Package 'EVER'

June 2, 2020

Type Package

Title Estimation of Variance by Efficient Replication

Description Delete-A-Group Jackknife replication. Calibration of replicate weights. Estimates, standard errors and confidence intervals for: totals, means, absolute and relative frequency distributions, contingency tables, ratios, quantiles and regression coefficients. Estimates, standard errors and confidence intervals for user-defined estimators (even non-analytic). Domain (subpopulation) estimation.

Version 1.3

Author Diego Zardetto [aut, cre]

Maintainer Diego Zardetto <zardetto@istat.it>

License EUPL

Imports stats, MASS

Depends R (>= 3.5.0)

ByteCompile TRUE

R topics documented:

bounds.hint *A hint for range restricted calibration*

Description

Suggests a sound bounds value for which kottcalibrate is likely to converge.

Usage

```
bounds.hint(deskott, df.population,
    calmodel = if (inherits(df.population, "pop.totals"))
                   attr(df.population, "calmodel"),
    partition = if (inherits(df.population, "pop.totals"))
                    attr(df.population, "partition") else FALSE)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
df.population	Data frame containing the known population totals for the auxiliary variables.
calmodel	Formula defining the linear structure of the calibration model.
partition	Formula specifying the variables that define the "calibration domains" for the model; FALSE (the default) implies no calibration domains.

Details

The function bounds.hint returns a bounds value for which kottcalibtrate is *likely* to converge. This interval is just a sound hint, *not* an exact result (see 'Note').

The mandatory argument deskott identifies the kott.design object on which the calibration problem is defined.

The mandatory argument df.population identifies the known totals data frame.

The argument calmodel symbolically defines the calibration model you want to use: it identifies the auxiliary variables and the constraints for the calibration problem. The deskott variables referenced by calmodel must be numeric or factor and must not contain any missing value (NA). The argument can be omitted provided df.population is an object of class pop.totals (see population.check).

The optional argument partition specifies the variables that define the calibration domains for the model. The default value (FALSE) means either that there are not calibration domains or that you want to solve the problem globally (even though it could be factorised). The deskott variables referenced by partition (if any) must be factor and must not contain any missing value (NA). The argument can be omitted provided df.population is an object of class pop.totals (see population.check).

Value

A numeric vector of length 2, representing the *suggested* value for the bounds argument of kottcalibrate. The attributes of that vector store additional information, which can lead to better understand why a given calibration problem is (un)feasible (see 'Examples').

Note

Assessing the feasibility of an arbitrary calibration problem is not an easy task. The problem is even more difficult whenever additional *"range restrictions"* are imposed. Indeed, even if one assumes that the calibration constraints define a consistent system, one also has to choose the bounds such that the feasible region is non-empty.

One can argue that there must exist a minimun-length interval $I = [L, U]$ such that, if it is covered by bounds, the specified calibration problem is feasible. Unfortunately in order to compute exactly that minimun-length interval I one should solve a big linear programming problem [Vanderhoeft 01]. As an alternative, a trial and error procedure has been frequently proposed [Deville et al 1993; Sautory 1993]: (i) start with a very large interval `bounds.0`; (ii) if convergence is achieved, shrink it so as to obtain a new inteval `bounds.1`; (iii) repeat until you get a sufficiently tight feasible interval `bounds.n`. The drawback is that this procedure can cost a lot of computer time since, for each choice of the bounds, the full calibration problem has to be solved.

A rather easy task is, on the contrary, the one of finding at least a given specific interval $I^* = [L^*, U^*]$ such that, if it is *not* covered by bounds, the current calibration problem is *surely unfeasible*. This means that any feasible bounds value must necessarily contain the I^* interval. The function `bounds.hint`: (i) first identifies such an I^* interval (by computing the range of the ratios between known population totals and corresponding direct Horvitz-Thompson estimates), (ii) then builds a new interval I^{sugg} with same midpoint and double length. The latter is the *suggested* value for the bounds argument of `kottcalibrate`. The return value of `bounds.hint` should be understood as a useful starting guess for bounds, even though there is definitely no warranty that the calibration algorithm will actually converge.

Author(s)

Diego Zardetto

References

Vanderhoeft, C. (2001) *"Generalized Calibration at Statistic Belgium"*, Statistics Belgium Working Paper n. 3, http://www.statbel.fgov.be/studies/paper03_en.asp.

Deville, J.C., Sarndal, C.E. and Sautory, O. (1993) *"Generalized Raking Procedures in Survey Sampling"*, Journal of the American Statistical Association, Vol. 88, No. 423, pp.1013-1020.

Sautory, O. (1993) *"La macro CALMAR: Redressement d'un Echantillon par Calage sur Marges"*, Document de travail de la Direction des Statistiques Demographiques et Sociales, no. F9310.

See Also

`kottcalibrate` for calibrating replicate weights, `pop.template` for constructing known totals data frames in compliance with the standard required by `kottcalibrate`, `population.check` to check that the known totals data frame satisfies that standard and `g.range` to compute the range of the obtained g-weights.

Examples

