$y = e_{\min} + e_{\max} \times \exp(log(\beta(\delta_1, \delta_2)) + \delta_1 \times \log(x) + \delta_2 * \log(\mathrm{sc} - x) - (\delta_1 + \delta_2) \times \log(\mathrm{sc})$$

where

$$\beta(\delta_1, \delta_2) = (\delta_1 + \delta_2)^(\delta_1 + \delta_2)/(\delta_1^{\delta_1} \times \delta_2^{\delta_2})$$

and

$$\mathrm{sc} = \max(x) + \delta_3.$$

Note that the Beta model depends on the maximum x value. For a particular data set, this may be set by

attr(theta), "maxX") = max(x).

Value

Let N = length(x). Then

- beta_model(theta, x) returns a numeric vector of length N.
- attr(beta_model, "gradient")(theta, x) returns an N x 5 matrix.
- attr(beta_model, "start")(x, y) returns a numeric vector of length 5 with starting values for

$$(e_{\min}, e_{\max}, \log(\delta_1), \log(\delta_2), \log(\delta_3)).$$

- attr(beta_model, "backsolve")(theta, y) returns a numeric vector of length=length(y) with the first x such that beta_model(theta, x)=y.

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(emin=0, emax=115, ldelta1=-1.5, ldelta2=9, ldelta3=11.5)
y = beta_model(theta, x)  + rnorm( length(x), mean=0, sd=1 )

beta_model(theta, x)
attr(beta_model, "gradient")(theta, x)
attr(beta_model, "start")(x, y)

attr(theta, "maxX") = max(x)
attr(beta_model, "backsolve")(theta, 50)
```

`exp_2o_decay` *Five-parameter second-order exponential decay, gradient, starting values, and back-calculation functions*

Description

Five-parameter second-order exponential decay, gradient, starting values, and back-calculation functions.

Usage

```
exp_2o_decay(theta, x)
```

Arguments

`theta`	Vector of five parameters: (A, B, k1, k2, p). See details.
`x`	Vector of concentrations.

Details

The five-parameter exponential decay model is given by:

$$y = A + B \times P \times \exp(-K1 \times x) + B \times (1 - P) \times \exp(-K2 \times x)$$

The parameter vector is (A, B, k1, k2, p) where $A = \min y$ (min y value), $A + B = \max y$ (max y value), $K1 = \exp(k1)$ which is the shape parameter for first term, $K2 = \exp(k2)$ which is the shape parameter for second term, and $P = 1/(1 + \exp(p))$ which is the proportion of signal from the first term.

Value

Let N = length(x). Then

- exp_2o_decay(theta, x) returns a numeric vector of length N.
- attr(exp_2o_decay, "gradient")(theta, x) returns an N x 5 matrix.
- attr(exp_2o_decay, "start")(x, y) returns a numeric vector of length 5 with starting values for (A, B, k1, k2, p).
- attr(exp_2o_decay, "backsolve")(theta, y) returns a numeric vector of length = length(y).

Author(s)

Steven Novick

See Also

`optim_fit`, `rout_fitter`

Examples

```
set.seed(123L)
x = 2^(-4:4)
theta = c(25, 75, log(3), log(1.2), 1/(1+exp(.7)))
y = exp_2o_decay(theta, x) + rnorm( length(x), mean=0, sd=1 )
attr(exp_2o_decay, "gradient")(theta, x)
attr(exp_2o_decay, "start")(x, y)
attr(exp_2o_decay, "backsolve")(theta, 38)
```

---

exp_decay *Three-parameter exponential decay, gradient, starting values, and back-calculation functions*

Description

Three-parameter exponential decay, gradient, starting values, and back-calculation functions.

Usage

```
exp_decay(theta, x)
```

Arguments

theta	Vector of three parameters: (A, B, k). See details.
x	Vector of concentrations.

Details

The three-parameter exponential decay model is given by:

$$y = A + B \times \exp(-Kx).$$

The parameter vector is (A, B, k) where $A = \min y$ (minimum y value), $A+B = \max y$ (maximum y value), and $K = \exp(k)$ whichi is the shape parameter.

Value

Let N = length(x). Then

- exp_decay(theta, x) returns a numeric vector of length N.
- attr(exp_decay, "gradient")(theta, x) returns an N x 3 matrix.
- attr(exp_decay, "start")(x, y) returns a numeric vector of length 3 with starting values for (A, B, k).
- attr(exp_decay, "backsolve")(theta, y) returns a numeric vector of length=length(y).

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123L)
x = 2^(-4:4)
theta = c(25, 75, log(3))
y = exp_decay(theta, x)  + rnorm( length(x), mean=0, sd=1 )
attr(exp_decay, "gradient")(theta, x)
attr(exp_decay, "start")(x, y)
attr(exp_decay, "backsolve")(theta, 38)
```

---

exp_decay_pl — *Three-parameter exponential decay with initial plateau, gradient, starting values, and back-calculation functions*

Description

Three-parameter exponential decay with initial plateau, gradient, starting values, and back-calculation functions.

Usage

```
exp_decay_pl(theta, x)
```

Arguments

theta	Vector of four parameters: (x0, yMax, yMin, k). See details.
x	Vector of concentrations.

