Package 'profileModel'

<div align="center">

February 6, 2026

</div>

Type Package

Title Profiling Inference Functions for Various Model Classes

Version 0.6.2

URL https://github.com/ikosmidis/profileModel

BugReports https://github.com/ikosmidis/profileModel/issues

Description Provides tools that can be used to calculate, evaluate, plot and use for inference the profiles of *arbitrary* inference functions for *arbitrary* 'glm'-like fitted models with linear predictors. More information on the methods that are implemented can be found in Kosmidis (2008) <https://www.r-project.org/doc/Rnews/Rnews_2008-2.pdf>.

License GPL (>= 2)

Depends R (>= 2.6.0)

Suggests MASS, gnm

RoxygenNote 7.1.1

NeedsCompilation no

Author Ioannis Kosmidis [aut, cre] (ORCID: <https://orcid.org/0000-0003-1556-0302>)

Maintainer Ioannis Kosmidis <ioannis.kosmidis@warwick.ac.uk>

Repository CRAN

Date/Publication 2026-02-06 17:00:02 UTC

Contents

---

confintModel   *Confidence intervals for model parameters*

Description

Computes confidence intervals for one or more parameters in a fitted model, based on the profiles of a specified objective.

Usage

```
confintModel(fitted, quantile = qchisq(0.95, 1), verbose = TRUE,
             endpoint.tolerance = 1e-3, max.zoom = 100,
             zero.bound = 1e-08, stepsize = 0.5, stdn = 5,
             gridsize = 20, scale = FALSE, which = 1:length(coef(fitted)),
             objective = stop("'objective' is missing."),
             agreement = TRUE, method = "smooth",
             n.interpolations = 100, ...)


## S3 method for class 'profileModel'
profConfint(prof, method = "smooth",
endpoint.tolerance = 1e-3, max.zoom = 100, n.interpolations = 100,
verbose = FALSE, ...)


## S3 method for class 'profileModel'
profZoom(prof, max.zoom = 100, endpoint.tolerance = 1e-03,
        verbose = FALSE, ...)

## S3 method for class 'profileModel'
profSmooth(prof, n.interpolations = 100, ...)
```

Arguments

fitted	a [glm](#)-like fitted object with **linear predictor** (see Details of [profileModel](#) for the methods that have to be supported by fitted).
prof	a "profileModel" object with non-NULL quantile.
quantile	The quantile to be used for the construction of the confidence intervals. The default is qchisq(0.95, 1).
verbose	if TRUE (default) progress indicators are printed during the progress of calculating the confidence intervals.
endpoint.tolerance	
	the tolerance on the absolute difference of the value of the profile at the endpoints from the quantile used. Only relevant when confidence intervals are constructed via the "profZoom" method (see Details).

max.zoom	the maximum number of iterations that the binary search algorithm will take towards the achievement of endpoint.tolerance.
zero.bound	same as in [profileModel].
stepsize	same as in [profileModel].
stdn	same as in [profileModel].
gridsize	same as in [profileModel].
scale	same as in [profileModel].
which	for which parameters should the confidence intervals be calculated?
objective	same as in [profileModel].
agreement	same as in [profileModel].
method	the method to be used for the calculation of the confidence intervals. Possible values are "smooth", which is the default, and "zoom" (see Details).
n.interpolations	
	if method="smooth" the number of interpolations to be used for spline smoothing. The default is 100.
...	for confintModel, further arguments passed to the specified objective. For the methods profZoom, profSmooth and profConfint, further arguments passed to or from other functions.

Details

The confidence intervals methods refer to convex objectives. Objectives that result in disjoint confidence regions are not currently supported.

When the profile object is available and was called with the specification of the appropriate quantile then profConfint should be used. confintModel applies directly to the fitted model and calls profileModel.

When method="zoom" the profZoom method is applied to the "profileModel" object. When method="smooth" the profSmooth method is applied to the "profileModel" object.

The profZoom method relies on a binary search and can find the endpoints of the confidence intervals for a pre-specified tolerance for the absolute difference of the value of the profile at each endpoint from the quantile used. It is a computationally intensive method and is useful in cases where the estimate is infinite and in coverage related simulations.

The profSmooth method, fits a smoothing spline on the points specified by the "profileModel" object and then interpolates the endpoints of the confidence intervals at the specified quantile. It is much faster than profZoom and can safely be used in cases where the profiled objective is nearly quadratic in shape, but could be misleading otherwise.

