Package 'jointCalib'

May 8, 2026

Type Package

Title A Joint Calibration of Totals and Quantiles

Version 0.1.0

Description A small package containing functions to perform a joint calibration of totals and quantiles. The calibration for totals is based on Deville and Särndal (1992) <doi:10.1080/01621459.1992.10475217>, the calibration for quantiles is based on Harms and Duchesne (2006) <https://www150.statcan.gc.ca/n1/en/catalogue/12-001-X20060019255>. The package uses standard calibration via the 'survey', 'sampling' or 'laeken' packages. In addition, entropy balancing via the 'ebal' package and empirical likelihood based on codes from Wu (2005) <https://www150.statcan.gc.ca/n1/pub/12-001-x/2005002/article/9051-eng.pdf> can be used. See the paper by Beręsewicz and Szymkowiak (2023) for details <doi:10.48550/arXiv.2308.13281>.

License GPL-3

Encoding UTF-8

RdMacros mathjaxr

Depends R (>= 3.5.0)

URL https://github.com/ncn-foreigners/jointCalib, https://ncn-foreigners.github.io/jointCalib/

BugReports https://github.com/ncn-foreigners/jointCalib/issues

RoxygenNote 7.2.3

Imports laeken, sampling, mathjaxr, survey, MASS, ebal

NeedsCompilation no

Author Maciej Beręsewicz [aut, cre] (ORCID: <https://orcid.org/0000-0002-8281-4301>)

Maintainer Maciej Beręsewicz <maciej.beresewicz@ue.poznan.pl>

Repository CRAN

Date/Publication 2026-04-02 16:47:39

Contents

`calib_el` *An internal function for calibration of weights using empirical likelihood method*

Description

`calib_el` performs calibration using empirical likelihood (EL) method. The function is taken from Wu (2005), if algorithm has problem with convergence codes from Zhang, Han and Wu (2022) using `constrOptim` is used.

In (pseudo) EL the following (pseudo) EL function is maximized

$$\sum_{i \in r} d_i \log(p_i),$$

under the following constraint

$$\sum_{i \in r} p_i = 1,$$

with constraints on quantiles (with notation as in Harms and Duchesne (2006))

$$\sum_{i \in r} p_i (a_i - \alpha/N) = 0,$$

where a_i is created using `joint_calib_create_matrix` function, and possibly means

$$\sum_{i \in r} p_i (x_i - \mu_x) = 0,$$

where μ_x is known population mean of X. For simplicity of notation we assume only one quantile and one mean is known. This can be generalized to multiple quantiles and means.

Usage

```
calib_el(X, d, totals, maxit = 50, tol = 1e-08, eps = .Machine$double.eps, ...)
```

Arguments

X	matrix of variables for calibration of quantiles and totals (first column should be intercept),
d	initial d-weights for calibration (e.g. design-weights),
totals	vector of totals (where 1 element is the population size),
maxit	a numeric value giving the maximum number of iterations,
tol	the desired accuracy for the iterative procedure,
eps	the desired accuracy for computing the Moore-Penrose generalized inverse (see MASS::ginv()),
...	arguments passed to stats::optim via stats::constrOptim.

Value

Returns a vector of empirical likelihood g-weights

Author(s)

Maciej Beręsewicz based on Wu (2005) and Zhang, Han and Wu (2022)

References

Wu, C. (2005). Algorithms and R codes for the pseudo empirical likelihood method in survey sampling. Survey Methodology, 31(2), 239 (code is taken from https://sas.uwaterloo.ca/~cbwu/Rcodes/LagrangeM2.txt).

Zhang, S., Han, P., and Wu, C. (2023) Calibration Techniques Encompassing Survey Sampling, Missing Data Analysis and Causal Inference. International Statistical Review, 91: 165–192. https://doi.org/10.1111/insr.1251 (code is taken from Supplementary Materials).