Details

The three-parameter exponential decay with initial plateau model is given by $y = \text{yMax}$ whenever $x \leq 0$ otherwise

$$y = \text{yMin} + (\text{yMax} - \text{yMin}) \times \exp(-K(x - X0)) \text{ if } x > X0,$$

where $X0 = \exp(x0)$ is an inflection point between plateau and exponential decay curve, $\text{yMin} = \min y$ (min response), $\text{yMax} = \max y$ (maximum response), and $K = \exp(k)$ is the shape parameter.

Value

Let N = length(x). Then

- exp_decay_pl(theta, x) returns a numeric vector of length N.
- attr(exp_decay_pl, "gradient")(theta, x) returns an N x 4 matrix.
- attr(exp_decay_pl, "start")(x, y) returns a numeric vector of length 4 with starting values for (x0, yMax, yMin, k).
- attr(exp_decay_pl, "backsolve")(theta, y) returns a numeric vector of length=length(y).

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(100)
x = 2^(-4:4)
theta = c(0.4, 75, 10, log(3))
y = exp_decay_pl(theta, x)  + rnorm( length(x), mean=0, sd=1 )
attr(exp_decay_pl, "gradient")(theta, x)
attr(exp_decay_pl, "start")(x, y)
attr(exp_decay_pl, "backsolve")(theta, 38)
```

---

f2djac *Compute derivative with respect to parameters*

Description

Compute derivative with respect to parameters.

Usage

```
f2djac(Func, theta, ...)
```

Arguments

Func	A function with theta as first argument that returns an n x 1 vector, where n represents the number of observations.
theta	A p x 1 vector of parameters.
...	Other arguments needed for function.

Value

Returns an n x p matrix of derivatives with respect to theta. Computes $\frac{\delta \, Func(\theta,...)}{\delta \, \theta}$, where θ = theta

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
f = function(theta, x){ theta[1] + (theta[2]-theta[1])/(1 + (x/theta[3])^theta[4]) }
theta0 = c(0, 100, .5, 2)
x = 0:10
f2djac(f, theta0, x=x)
```

---

getData *Extract data object from an optim fit*

Description

Extract data object from an `optim_fit` object.

Usage

```
getData(object)
```

Arguments

`object`	object of class `optim_fit`.

Value

Returns a data frame with elements x and y.

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each =4 )
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x) + rnorm( length(x), sd=2 )
fit = optim_fit(c(0, 100, .5, 1), f.model=hill_model, x=x, y=y)
d=getData(fit)
```

`get_se_func` *Compute standard error for a function of model parameter estimates*

Description

Compute standard error for a function of model parameter estimates via the delta method.

Usage

```
get_se_func(object, Func, ..., level=0.95)
```

Arguments

`object`	An optim_fit() object
`Func`	Function that returns a numeric value. See details.
`...`	Other arguments needed for Func.
`level`	Confidence level for confidence interval

Details

Func is of the form function(theta, ...). For example,

Func = function(theta, x){ exp(theta[1])*log(x)/theta[2] }

Value

Returns a data.frame with a single row for the estimated Func call (Est), its standard error (SE), and a confidence interval (lower, upper).

Author(s)

Steven Novick

See Also

`optim_fit`, `rout_fitter`

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each =4 )
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x) + rnorm( length(x), sd=2 )
fit = optim_fit(theta, hill_model, x=x, y=y)

## Get SE for IC20 and IC40
ic.z = function(theta, z){  attr(hill_model, "backsolve")(theta, z)  }
get_se_func(object=fit, Func=ic.z, z=20)
get_se_func(object=fit, Func=ic.z, z=40)
```

gompertz_model *Four-parameter Gompertz model, gradient, starting values, and back-calculation functions*

Description

Four-parameter Gompertz model, gradient, starting values, and back-calculation functions.

Usage

```
gompertz_model(theta, x)
```

Arguments

`theta`	Vector of four parameters: (A, B, m, offset). See details.
`x`	Vector of concentrations for the Gompertz model.

Details

The four parameter Gompertz model is given by:

$$y = A + (B - A) \times \exp(-\exp(m(x - \text{offset}))), \text{ where}$$

$A = \min y$ (minimum y value), $A + (B - A)\exp(-\exp(-m * \text{offset}))$ is the maximum y value, m is the shape parameter, and offset shifts the curve, relative to the concentration x.

Value

Let N = length(x). Then

- gompertz_model(theta, x) returns a numeric vector of length N.
- gompertz_model(hill_model, "gradient")(theta, x) returns an N x 4 matrix.
- attr(gompertz_model, "start")(x, y) returns a numeric vector of length 4 with starting values for (A, B, m, offset).
- attr(gompertz_model, "backsolve")(theta, y) returns a numeric vector of length=length(y).

Author(s)

Steven Novick

See Also

`optim_fit`, `rout_fitter`

Examples

```
set.seed(100)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y = gompertz_model(theta, x)  + rnorm( length(x), mean=0, sd=1 )
attr(gompertz_model, "gradient")(theta, x)
attr(gompertz_model, "start")(x, y)
attr(gompertz_model, "backsolve")(theta, 50)
```

`hill5_model` *Five-parameter Hill model, gradient, starting values, and back-calculation functions*

Description

Five-parameter Hill model, gradient, starting values, and back-calculation functions.