Both methods can report an infinite endpoint. The detection is based on the intersects component of the "profileModel" object.

profConfint is a wrapper method that collects the capabilities of profZoom and profSmooth.

profSmooth, profZoom and profConfint use the quantile that comes with the "profileModel" object prof.

Value

All the functions return a matrix with columns the endpoints of the confidence intervals for the specified (or profiled) parameters.

Additionally, `confintModel` and `profConfint` have an attribute carrying the name of the fitted object and the name of the `"profileModel"` object, respectively.

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

See Also

confint, profileModel.

Examples

```
## Not run:
## Begin Example: quasi likelihood estimation.
## Incidence of leaf-blotch on barley
## McCullagh and Nelder (1989), pp. 328--332
library(gnm)
data(barley)
logitModel <- glm(y ~ site + variety, family = wedderburn, data = barley)
profQuasi <- profileModel(logitModel, objective = "ordinaryDeviance",
                          quantile=qchisq(0.95, 1),
                          which = paste("variety",c(2:9,"X"),sep=""))
# very accurate confidence intervals (with endpoints accurate up to 10
# decimals) for the variety parameters using profConfint with
# method="zoom":
c1 <- profConfint(profQuasi, endpoint.tolerance = 1e-10, maxit = 100,
                  method="zoom" )
# confidence intervals using smoothing:
c2 <- profConfint(profQuasi, method="smooth" )
# c2 has accurate endpoints at least up to four decimals
# this is because of the quadratic shape of the profiles
plot(profQuasi, cis = c1)
plot(profQuasi, cis = c1, signed = TRUE, print.grid.points = TRUE)
# pairs plot
pairs(profQuasi)
# Notice the direction of the pairs plots. The fact that the
# correlations among the estimates are 1/2 is clear.

# profiling using the Rao score statistic
# This can be used as deviance in cases were a quasi likelihood does not
# exist.
profRao <- update(profQuasi, objective = "RaoScoreStatistic",
                  X = model.matrix(logitModel))
## End Example

## End(Not run)
```

```
objectives-profileModel
```
Objectives to be profiled

Description

Objectives to be used in **profileModel**.

Usage

```
ordinaryDeviance(fm, dispersion = 1)

RaoScoreStatistic(fm, X, dispersion = 1)
```

Arguments

fm	the **restricted** fit.
X	the model matrix of the fit on all parameters.
dispersion	the dispersion parameter.

Details

The objectives used in **profileModel** have to be functions of the **restricted** fit. Given a fitted object, the restricted fit is an object resulted by restricting a parameter to a specific value and then estimating the remaining parameters. Additional arguments could be used and are passed to the objective matching the ... in profileModel or in other associated functions. An objective function should return a scalar which is the value of the objective at the restricted fit.

The construction of a custom objective should follow the above simple guidelines (see also Example 3 in profileModel and the sources of either ordinaryDeviance or RaoScoreStatistic).

ordinaryDeviance refers to [glm](glm)-like objects. It takes as input the restricted fit fm and optionally the value of the dispersion parameter and returns the deviance corresponding to the restricted fit divided by dispersion.

RaoScoreStatistic refers to [glm](glm)-like objects. It returns the value of the Rao score statistic $s(\beta)^T i^{-1}(\beta) s(\beta)/\phi$, where s is the vector of estimating equations, ϕ is the dispersion parameter and

$$i(\beta) = cov(s(\beta)) = X^T W(\beta) X/\phi,$$

in standard GLM notation. The additional argument X is the model matrix of the full (**not** the restricted) fit. In this way the original fit has always smaller or equal Rao score statistic from any restricted fit. The Rao score statistic could be used for the construction of confidence intervals when quasi-likelihood estimation is used (see Lindsay and Qu, 2003).

Value

A scalar.

Note

Because the objective functions are evaluated many times in `profiling`, `prelim.profiling` and `profileModel`, they should be as computationally efficient as possible.

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

References

Lindsay, B. G. and Qu, A. (2003). Inference functions and quadratic score tests. *Statistical Science* **18**, 394–410.

See Also

`profiling`, `prelim.profiling`, `profileModel`.

---

plot.profileModel   *Plot methods for 'profileModel' objects*

Description

`plot.profileModel` plots the profiles contained in the profiled object. `pairs.profileModel` is a diagnostic tool that plots pairwise profile traces.