```
# Load sample data (the only reason for fixing
# the RNG seed is to achieve reproducible examples)
data(data.examples)
set.seed(123)

# Creation of the object to be calibrated:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
```

```
      weights=~weight,nrg=15)

# Calibration (global solution) on the joint distribution
# of sex and marstat (totals in pop03). Get a hint for feasible bounds:
hint<-bounds.hint(kdes,pop03,~marstat:sex-1)

# Let's first verify if calibration converges with the suggested
# value for the bounds argument (i.e. c(0.909, 1.062) ):
kdescal03<-kottcalibrate(deskott=kdes,df.population=pop03,
           calmodel=~marstat:sex-1,calfun="logit",bounds=hint)

# Now let's verify that calibration fails, if bounds don't cover
# the interval [0.947, 1.023]:
## Not run:
kdescal03<-kottcalibrate(deskott=kdes,df.population=pop03,
           calmodel=~marstat:sex-1,calfun="logit",bounds=c(0.95, 1.03))

## End(Not run)


# Calibration (iterative solution) on the totals for the quantitative
# variables x1, x2 and x3 in the subpopulations defined by the
# regcod variable (totals in pop04p): Get a hint for feasible bounds:
hint<-bounds.hint(kdes,pop04p,~x1+x2+x3-1,~regcod)

# Let's verify if calibration converges with the suggested
# value for the bounds argument (i.e. c(0.133, 2.497) ):
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calmodel=~x1+x2+x3-1,partition=~regcod,calfun="logit",
            bounds=hint,aggregate.stage=2)

# Now let's verify that calibration fails, if bounds don't cover
# the interval [0.724, 1.906]:
## Not run:
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calmodel=~x1+x2+x3-1,partition=~regcod,calfun="logit",
            bounds=c(0.71,1.89),aggregate.stage=2)

## End(Not run)

# By analysing kottcal.status one understands that calibration
# failed due to the sub-task identified by replicate.12 and
# regcod 6:
kottcal.status

# this is easily explained by inspectioning the "bounds"
# attribute of the bounds.hint output object:
hint

# indeed the specified upper bound (1.89) was too low
# for replicate.12 and regcod 6
```

---

data.examples *Example data for the EVER package*

Description

Example data frames and functions. Allow to run R code contained in the 'Examples' section of the EVER package help pages.

Usage