Usage

```
hill5_model(theta, x)
```

Arguments

`theta`	Vector of five parameters: $(e_{\min}, e_{\max}, \text{log.ic50}, m, \text{log.sym})$. See details.
`x`	Vector of concentrations for the five-parameter Hill model.

Details

The five parameter Hill model is given by:

$$y = e_{\min} + \frac{e_{\max} - e + \min}{1 + \exp(m \log(x) - m \text{ log.ic50}))^{\exp(\text{log.sym})}}$$

$e_{\min} = \min y$ (minimum y value), $e_{\max} = \max y$ (maximum y value), log.ic50 $=\log(\text{ic50})$, m is the shape parameter, and log.sym $= \log(\text{symmetry parameter})$.

Note: ic50 is defined such that hill5_model(theta, ic50) $= e_{\min} + (e_{\max} - e_{\min})/2^{\exp(\text{log.sym})}$

Value

Let N = length(x). Then

- hill5_model(theta, x) returns a numeric vector of length N.
- attr(hill5_model, "gradient")(theta, x) returns an N x 5 matrix.
- attr(hill5_model, "start")(x, y) returns a numeric vector of length 5 with starting values for $(e_{\min}, e_{\max}, \text{log.ic50}, m, \text{log.sym})$.
- attr(hill5_model, "backsolve")(theta, y) returns a numeric vector of length=length(y).

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2, log(10))
y = hill5_model(theta, x)  + rnorm( length(x), mean=0, sd=1 )
attr(hill5_model, "gradient")(theta, x)
attr(hill5_model, "start")(x, y)
attr(hill5_model, "backsolve")(theta, 50)
```

---

hill_model — *Four-parameter Hill model, gradient, starting values, and back-calculation functions*

Description

Four-parameter Hill model, gradient, starting values, and back-calculation functions.

Usage

```
hill_model(theta, x)
```

Arguments

theta	Vector of four parameters: $(e_{\min}, e_{\max}, \text{lec50}, m)$. See details.
x	Vector of concentrations for the Hill model.

Details

The four parameter Hill model is given by:

$$y = e_{\min} + \frac{(e_{\max} - e_{\min})}{(1 + \exp(m \log(x) - m * \text{lec50}))}, \text{ where}$$

$e_{\min} = \min y$ (minimum y value), $e_{\max} = \max y$ (maximum y value), $\text{lec50} = \log(\text{ec5})$, and m is the shape parameter. Note: ec50 is defined such that hill.model(theta, ec50) = .5*(emin+ emax).

Value

Let N = length(x). Then

- hill_model(theta, x) returns a numeric vector of length N.
- attr(hill_model, "gradient")(theta, x) returns an N x 4 matrix.
- attr(hill_model, "start")(x, y) returns a numeric vector of length 4 with starting values for $(e_{\min}, e_{\max}, \text{lec50}, m)$.
- attr(hill_model, "backsolve")(theta, y) returns a numeric vector of length=length(y).

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x)  + rnorm( length(x), mean=0, sd=1 )
attr(hill_model, "gradient")(theta, x)
attr(hill_model, "start")(x, y)
attr(hill_model, "backsolve")(theta, 50)
```

---

hill_quad_model *Five-parameter Hill model with quadratic component, gradient, starting values, and back-calculation functions*

Description

Five-parameter Hill model with quadratic component, gradient, starting values, and back-calculation functions.

Usage

```
hill_quad_model(theta, x)
```

Arguments

theta	Vector of five parameters: (A, B, a, b, c). See details.
x	Vector of concentrations for the five-parameter Hill model with quadratic component.

Details

The five parameter Hill model with quadratic component is given by:

$$y = A + \frac{B - A}{(1 + \exp(-(a + bz + cz^2)))}, \text{ where } z = \log(x)$$

$A = \min y$ (minimum y value), $B = \max y$ (maximum y value), (a, b, c) are quadratic parameters for $\log(x)$.

Notes:

1. If $c = 0$, this model is equivalent to the four-parameter Hill model (hill.model).

2. The ic50 is defined such that $a + bz + cz^2 = 0$. If the roots of the quadratic equation are real, then the ic50 is given by $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Value

Let N = length(x). Then

- hill_quad_model(theta, x) returns a numeric vector of length N.
- attr(hill_quad_model, "gradient")(theta, x) returns an N x 5 matrix.
- attr(hill_quad_model, "start")(x, y) returns a numeric vector of length 5 with starting values for (A, B, a, b, c).

  If the quadratic roots are real-valued, attr(hill_quad_model, "backsolve")(theta, y) returns a numeric vector of length=2.

Author(s)

Steven Novick

See Also

`optim_fit`, `rout_fitter`

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=3 )      ## Dose
theta = c(0, 100, 2, 1, -0.5)          ## Model parameters
y = hill_quad_model(theta, x) + rnorm( length(x), mean=0, sd=5 )

## Generate data
hill_quad_model(theta, x)
attr(hill_quad_model, "gradient")(theta, x)
attr(hill_quad_model, "start")(x, y)
attr(hill_quad_model, "backsolve")(theta, 50)
```

`hill_switchpoint_model`

Five-parameter Hill model with switch point component, gradient, starting values, and back-calculation functions

Description

Five-parameter Hill model with switch point component, gradient, starting values, and back-calculation functions.

Usage