Usage

```
## S3 method for class 'profileModel'
plot(x, cis = NULL, signed = FALSE, interpolate = TRUE,
                n.interpolations = 100, print.grid.points = FALSE,
                title = NULL, ...)

## S3 method for class 'profileModel'
pairs(x, colours = 2:3, title=NULL, ...)
```

Arguments

x	a "profileModel" object.
cis	the confidence intervals resulted from profConfint(prof). The default is NULL where no intervals are plotted. Only used in plot.profileModel.
signed	if TRUE the signed square roots of the values of the profiled objective are plotted. The default is FALSE. Available only in plot.profileModel.
interpolate	if TRUE spline interpolation is used in order to get a smooth plot of the profiled objective. If FALSE the points that are contained in the "profileModel" object are simply joint by segments. The default is TRUE. Available only in plot.profileModel.

n.interpolations

> The number of interpolations to take place in the profile range of each parameter. The default value is 100. It is only used when interpolate=TRUE. Available only in plot.profileModel.

print.grid.points

> logical indicating whether the points contained in the "profileModel" object should be printed along with the objective. The default is FALSE. Available only in plot.profileModel.

colours
> A vector of two elements indicating the colours to be used for plotting pairwise profile traces. Available only in pairs.profileModel.

title
> A character string to be displayed at the top of the resultant plotting device. The default is NULL where nothing is printed.

...
> further arguments passed to or from other methods.

Details

pairs.profileModel is a minor modification of pairs.profile in **MASS** library. The modification was done under the GPL licence 2 or greater and after the permission of the authors, in order to comply with objects of class "profileModel". As in the description of pairs.profile in Venables and Ripley (2002b), pairs.profileModel shows the lines that would join up the points where the contours have horizontal and vertical tangents, respectively, and the fine 'hairs' cutting the lines in the pairs plot are an indication of those tangents.

The pair plots should only be used for diagnostic purposes.

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

References

Venables, W. N. and Ripley, B. D. (2002a). *Modern applied statistics with S* (4th Edition). Springer.

Venables, W. N. and Ripley, B. D. (2002b). Statistics complements to modern applied statistics with S (4th Edition).
<http://www.stats.ox.ac.uk/pub/MASS4/VR4stat.pdf>.

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Chapman & Hall/CRC.

See Also

[profileModel](), [confintModel](), [profile.glm]()

Examples

```
# see example in 'confintModel'.
```

---

print.profileModel   *Printing 'profileModel' objects*

Description

Print method for objects of class `profileModel`.

Usage

```
## S3 method for class 'profileModel'
print(x, print.fit = FALSE,  ...)
```

Arguments

x	a "profileModel" object.
print.fit	logical indicating whether the fitted object supplied in [profileModel](#) should be printed. The default value is FALSE.
...	additional arguments to [print](#).

Details

This is the `print` method for objects inheriting from class "profileModel".

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

See Also

[print](#), [profileModel](#).

Examples

```
## Begin Example
y <- c(1,1,0,0)
x1 <- c(1,0,1,0)
x2 <- c(1,1,0,0)
prof1 <- profileModel(glm(y ~ x1 + x2, family = binomial),
                      objective = "ordinaryDeviance",
                      grid.bounds = rep(c(-1,1),3))
print(prof1)
prof2 <- update(prof1, quantile = qchisq(0.95,1), grid.bounds=NULL)
print(prof2, print.fit = TRUE)
## End Example
```

| profileModel | *Get the profiles of arbitrary objectives for arbitrary 'glm'-like models* |

Description

Calculates the profiles of **arbitrary** objectives (inference functions in the terminology of Lindsay and Qu, 2003) for the parameters of **arbitrary** glm-like models with linear predictor. It provides a variety of options such as profiling over a pre-specified grid, profiling until the profile of the objective reaches the values of a quantile, calculating the profile traces along with the profiled objectives, and others.