Examples

```
## generate data based on Haziza and Lesage (2016)
set.seed(123)
N <- 1000
x <- runif(N, 0, 80)
y <- exp(-0.1 + 0.1*x) + rnorm(N, 0, 300)
p <- rbinom(N, 1, prob = exp(-0.2 - 0.014*x))
totals_known <- c(N=N, x=sum(x))
df <- data.frame(x, y, p)
df_resp <- df[df$p == 1, ]
df_resp$d <- N/nrow(df_resp)
res <- calib_el(X = model.matrix(~x, df_resp),
                d = df_resp$d,
                totals = totals_known)
data.frame(known = totals_known, estimated=colSums(res*df_resp$d*model.matrix(~x, df_resp)))
```

`control_calib` *control parameters*

Description

`control_calib` is function that contains control parameters for `joint_calib_create_matrix`

Usage

```
control_calib(
  interpolation = c("logit", "linear"),
  logit_const = -1000,
  survey_sparse = FALSE,
  ebal_constraint_tolerance = 1,
  ebal_print_level = 0
)
```

Arguments

`interpolation`	type of interpolation: `logit` or `linear`,
`logit_const`	constant for `logit` interpolation,
`survey_sparse`	whether to use sparse matrices via `Matrix` package in `survey::grake()` (currently not supported),
`ebal_constraint_tolerance`	This is the tolerance level used by ebalance to decide if the moments in the reweighted data are equal to the target moments (see `ebal::ebalance()`),
`ebal_print_level`	Controls the level of printing: 0 (normal printing), 2 (detailed), and 3 (very detailed) (see `ebal::ebalance()`).

Value

a list with parameters

Author(s)

Maciej Beręsewicz

joint_calib *Function for the joint calibration of totals and quantiles*

Description

joint_calib allows joint calibration of totals and quantiles. It provides a user-friendly interface that includes the specification of variables in formula notation, a vector of population totals, a list of quantiles, and a variety of backends and methods.

Usage

```
joint_calib(
  formula_totals = NULL,
  formula_quantiles = NULL,
  data = NULL,
  dweights = NULL,
  N = NULL,
  pop_totals = NULL,
  pop_quantiles = NULL,
  subset = NULL,
  backend = c("sampling", "laeken", "survey", "ebal", "base"),
  method = c("raking", "linear", "logit", "sinh", "truncated", "el", "eb"),
  bounds = c(0, 10),
  maxit = 50,
  tol = 1e-08,
  eps = .Machine$double.eps,
  control = control_calib(),
  ...
)
```

Arguments

formula_totals	a formula with variables to calibrate the totals,
formula_quantiles	a formula with variables for quantile calibration,
data	a data.frame with variables,
dweights	initial d-weights for calibration (e.g. design weights),
N	population size for calibration of quantiles,
pop_totals	a named vector of population totals for formula_totals. Should be provided exactly as in survey package (see survey::calibrate),
pop_quantiles	a named list of population quantiles for formula_quantiles or an newsvyquantile class object (from survey::svyquantile function),
subset	a formula for subset of data,
backend	specify an R package to perform the calibration. Only sampling, laeken, survey, ebal or base are allowed,

method	specify method (i.e. distance function) for the calibration. Only raking, linear, logit, sinh, truncated, el (empirical likelihood), eb (entropy balancing) are allowed,
bounds	a numeric vector of length two giving bounds for the g-weights,
maxit	a numeric value representing the maximum number of iterations,
tol	the desired accuracy for the iterative procedure (for sampling, laeken, ebal, el) or tolerance in matching population total for survey::grake (see help for survey::grake)
eps	the desired accuracy for computing the Moore-Penrose generalized inverse (see MASS::ginv())
control	a list of control parameters (currently only for joint_calib_create_matrix)
...	arguments passed either to sampling::calib, laeken::calibWeights, survey::calibrate or optim::constrOptim

Value

Returns a list with containing:

- g – g-weight that sums up to sample size,
- Xs – matrix used for calibration (i.e. Intercept, X and X_q transformed for calibration of quantiles),
- totals – a vector of totals (i.e. N, pop_totals and pop_quantiles),
- method – selected method,
- backend – selected backend.

Author(s)

Maciej Beręsewicz

References

Beręsewicz, M., and Szymkowiak, M. (2023). A note on joint calibration estimators for totals and quantiles Arxiv preprint https://arxiv.org/abs/2308.13281

Deville, J. C., and Särndal, C. E. (1992). Calibration estimators in survey sampling. Journal of the American statistical Association, 87(418), 376-382.

Harms, T. and Duchesne, P. (2006). On calibration estimation for quantiles. Survey Methodology, 32(1), 37.

Wu, C. (2005) Algorithms and R codes for the pseudo empirical likelihood method in survey sampling, Survey Methodology, 31(2), 239.