```
hill_switchpoint_model(theta, x)
```

Arguments

`theta`	Vector of five parameters: $(e_{\min}, e_{\max}, \mathrm{lec50}, m, \mathrm{lsp})$. See details.
`x`	Vector of concentrations for the five-parameter Hill model with switch point component.

Details

The five parameter Hill model with switch point component is given by:

$$y = e_{\min} + \frac{e_{\max} - e_{\min}}{1 + \exp(m \times f(\exp(\mathrm{lsp}), x) \times (\log(x) - \mathrm{log.ic50}))}, \text{ where}$$

$e_{\min} = \min y$ (minimum y value), $e_{\max} = \max y$ (maximum y value), $\mathrm{log.ic50} = \log(\mathrm{ic50})$, m is the shape parameter, and $f(s, x)$ is the switch point function.

The function $f(s, x) = (s - x)/(s + x) = 2/(1 + x/s) - 1$. This function is constrained to be between -1 and +1 with $s > 0$.

Notes:

1. At $x = 0$, $f(s, x) = 1$, which reduces to hill_model(theta[1:4], 0).

2. The `hill_switchpoint_model`() is more flexible compared to `hill_quad_model`().

3. When the data does not contain a switchpoint, then lsp should be a large value so that $f(\exp(\mathrm{lsp}), x)$ will be near 1 for all x.

Value

Let N = length(x). Then

- hill_switchpoint_model(theta, x) returns a numeric vector of length N.
- attr(hill_switchpoint_model, "gradient")(theta, x) returns an N x 5 matrix.

- attr(hill_switchpoint_model, "start")(x, y) returns a numeric vector of length 5 with starting values for $(e_{\min}, e_{\max}, \text{lec50}, m, \text{lsp})$.

  Because `hill_switchpoint_model()` can be fitted to biphasic data with a hook-effect, attr(hill_switchpoint_model, "backsolve")(theta, y0) returns the first x that satisfies y0=hill_switchpoint_model(theta, x)

Author(s)

Steven Novick

See Also

`optim_fit`, `rout_fitter`

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=3 )      ## Dose
## Create model with no switchpoint term
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x) + rnorm( length(x), mean=0, sd=5 )

## fit0 and fit return roughly the same r-squared and sigma values.
## Note that BIC(fit0) < BIC(fit), as it should be.
fit0 = optim_fit(NULL, hill_model, x=x, y=y)
fit = optim_fit(c(coef(fit0), lsp=0), hill_switchpoint_model, x=x, y=y)
fit = optim_fit(NULL, hill_switchpoint_model, x=x, y=y)

## Generate data from hill.quad.model() with a biphasic (hook-effect) profile
set.seed(123L)
theta = c(0, 100, 2, 1, -0.5)          ## Model parameters
y = hill_quad_model(theta, x) + rnorm( length(x), mean=0, sd=5 )

## fit.qm and fit.sp return nearly identical fits
fit.qm = optim_fit(theta, hill_quad_model, x=x, y=y)
fit.sp = optim_fit(NULL, hill_switchpoint_model, x=x, y=y, ntry=50)

plot(log(x+0.01), y)
lines(log(x+0.01), fitted(fit.qm))
lines(log(x+0.01), fitted(fit.sp), col="red")

## Generate data from hill.switchback.model()
set.seed(123)
theta = c(0, 100, log(0.25), -3, -2)
y = hill_switchpoint_model(theta, x) + rnorm( length(x), mean=0, sd=5 )
plot( log(x+0.01), y )

## Note that this model cannot be fitted by hill.quad.model()
fit = optim_fit(NULL, hill_switchpoint_model, x=x, y=y, ntry=50,
       start.method="fixed", until.converge=FALSE)
pred = predict(fit, x=exp(seq(log(0.01), log(16), length=50)), interval='confidence')

plot(log(x+0.01), y, main="Fitted curve with 95% confidence bands")
```

```
lines(log(pred[,'x']+0.01), pred[,'y.hat'], col='black')
lines(log(pred[,'x']+0.01), pred[,'lower'], col='red', lty=2)
lines(log(pred[,'x']+0.01), pred[,'upper'], col='red', lty=2)


## Other functions
hill_switchpoint_model(theta, x)
attr(hill_switchpoint_model, "gradient")(theta, x)
attr(hill_switchpoint_model, "start")(x, y)
attr(hill_switchpoint_model, "backsolve")(theta, 50)
```

`linear_model` *Linear model, gradient, and starting values*

Description

Linear model, gradient, and starting values.

Usage

```
linear_model(theta, x)
```

Arguments

`theta`	Vector of model parameters intercept and slope. See details.
`x`	Matrix, possibly from `model.matrix`().

Details

The linear model is given by:

$$y = x * \theta, \text{ where}$$

x is a `matrix`, possibly generated from `model.matrix`() θ is a vector of linear parameters

Value

Let N = nrow(x). Then

- linear_model(theta, x) returns a numeric vector of length N.
- attr(linear_model, "gradient")(theta, x) returns x.
- attr(linear_model, "start")(x, y) returns solve(t(x) * x) * t(x) * y

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

Examples

```
set.seed(123)
d = data.frame( Group=factor(rep(LETTERS[1:3], each=5)), age=rnorm(15, mean=20, sd=3) )
d$y = c(80, 100, 120)[unclass(d$Group)] - 3*d$age + rnorm(nrow(d), mean=0, sd=5)

X = model.matrix(~Group+age, data=d)  ## This is the "x" in linear.model()
theta = c(80, 20, 40, -3)   ## Intercept, effect for B, effect for C, slope for age
linear_model(theta, x=X)
attr(linear_model, "gradient")(theta, x=X)
attr(linear_model, "start")(x=X, y=d$y)
```

---

nlogLik_cauchy *Negative log-likelihood function for Cauchy regression*

Description

The negative log-likelihood function for Cauchy regression, for use with `rout_fitter`. Usually not called by the end user.