Usage

```
profileModel(fitted, gridsize = 20, stdn = 5, stepsize = 0.5,
             grid.bounds = NULL, quantile = NULL,
             objective = stop("'objective' is missing."),
             agreement = TRUE, verbose = TRUE, trace.prelim = FALSE,
             which = 1:length(coef(fitted)), profTraces = TRUE,
             zero.bound = 1e-08, scale = FALSE,
             stdErrors = NULL, ...)

prelim.profiling(fitted, quantile = qchisq(0.95, 1),
                 objective = stop("'objective' is missing."),
                 verbose = TRUE, which = 1:length(coef(fitted)),
                 stepsize = 0.5, stdn = 5, agreement = TRUE,
                 trace.prelim = FALSE,
                 stdErrors = NULL, ...)

profiling(fitted, grid.bounds, gridsize = 20, verbose = TRUE,
          objective = stop("'objective' is missing."),
          agreement = TRUE, which = 1:length(coef(fitted)),
          profTraces = TRUE, zero.bound = 1e-08, ...)
```

Arguments

fitted	a glm-like fitted object with **linear predictor** (see Details for the methods that have to be supported by fitted).
which	which parameters should be profiled? Has to be a vector of integers for profiling and prelim.profiling but for profileModel it could also be a vector of parameter names. The default is 1:length(coef(fitted)), i.e. all the parameters estimated in fitted.
grid.bounds	a matrix of dimension length(which) by 2 or a 2*length(which) vector that specifies the range of values in which profiling takes place for each parameter. It has to be set for profiling and the default is NULL for profileModel
gridsize	The number of equidistant parameter values to be taken between the values specified in the entries of grid.bounds.

stepsize	a positive integer that is used in prelim.profiling to penalize the size of the steps taken to the left and to the right of the estimate. The default value is 0.5.
stdn	in profileModel, the number of estimated standard deviations to move left or right from the estimated parameter value, when both quantile and grid.bounds are NULL. In prelim.profiling, stdn/stepsize is the maximum number of steps that are taken to the left and to the right of the estimate. The default value of stdn is 5 (see Details).
quantile	a quantile, indicating the range that the profile must cover. The default value in profileModel is NULL and in prelim.profiling, qchisq(0.95,1) (see Details).
objective	the function to be profiled. It is a function of the **restricted** fitted object and other arguments (see [objectives](#)). It should be of class function for profiling and prelim.profiling but it could also be a character string to be matched for profileModel.
agreement	logical indicating whether the fitting method used for fitting agrees with the specified objective, i.e. whether the objective is minimized at coef(fitted). The default is TRUE.
verbose	logical. If TRUE (default) progress indicators are printed during the profiling progress.
trace.prelim	logical. If TRUE the preliminary iteration is traced. The default is FALSE.
profTraces	logical indicating whether the profile traces should be returned. The default is TRUE.
zero.bound	a small positive constant. The difference of the objective at the **restricted** fit from the objective at fitted takes value zero if it is smaller than zero.bound. zero.bound is only used when agreement=TRUE and the default value is 1e-08.
scale	logical. The default is FALSE. Currently has no effect. Only available in profileModel.
stdErrors	The vector estimated asymptotic standard errors reported from the fitting procedure. The default is NULL (see Details).
...	further arguments passed to the specified objective.

Details

fitted has to be an object which supports the method [coef](#) and which has fitted$terms with the same meaning as, for example, in [lm](#) and [glm](#) (see also [terms](#)). coef(fitted) has to be a **vector** of coefficients with each component corresponding to a column of the model matrix returned by

mf <- model.frame(fitted$terms,data=eval(fitted$call$data)) ; model.matrix(fitted$terms,mf,contrasts = fitted$contrasts)

(or just model.matrix(fitted), for fitted objects that support the [model.matrix](#) method.)

Exception to this are objects returned by BTm of the **BradleyTerry** package, where some special handling of the required objects takes place.

Note that any or both of data and contrasts could be NULL. This depends whether the data argument has been supplied to the procedure and whether fitted$contrast exists.

The fitting procedure that resulted fitted has to support [offset](#) in [formula](#). Also, fitted$call has to be the call that generated fitted.

If the fitting procedure that resulted `fitted` supports an `etastart` argument (see `glm`) and `fitted$linear.predictor` contains the estimated linear predictors then during profiling, the appropriate starting values are supplied to the fitting procedure. In this way, the iteration is accelerated and is more stable, numerically. However, it is not necessary that `etastart` is supported. In the latter case no starting values are supplied to the fitting procedure during profiling.