Zhang, S., Han, P., and Wu, C. (2023) Calibration Techniques Encompassing Survey Sampling, Missing Data Analysis and Causal Inference, International Statistical Review 91, 165–192.

Haziza, D., and Lesage, É. (2016). A discussion of weighting procedures for unit nonresponse. Journal of Official Statistics, 32(1), 129-145.

See Also

sampling::calib() – for standard calibration.

laeken::calibWeights() – for standard calibration.

survey::calibrate() – for standard and more advanced calibration.

ebal::ebalance() – for standard entropy balancing.

Examples

```
## generate data based on Haziza and Lesage (2016)
set.seed(123)
N <- 1000
x <- runif(N, 0, 80)
y <- exp(-0.1 + 0.1*x) + rnorm(N, 0, 300)
p <- rbinom(N, 1, prob = exp(-0.2 - 0.014*x))
probs <- seq(0.1, 0.9, 0.1)
quants_known <- list(x=quantile(x, probs))
totals_known <- c(x=sum(x))
df <- data.frame(x, y, p)
df_resp <- df[df$p == 1, ]
df_resp$d <- N/nrow(df_resp)
y_quant_true <- quantile(y, probs)
## standard calibration for comparison
result0 <- sampling::calib(Xs = cbind(1, df_resp$x),
                           d = df_resp$d,
                           total = c(N, totals_known),
                           method = "linear")

y_quant_hat0 <- laeken::weightedQuantile(x = df_resp$y,
                                         probs = probs,
                                         weights = result0*df_resp$d)
x_quant_hat0 <- laeken::weightedQuantile(x = df_resp$x,
                                         probs = probs,
                                         weights = result0*df_resp$d)

## example 1: calibrate only quantiles (deciles)
result1 <- joint_calib(formula_quantiles = ~x,
                       data = df_resp,
                       dweights = df_resp$d,
                       N = N,
                       pop_quantiles = quants_known,
                       method = "linear",
                       backend = "sampling")
## estimate quantiles
y_quant_hat1 <- laeken::weightedQuantile(x = df_resp$y,
                                         probs = probs,
                                         weights = result1$g*df_resp$d)
x_quant_hat1 <- laeken::weightedQuantile(x = df_resp$x,
                                         probs = probs,
                                         weights = result1$g*df_resp$d)

## compare with known
```

```
data.frame(standard = y_quant_hat0, est=y_quant_hat1, true=y_quant_true)

## example 2: calibrate with quantiles (deciles) and totals
result2 <- joint_calib(formula_totals = ~x,
                       formula_quantiles = ~x,
                       data = df_resp,
                       dweights = df_resp$d,
                       N = N,
                       pop_quantiles = quants_known,
                       pop_totals = totals_known,
                       method = "linear",
                       backend = "sampling")
## estimate quantiles
y_quant_hat2 <- laeken::weightedQuantile(x = df_resp$y,
                                         probs = probs,
                                         weights = result2$g*df_resp$d)
x_quant_hat2 <- laeken::weightedQuantile(x = df_resp$x,
                                         probs = probs,
                                         weights = result2$g*df_resp$d)

## compare with known
data.frame(standard = y_quant_hat0, est1=y_quant_hat1,
           est2=y_quant_hat2, true=y_quant_true)

## example 3: calibrate wigh quantiles (deciles) and totals with
## hyperbolic sinus (sinh) and survey package

result3 <- joint_calib(formula_totals = ~x,
                       formula_quantiles = ~x,
                       data = df_resp,
                       dweights = df_resp$d,
                       N = N,
                       pop_quantiles = quants_known,
                       pop_totals = totals_known,
                       method = "sinh",
                       backend = "survey")

## estimate quantiles
y_quant_hat3 <- laeken::weightedQuantile(x = df_resp$y,
                                         probs = probs,
                                         weights = result3$g*df_resp$d)
x_quant_hat3 <- laeken::weightedQuantile(x = df_resp$x,
                                         probs = probs,
                                         weights = result3$g*df_resp$d)

## example 4: calibrate wigh quantiles (deciles) and totals with ebal package
result4 <- joint_calib(formula_totals = ~x,
                       formula_quantiles = ~x,
                       data = df_resp,
                       dweights = df_resp$d,
                       N = N,
                       pop_quantiles = quants_known,
                       pop_totals = totals_known,
```

```
                        method = "eb",
                        backend = "ebal")

## estimate quantiles
y_quant_hat4 <- laeken::weightedQuantile(x = df_resp$y,
                                         probs = probs,
                                         weights = result4$g*df_resp$d)
x_quant_hat4 <- laeken::weightedQuantile(x = df_resp$x,
                                         probs = probs,
                                         weights = result4$g*df_resp$d)

## compare with known
data.frame(standard = y_quant_hat0,
           est1=y_quant_hat1,
           est2=y_quant_hat2,
           est3=y_quant_hat3,
           est4=y_quant_hat4,
           true=y_quant_true)
## compare with known X
data.frame(standard = x_quant_hat0,
           est1=x_quant_hat1,
           est2=x_quant_hat2,
           est3=x_quant_hat3,
           est4=x_quant_hat4,
           true = quants_known$x)
```