Usage

```
nlogLik_cauchy(theta, x, y, f.model, lbs)
```

Arguments

`theta`	Parameters for f.model and an extra parameter for the scale parameter; e.g., f.model=hill.model
`x`	Explanatory variable(s). Can be vector, matrix, or data.frame
`y`	Response variable.
`f.model`	Name of mean model function.
`lbs`	Logical. lbs = log both sides. See details.

Details

The function provides the negative log-likelihood for Cauchy regression

Let mu = f.model(theta[1:(p-1)], x) and sigma = exp(theta[p]), where p = number of parameters in theta.

The Cauchy likelihood is

$$L = \prod \frac{1}{\pi\sigma}(1 + (\frac{y_i - \mu_i}{\sigma})^2)^{-1}$$

.

The function returns $\log(L)$.

If lbs == TRUE, then μ is replaced with $\log(mu)$.

Value

Returns a single numerical value.

Author(s)

Steven Novick

See Also

rout_fitter

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(emin=0, emax=100, lec50=log(.5), m=2)
y = hill_model(theta, x)  + rnorm( length(x), mean=0, sd=2 )

theta1 = c(theta, lsigma=log(2))
nlogLik_cauchy(theta1, x=x, y=y, f.model=hill_model, lbs=FALSE)

  ## Cauchy regression via maximum likelihood
optim( theta1, nlogLik_cauchy, x=x, y=y, f.model=hill_model, lbs=FALSE )
```

---

optim_fit *Fit Model with optim*

Description

Fit nonlinear model using the optim function in the **stats** library. This defaults to Ordinary Least Squares (OLS) The other options are Iterative Reweighted Least Squares (IRWLS), and Maximum Likelihood Estimator (MLE).

Usage

```
optim_fit( theta0, f.model, gr.model=NULL, x, y, wts,
            fit.method=c("ols", "irwls", "mle"),
            var.method=c("hessian", "normal", "robust"),
            phi0=NULL, phi.fixed=TRUE, conf.level = 0.95, tol = 1e-3,
            n.start=1000, ntry=1, start.method=c("fixed", "random"),
            until.converge=TRUE, check.pd.tol = 1e-8, ...)
```