Support for a `summary` method is optional. `summary` is only used for obtaining the estimated asymptotic standard errors associated to the coefficients in `fitted`. If `stdErrors=NULL` the standard errors are taken to be summary(fitted)$coefficients[,2] which is the place where the estimated asymptotic standard errors usually are for `glm`-like objects. If this this is not the case then `stdErrors` should be set appropriately.

`profiling` is the workhorse function that does the basic operation of profiling objectives over a user-specified grid of values. For a given parameter β, the **restricted** fit $F_{\beta=b}$ is calculated by constraining β to a point b of the grid. Then the difference

$$D(F_{\beta=b}) = P(F_{\beta=b}) - P(F_0),$$

is calculated, where P is the objective specified by the user and G is the original fit (`fitted`). For convex objectives that are minimized at the estimates of G (see `agreement`), $D(G) = 0$.

`prelim.profiling` refers only to convex objectives and searches for and returns the grid bounds (`grid.bounds`) for each profiled parameter that should be used in order the profile to cover `quantile`. For a given parameter β, `prelim.profiling` also checks whether such enclosure can be found and returns a logical matrix `intersects` of dimension length(which) by 2 that indicates if the profile covers the quantile to the left and to the right of the estimate in `fitted`. At step i of the search a value b_i is proposed for β and $D(F_{\beta=b_i})$ is calculated. If $D(F_{\beta=b_i}) < q$, where q is `quantile`, the next proposed value is

$$b_{i+1} = b_i \pm (i+1)C \min(s, 30)/|L|,$$

where C is `stepsize`, s is the estimated asymptotic standard error of β from G and L is the slope of the line segment connecting the points $(b_i, D(F_{\beta=b_i}))$ and $(b_{i-1}, D(F_{\beta=b_{i-1}}))$. $\pm$ is $+$ if the search is on the right of the estimate of β and $-$ on the left. If an increase of D is expected then the step slows down. If $|L| < 1$ then $|L|$ is set to 1 and if $|L| > 500$ then $|L|$ is set to 500. In this way the iteration is conservative by avoiding very small steps but not over-conservative by avoiding very large steps.

If the maximum number of steps `stdn/stepsize` (call this M) was taken and the quantile was not covered by the profile but the three last absolute slopes where positive then the iteration is restarted form b_{M-1} with $2C$ instead of C in the step calculation. If the three last slopes were less than `1e-8` in absolute value then the iteration stops and it is considered that D has an asymptote at the corresponding direction (left or right). Note that when the latter takes place the iteration has already moved $6C \min(s, 30)$ units on the scale of β, since the first value of b were a slope of 1e-8 in absolute value was detected. Thus we could safely say that an asymptote has been detected and avoid calculation of $F_{beta=b}$ for extremely large b's.

Very small values of `stepsize` make `prelim.profiling` take very small steps with the effect of slowing down the execution time. Large values of `stepsize` are only recommended when the estimated asymptotic standard errors are very small in `fitted`.

profileModel is a wrapper function that collects and combines the capabilities of `profiling` and `prelim.profiling` by providing a unified interface for their functions, as well as appropriateness checks on the arguments. When both `quantile` and `grid.bounds` are NULL then `profiling` is called and profiling takes place for `stdn` estimated asymptotic standard errors on the left and on the right of the estimates in `fitted`. This could be used for taking a quick look of the profiles around the estimate. With only the `quantile` being NULL, profiling is performed on the the specified grid of values. When `quantile` is specified and `grid.bounds` is NULL, `prelim.profiling` is called and its result is passed to `profiling`. If both `quantile` and `grid.bounds` then `grid.bounds` prevails and profiling is performed on the specified grid.

Value

`profiling` returns a list of profiles, with one named component for each parameter profiled. Each component of the list contains the profiled parameter values and the corresponding differences of the objective at the **restricted** fit from the objective at `fitted`. When `profTraces=TRUE` the corresponding profile traces are [cbind](#)'ed to each component of the list.

`prelim.profiling` returns a list with components `intersects` and `grid.bounds`.