`joint_calib_create_matrix`

An internal function to create an A matrix for calibration of quantiles

---

Description

`joint_calib_create_matrix` is function that creates an $A = [a_{ij}]$ matrix for calibration of quantiles. Function allows to create matrix using `logistic` interpolation (using `stats::plogis`, default) or `linear` (as in Harms and Duchesne (2006), i.e. slightly modified Heavyside function).

In case of `logistic` interpolation elements of A are created as follows

$$a_{ij} = \frac{1}{(1 + \exp\left(-2l\left(x_{ij} - Q_{x_j,\alpha}\right)\right))N},$$

where x_{ij} is the ith row of the auxiliary variable X_j, N is the population size, $Q_{x_j,\alpha}$ is the known population αth quantile, and l is set to -1000 (by default).

In case of `linear` interpolation elements of A are created as follows

$$a_{ij} = \begin{cases} N^{-1}, & x_{ij} \leqslant L_{x_j,r}\left(Q_{x_j,\alpha}\right), \\ N^{-1}\beta_{x_j,r}\left(Q_{x_j,\alpha}\right), & x_{ij} = U_{x_j,r}\left(Q_{x_j,\alpha}\right), \\ 0, & x_{ij} > U_{x_j,r}\left(Q_{x_j,\alpha}\right), \end{cases}$$

$i = 1, ..., r$, $j = 1, ..., k$, where r is the set of respondents, k is the auxiliary variable index and

$$L_{x_j,r}(t) = \max\left\{\{x_{ij}, i \in s \mid x_{ij} \leqslant t\} \cup \{-\infty\}\right\},$$
$$U_{x_j,r}(t) = \min\left\{\{x_{ij}, i \in s \mid x_{ij} > t\} \cup \{\infty\}\right\},$$
$$\beta_{x_j,r}(t) = \frac{t - L_{x_j,s}(t)}{U_{x_j,s}(t) - L_{x_j,s}(t)},$$

$i = 1, ..., r$, $j = 1, ..., k$, $t \in \mathbb{R}$.

Usage

```
joint_calib_create_matrix(X_q, N, pop_quantiles, control = control_calib())
```

Arguments

X_q	matrix of variables for calibration of quantiles,
N	population size for calibration of quantiles,
pop_quantiles	a vector of population quantiles for X_q,
control	a control parameter for creation of X_q matrix.

Value

Return matrix A

Author(s)

Maciej Beręsewicz

References

Harms, T. and Duchesne, P. (2006). On calibration estimation for quantiles. Survey Methodology, 32(1), 37.

Examples

```
# Create matrix for one variable and 3 quantiles
set.seed(123)
N <- 1000
x <- as.matrix(rnorm(N))
quants <- list(quantile(x, c(0.25,0.5,0.75)))
A <- joint_calib_create_matrix(x, N, quants)
head(A)
colSums(A)

# Create matrix with linear interpolation
A <- joint_calib_create_matrix(x, N, quants, control_calib(interpolation="linear"))
head(A)
colSums(A)
```

```
# Create matrix for two variables and different number of quantiles

set.seed(123)
x1 <- rnorm(N)
x2 <- rchisq(N, 1)
x <- cbind(x1, x2)
quants <- list(quantile(x1, 0.5), quantile(x2, c(0.1, 0.75, 0.9)))
B <- joint_calib_create_matrix(x, N, quants)
head(B)
colSums(B)
```

Index