```
robust_fit(theta0, f.model, gr.model=NULL, x, y, wts=c("huber", "tukey"),
    var.method=c("hessian", "normal", "robust"), conf.level=.95, tol=1e-3, ...)
```

Arguments

`theta0`	starting values. Alternatively, if given as NULL, theta0 can be computed within `optim.fit()` if a starting values function is supplied as attr(f.model, "start"), as a function of x and y.
`f.model`	Name of mean model function.
`gr.model`	If specified, name of the partial derivative of f.model with respect to its parameter argument. If not specified, `f2djac` will approximate the derivative. Alternatively, the gradient of f.model can be specified as attr(f.model, "gradient")
`x`	Explanatory variable(s). Can be `vector`, `matrix`, or `data.frame`
`y`	Response variable.
`fit.method`	"ols" for ordinary least squares, "irwls" for iterative re-weighted least squares, "mle" for maximum likelihood.
`wts`	For `optim.fit()`, can be a numeric vector or a function. Functions supplied in the library are `weights_varIdent`, `weights_tukey_bw`, `weights_huber`, `weights_varExp`, `weights_varPower`, and `weights_varConstPower`. For `robust_fit()`, choices are a character string of "huber" for `weights_huber` and "tukey" for `weights_tukey_bw`
`var.method`	Method to compute variance-covariance matrix of estimated model parameters. Choices are "hessian" to use the hessian inverse, "normal" to use the so-called 'folk-lore' theorem estimator, and "robust" to use a sandwich-variance estimator. When `fit.method = "mle"`, `var.method = "hessian"` and cannot be overridden.
`phi0`	Not meaningful for `fit.method = "ols"`. Starting value(s) for variance parameters (for weights).
`phi.fixed`	Not meaningful for `fit.method = "ols"`. If set to `TRUE`, the variance parameter(s) remain fixed at the given starting value, phi0. Otherwise, the variance parameter(s) are estimated.
`conf.level`	Confidence level of estimated parameters.
`tol`	Tolerance for `optim` algorithm.
`n.start`	Number of starting values to generate (see details).
`ntry`	Maximum number of calls to `optim.fit()`.
`start.method`	Parameter
`until.converge`	Logical (`TRUE`/`FALSE`) indicating when algorithm should stop.
`check.pd.tol`	absolute tolarence for determing whether a matrix is positive definite. Refer to `test_fit`.
`...`	Other arguments to passed to `optim`. See ?`optim`. For example, lower=, upper=, method=

Details

`optim_fit()` is a wrapper for `stats::optim()`, specifically for non-linear regression. The Default algorithm is ordinary least squares (ols) using method="BFGS" or "L-BFGS-B", if lower= and upper= are specified. These can easily be overridden.

The assumed model is:

$$y = \text{f.model}(\theta, x) + g(\theta, \phi, x)\sigma\epsilon, \text{ where } \epsilon \sim N(0, 1).$$

Usually the function $g(\cdot) = 1$.

With the exception of weights.tukey.bw and weights.huber, the weights functions are equivalent to $g(\theta, \phi, x)$.

Note that `robust_fit()` is a convenience function to simplify the model call with `fit.method = "irwls"`, `phi0 = NULL`, and `phi.fixed = TRUE`.

Algorithms:

1. OLS. Minimize sum((y - f.model(theta, x))^2)

2. IRWLS. Minimize sum(g(theta, phi, x)*(y - f.model(theta, x))^2), where g(theta, phi, x) act as weights. See section on Variance functions below for more details on $g(\cdot)$.

3. MLE. Minimize the -log(Likelihood). See section on Variance functions below for more details on $g(\cdot)$.

Variance functions:

Weights are given by some variance function. Some common variance functions are supplied.

See `weights_varIdent`, `weights_varExp`, `weights_varPower`, `weights_varConstPower`, `weights_tukey_bw`, `weights_huber`.

User-specified variance functions can be provided for weighted regression.

Value

The returned object is a list with the following components and attributes:

coefficients = estimated model coefficients

value, counts, convergence = returns from optim()

message = character, indicating problem if any. otherwise=NULL

hessian = hessian matrix of the objective function (e.g., sum of squares)

fit.method = chosen fit.method (e.g., "ols")

var.method = chosen var.method (e.g., "hessian")

call = optim.fit() function call

fitted, residuals = model mean and model residuals

r.squared, bic = model statistics

df = error degrees of freedom = N - p, where N = # of observations and p = # of parameters

dims = list containing the values of N and p

sigma = estimated standard deviation parameter

varBeta = estimated variance-covariance matrix for the coefficients

beta = data.frame summary of the fit

attr(object, "weights") = weights

attr(object, "w.func") = weights model for the variance

attr(object, "var.param") = variance parameter values

attr(object, "converge.pls") = logical indicating if IRWLS algorithm converged.

Author(s)

Steven Novick

See Also

optim_fit, rout_outlier_test, beta_model, exp_2o_decay, exp_decay_pl, gompertz_model, hill_model, hill5_model, hill_quad_model, hill_switchpoint_model, linear_model, weights_huber, weights_tukey_bw, weights_varConstPower, weights_varExp, weights_varIdent, weights_varPower

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y1 = hill_model(theta, x) + rnorm( length(x), sd=2 )
y2 = hill_model(theta, x) + rnorm( length(x), sd=.1*hill_model(theta, x) )
wts = runif( length(y1) )
fit1=optim_fit(theta, hill_model, x=x, y=y1)
fit2=optim_fit(theta, hill_model, x=x, y=y1, wts=weights_varIdent)
fit3=optim_fit(theta, hill_model, x=x, y=y1, wts=wts)
fit4=optim_fit(theta, hill_model, attr(hill_model, "gradient"), x=x, y=y1, wts=wts)

fit5=optim_fit(NULL, hill_model, x=x, y=y2, wts=weights_varPower, fit.method="irwls")
fit6=optim_fit(theta, hill_model, x=x, y=y2, wts=weights_varPower, fit.method="mle")

fit7=optim_fit(theta, hill_model, x=x, y=y2, wts=weights_varPower, fit.method="irwls",
               phi0=0.5, phi.fixed=FALSE)
fit8=optim_fit(theta, hill_model, x=x, y=y2, wts=weights_varPower, fit.method="mle",
              phi0=0.5, phi.fixed=FALSE)

fit9a=optim_fit(theta, hill_model, x=x, y=y1, wts=weights_tukey_bw, fit.method="irwls",
             phi0=4.685, phi.fixed=TRUE)

fit9b=robust_fit(theta, hill_model, x=x, y=y1, wts="tukey")
```

optim_weights *Weight functions for optim_fit*

Description

Weight functions for `optim_fit`. May be used when `fit.method=="irwls"` or `fit.method=="mle"`. Generally, not called by the user.

Usage

```
weights_varIdent(phi, mu)
weights_varExp(phi, mu)
weights_varPower(phi, mu)
weights_varConstPower(phi, mu)
weights_tukey_bw(phi = 4.685, resid)
weights_huber(phi=1.345, resid)
```

Arguments

`phi`	Variance parameter(s)
`mu`	Vector of means
`resid`	Vector of model residuals

Details

- `weights_varIdent` returns a vector of ones.
- `weights_varExp` returns $\exp(\phi * \mu)$
- `weights_varPower` returns $|\mu|^\phi$
- `weights_varConstPower` returns $\phi_1 + |\mu|^{\phi_2}$ where $\phi_i = \phi$[i]
- `weights_tukey_bw` is a Tukey bi-weight function. Let

  $$r = \frac{|\text{resid}|}{\text{mad(resid,center=TRUE)}}.$$

  Then this function returns

  $$\left(1 - \left(\frac{r}{\phi}\right)^2\right)^2 \text{ whenever } r <= \phi \text{ and } 0 o.w.$$

  For this the user should use `phi.fixed=TRUE` in the `optim_fit` function.
- `weights_huber` is a Huber weighting function that returns $\min(1, \phi/r)$, where $r = |\text{resid}|/\text{sig}$ and $\text{sig} = \text{mad}(\text{resid}, \text{center} = \text{TRUE})$. For this the user should use `phi.fixed = TRUE` in the `optim_fit` function.