`profileModel` returns an object of class `"profileModel"` that has the attribute `includes.traces` corresponding to the value of the `profTraces` argument. The `"profileModel"` object is a list of the following components:

profiles	the result of `profiling`.
fit	the `fitted` object that was passed to `profileModel`.
quantile	the quantile that was passed to `profileModel`.
gridsize	the gridsize that was passed to `profileModel`.
intersects	if `quantile=NULL` then `intersects=NULL` else `intersects` is as for `prelim.profiling`.
profiled.parameters	
	a vector of integers indicating which parameters were profiled.
profiled.objective	
	the profiled objective with any additional arguments passed through `...` evaluated.
isNA	a logical vector indicating which of the parameters in `which` were NA in `fitted`.
agreement	the agreement that was passed to `profileModel`.
zero.bound	the `zero.bound` that was passed to `profileModel`.
grid.bounds	the grid bounds that were used for profiling.
call	the matched call.

Note

Methods specific to objects of class `"profileModel"` are

- print, see [print.profileModel](#).
- signedSquareRoots, see [signedSquareRoots](#).
- profConfint, see [profConfint](#).
- plot, see [plot.profileModel](#).

• pairs, see `pairs.profileModel`.

`profileModel` has been tested and is known to work for fitted objects resulting from `lm`, `glm`, polr, gee, geeglm, brglm and BTm.

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

References

Lindsay, B. G. and Qu, A. (2003). Inference functions and quadratic score tests. *Statistical Science* **18**, 394–410.

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Chapman & Hall/CRC.

See Also

`confintModel`, `plot.profileModel`.

Examples

```
## Begin Example 1
library(MASS)
m1 <- glm(Claims ~ District + Group + Age + offset(log(Holders)),
          data = Insurance, family = poisson)
# profile deviance +-5 estimated standard errors from the estimate
prof0 <- profileModel(m1, objective = "ordinaryDeviance")
# profile deviance over a grid of values
gridd <- rep(c(-1,1), length(coef(m1)))
prof1 <- profileModel(m1, grid.bounds = gridd,
                      objective = "ordinaryDeviance")
# profile deviance until the profile reaches qchisq(0.95,1)
prof2 <- profileModel(m1, quantile = qchisq(0.95,1) ,
                      objective = "ordinaryDeviance")
# plot the profiles of the deviance
plot(prof2)
# quite quadratic in shape. Just to make sure:
plot(prof2, signed = TRUE)
# Ok straight lines. So we expect first order asymptotics to work well;
## Not run:
# plot the profiles of the Rao score statistic
# profile Rao's score statistic
prof3 <- update(prof2, objective = "RaoScoreStatistic",
                X = model.matrix(m1))
plot(prof3)
# The 95% confidence intervals based on prof2 and prof3 and the simple Wald
# confidence intervals:
profConfint(prof2)
profConfint(prof3)
stdErrors <- coef(summary(m1))[,2]
coef(m1)+ qnorm(0.975) * cbind(-stdErrors,stdErrors)
# They are all quite similar in value. The result of a quadratic likelihood.
```

```
## End Example

## End(Not run)
## Begin Example 2: Monotone likelihood; data separation;
library(MASS)
y <- c(0, 0, 1, 0)
tots <- c(2, 2, 5, 2)
x1 <- c(1, 0, 1, 0)
x2 <- c(1, 1, 0, 0)
m2 <- glm(y/tots ~ x1 + x2, weights = tots,
          family = binomial)
prof <- profileModel(m2, quantile=qchisq(0.95,1),
                     objective = "ordinaryDeviance")
plot(prof)
profConfint(prof)
# profile.glm fails to detect the finite endpoints
confint(m2)
## End Example
## Not run:
## Begin Example 3: polr
library(MASS)
options(contrasts = c("contr.treatment", "contr.poly"))
house.plr <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
prof.plr0 <- profileModel(house.plr, objective = function(fm) fm$deviance)
plot(prof.plr0)
# do it with a quantile
prof.plr1 <- update(prof.plr0, quantile = qchisq(0.95, 1))
plot(prof.plr1)
## End Example

## End(Not run)
```

---

signedSquareRoots   *Get the signed square roots of the profiles in 'profileModel'*

Description

Convert a "profileModel" object to contain the signed square roots of the profiles.

Usage

```
## S3 method for class 'profileModel'
signedSquareRoots(prof)
```

Arguments

prof   a "profileModel" object.

Details

signedSquareRoots takes as input a "profileModel" object and results to another "profileModel" object that contains the signed square roots of the profiled differences. The method only applies if agreement is set to TRUE in prof$call.

Value

an object of class "profileModel".

Author(s)

Ioannis Kosmidis <email: ioannis.kosmidis@warwick.ac.uk>

See Also

[plot.profileModel](#).

Index