Value

A vector of numeric weights.

Author(s)

Steven Novick

See Also

optim_fit, rout_fitter

---

predict.optim_fit *Predicted values for optim.fit objects*

Description

Provides predicted values, standard errors, confidence intervals and prediction intervals for `optim_fit` objects.

Usage

```
    ## S3 method for class 'optim_fit'
predict(object, x, se.fit=FALSE,
    interval=c("none", "confidence", "prediction"), K = 1, level = 0.95,...)
```

Arguments

`object`	An object resulting from `optim_fit`.
`x`	If supplied, a vector, data.frame, or matrix of Explanatory variables.
`se.fit`	Logical. Should standard errors be returned? Requires that 'x' is supplied.
`interval`	If equal to 'confidence', returns a 100*level% confidence interval for the mean response. If equal to 'prediction', returns a 100*level% prediction interval for the mean of the next K observations. Requires that 'x' is supplied.
`K`	Only used for prediction interval. Number of observations in the mean for the prediction interval.
`level`	Confidence/prediction interval level.
`...`	mop up additional arguments.

Value

Returns a vector (if interval='none'). Otherwise returns a data.frame with possible columns 'x', 'y.hat', 'se.fit', 'lower', and 'upper'.

Author(s)

Steven Novick

See Also

optim_fit

Examples

```
set.seed(123L)

x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y1 = hill_model(theta, x) + rnorm( length(x), sd=2 )


fit1=optim_fit(theta, hill_model, x=x, y=y1)
fitted(fit1)
predict(fit1)
predict(fit1, x=x)
predict(fit1, x=seq(0, 1, by=.1), se.fit=TRUE)
predict(fit1, x=seq(0, 1, by=.1), interval="conf")
predict(fit1, x=seq(0, 1, by=.1), interval="pred")
```

---

print.optim_fit *Prints optim_fit objects*

Description

Provides a nice printed summary of optim_fit objects.

Usage

```
  ## S3 method for class 'optim_fit'
print(x, digits=4,...)
```

Arguments

x	An object resulting from optim_fit().
digits	Number of digits to print for output.
...	other arguments not used by this function.

Value

No Return Value.

Author(s)

Steven Novick

See Also

optim_fit

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y1 = hill_model(theta, x) + rnorm( length(x), sd=2 )

fit1=optim_fit(theta, hill_model, x=x, y=y1)
print(fit1)
fit1
```

---

residuals.optim_fit *Residuals for optim.fit objects*

Description

Provides raw and studentized residuals for `optim_fit` objects.

Usage

```
    ## S3 method for class 'optim_fit'
residuals(object, type=c("raw", "studentized"),...)
```

Arguments

`object`	An object resulting from `optim.fit`().
`type`	'raw' or 'studentized' residuals.
`...`	mop up additional arguments.

Value

Returns a numeric vector.

Author(s)

Steven Novick

See Also

optim_fit

Examples

```
set.seed(123)

x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y1 = hill_model(theta, x) + rnorm( length(x), sd=2 )

fit1=optim_fit(theta, hill_model, x=x, y=y1)
residuals(fit1)
residuals(fit1, type="s")
```

`rout_fitter` *Fit Model with ROUT*

Description

The `rout_fitter` method in R fits nonlinear regression using the ROUT method as described in the reference below. The starting point is to fit a robust nonlinear regression approach assuming the Lorentzian distribution. An adaptive method then proceeds. The False Discovery Rate is used to detect outliers and the method fits in an iterative fashion.

The `rout_fitter` function provides a wrapper algorithm to the `optim` function in package **stats**.

Usage

```
rout_fitter(theta0 = NULL, f.model, x, y, lbs = FALSE, ntry = 0, tol = 1e-3,
            Q = 0.01, check.pd.tol = 1e-8, ...)
```

Arguments

`theta0`	a numeric vector of starting values. Alternatively, if given as NULL, `theta0` can be computed within [rout.fitter()] if a starting values function is supplied as attr(f.model, "start"), as a function of `x` and `y`. If `theta0` is user supplied, the last entry of `theta0` should be log(sigma), where sigma = residual standard deviation. Otherwise, log(sigma) will be estimated and appended to the results from attr(f.model, "start").
`f.model`	Name of mean model function. See detials below.
`x`	Explanatory variable(s). Can be a numeric `vector`, a `matrix`, or a `data.frame`.
`y`	a numeric `vector` for the response variable.
`lbs`	Parmeter
`ntry`	Parmeter
`tol`	Tolerance for `optim` algorithm.
`Q`	The test size for ROUT testing.
`check.pd.tol`	absolute tolarence for determing whether a matrix is positive definite. Refer to `test_fit`.
`...`	Other arguments to passed to `optim`. See ?optim. For example, lower=, upper=, method=

Details

[rout.fitter()] is a wrapper for [optim()], specifically for Cauchy likelihood linear and non-linear regression. The Default algorithm uses method="BFGS" or "L-BFGS-B", if lower= and upper= arguments are specified. These can easily be overridden using the "...".

The assumed model is:

$$y = \text{f.model}(theta, x) + \sigma * \epsilon, \text{ where } \epsilon \sim Cauchy(0, 1).$$

After the Cauchy likelihood model is fitted to data, the residuals are interrogated to determine which observations might be outliers. An FDR correction is made to p-values (for outlier testing) through the p.adjust(method="fdr") function of the **stats** package.

The package supports several mean model functions for the f.model parameter. This includes `beta_model`, `exp_2o_decay`, `exp_decay_pl`, `gompertz_model`, `hill_model`, `hill_quad_model`, `hill_switchpoint_model`, `hill5_model` and `linear_model`.

Value

An object with class "rout_fit" is returned that gives a list with the following components and attributes:

par = estimated Cauchy model coefficients. The last term is log(sigma)

value, counts, convergence = returns from [optim()]

message = character, indicating problem if any. otherwise = NULL

hessian = hessian matrix of the objective function (e.g., sum of squares)

Converge = logical value to indicate hessian is positive definite

call = [rout.fitter()] function call

residuals = model residuals

rsdr = robust standard deviation from ROUT method

sresiduals = residuals/rsdr

outlier = logical vector. TRUE indicates observation is an outlier via hypothesis testing with unadjust p-values.

outlier.adj = logical vector. TRUE indicates observation is an outlier via hypothesis testing with FDR-adjust p-values.

`attr`(object, "Q") = test size for outlier detection

Author(s)

Steven Novick

References

Motulsky, H.J. and Brown, R.E. (2006) Detecting Outliers When Fitting Data with Nonlinear Regression: A New Method Based on Robust Nonlinear Regression and the False Discovery Rate. BMC Bioinformatics, 7, 123.

See Also

optim_fit, rout_outlier_test, beta_model, exp_2o_decay, exp_decay_pl, gompertz_model, hill_model, hill5_model, hill_quad_model, hill_switchpoint_model, linear_model

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x) + rnorm( length(x), sd=2 )

rout_fitter(NULL, hill_model, x=x, y=y)
rout_fitter(c( theta, log(2) ), hill_model, x=x, y=y)

ii = sample( 1:length(y), 2 )
y[ii] = hill_model(theta, x[ii]) + 5.5*2 + rnorm( length(ii), sd=2 )

rout_fitter(c( theta, log(2) ), hill_model, x=x, y=y, Q=0.01)
rout_fitter(c( theta, log(2) ), hill_model, x=x, y=y, Q=0.05)
rout_fitter(c( theta, log(2) ), hill_model, x=x, y=y, Q=0.0001)

## Use optim method="L-BFGS-B"
rout_fitter(NULL, hill_model, x=x, y=y, Q=0.01, lower=c(-2, 95, NA, 0.5), upper=c(5, 110, NA, 4) )
```

---

rout_outlier_test *ROUT Outlier Testing*

Description

Perform ROUT outlier testing on `rout.fitter` object.

Usage

```
rout_outlier_test(fit, Q = 0.01)
```

Arguments

fit	A 'rout.fitter' object from the `rout_fitter` function.
Q	Test size for ROUT outlier detection.

Details

`rout_outlier_test()` is typically called from `rout_fitter()`, but may also be called directly by the user.

Value

outlier = logical vector. TRUE indicates observation is an outlier via hypothesis testing with unadjust p-values.

outlier.adj = logical vector. TRUE indicates observation is an outlier via hypothesis testing with FDR-adjust p-values.

attr(object, "Q") = test size for outlier detection

Author(s)

Steven Novick

See Also

rout_fitter

Examples

```
set.seed(123L)

x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y = hill_model(theta, x) + rnorm( length(x), sd=2 )

ii = sample( 1:length(y), 2 )
y[ii] = hill_model(theta, x[ii]) + 5.5*2 + rnorm( length(ii), sd=2 )

fit = rout_fitter(c( theta, log(2) ), hill_model, x=x, y=y, Q=0.01)
rout_outlier_test(fit, Q=0.001)
```

---

test_fit *Test Fit Parameters*

Description

Test if estimated parameters optimize the regression system (i.e., minimize sums of squares, maximize likelihood).

Usage

```
test_fit(obj, check.pd.tol = 1e-8)
```

Arguments

obj	An optim_fit object
check.pd.tol	absolute tolarence for determing whether a matrix is positive definite.

Details

The function checks if `optim` convergence has been reached and also checks if the cholesky decomposiition of the Hessian matrix is positive definite. The latter is an indication that optimization has been reached. Sometimes the `chol` decomposition check doesn't work and to enforce that constriant we use the `check.pd.tol` to make sure all the eigenvalues are larger than this minimum threshhold.

Value

Returns a TRUE or FALSE as to whether or not hessian component of the object is Positive Definite.

Author(s)

Steven Novick

See Also

`optim_fit`

Examples

```
set.seed(123L)
x = rep( c(0, 2^(-4:4)), each=4 )
theta = c(0, 100, log(.5), 2)
y1 = hill_model(theta, x) + rnorm( length(x), sd=2 )
wts = runif( length(y1) )
fit1=optim_fit(theta, hill_model, x=x, y=y1)
test_fit(fit1)
```

Index