```
data(data.examples)
```

Format

The main data frame, named `example`, contains (artificial) data from a two stage stratified cluster sampling design. The sample is made up of 3000 final units, for which the following 21 variables were observed:

`towcod` Code identifying "variance PSUs": towns (PSUs) in NSR strata, families (SSUs) in SR strata, `numeric`

`famcod` Code identifying families (SSUs), `numeric`

`key` Key identifying final units (individuals), `numeric`

`weight` Initial weights, `numeric`

`stratum` Stratification variable, `factor` with levels `801 802 803 901 902 903 904 905 906 907 908 1001 1002 1003 1004 1005 1006 1007 1008 1009 1101 1102 1103 1104 3001 3002 3003 3004 3005 3006 3007 3008 3009 3010 3011 3012 3101 3102 3103 3104 3105 3106 3107 3108 3201 3202 3203 3204 5401 5402 5403 5404 5405 5406 5407 5408 5409 5410 5411 5412 5413 5414 5415 5416 5501 5502 5503 5504 9301 9302 9303 9304 9305 9306 9307 9308 9309 9310 9311 9312`

`SUPERSTRATUM` Collapsed strata variable (eliminates lonely PSUs), `factor` with levels `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55`

`sr` Strata type, `integer` with values `0` (NSR strata) and `1` (SR strata)

`regcod` Code identifying regions, `factor` with levels `6 7 10`

`procod` Code identifying provinces, `factor` with levels `8 9 10 11 30 31 32 54 55 93`

`x1` Indicator variable (integer), `numeric`

`x2` Indicator variable (integer), `numeric`

`x3` Indicator variable (integer), `numeric`

`y1` Indicator variable (integer), `numeric`

`y2` Indicator variable (integer), `numeric`

`y3` Indicator variable (integer), `numeric`

`age5c` Age variable with 5 classes, `factor` with levels `1 2 3 4 5`

`age10c` Age variable with 10 classes, `factor` with levels `1 2 3 4 5 6 7 8 9 10`

`sex` Sex variable, `factor` with levels `f m`

`marstat` Marital status variable, `factor` with levels `married unmarried widowed`

`z` A continuous quantitative variable, `numeric`

`income` Income variable, `numeric`

Details

Objects pop01, ..., pop05p contain known population totals for various calibration models. Object pairs with names differing in the 'p' suffix (such as pop03 and pop03p) refer to the *same* calibration problem but pertain to *different* solution methods (global and iterative respectively, see kottcalibrate). The two-component numeric vector bounds expresses a possible choice for the allowed range for the ratios between calibrated weights and direct weights in the aforementioned calibration problems.

Functions ones, poverty and ratio are intended to show how to use kottby.user for calculating estimates, standard errors and confidence intervals for user-defined estimators.

Examples

```
data(data.examples)
```

---

desc *Description of replicated objects*

Description

Concisely describes a kott.design object.

Usage

```
desc(deskott, descfun = NULL, ...)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
descfun	Optional description function to be used; must accept a data.frame object as first argument.
...	Additional parameters to be passed to descfun.

Details

This function prints a concise description (i) of the sampling design for the original survey data and (ii) of the replication process these data have undergone.

The optional argument descfun allows to specify an R function (like head, str, summary, ...) to be used to analyse, describe, or summarise the data frame contained in deskott.

Value

The return value depends on the descfun parameter. If not specified (the default option), desc does not return any value.

Author(s)

Diego Zardetto

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Concise description:
desc(kdes)

# Display first rows of kdes data:
desc(kdes,head)

# Ask essential information on kdes internal structure:
desc(kdes,str)


# Creation of a kott.cal.design object:
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calmodel=~x1+x2+x3-1,partition=~regcod,calfun="logit",
            bounds=bounds,aggregate.stage=2)

# Concise description:
desc(kdescal04p)

# Display first rows of kdescal04p data:
desc(kdescal04p,head)
```

`g.range` *Range of g-weights*

Description

Computes the range of the ratios between calibrated weights and direct weights (*g-weights*) for the original sample and all its replicates.

Usage

```
g.range(cal.deskott)
```

Arguments

`cal.deskott`	Object of class `kott.cal.design`.

Details

This function computes, for the original sample and all its replicates, the smallest interval which contains the ratios between calibrated weights and direct weights.

Value

A `data.frame` object.

Author(s)

Diego Zardetto

See Also

kottcalibrate for calibrating replicate weights and bounds.hint to obtain an hint for calibration problems where range restrictions are imposed on the *g-weights*.

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Calibration:
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calmodel=~x1+x2+x3-1,partition=~regcod,calfun="logit",
            bounds=bounds,aggregate.stage=2)

# Computation of g-weights range:
g.range(kdescal04p)
```

---

kott.addvars *Add variables to replicated objects*

Description

Modifies a kott.design object by adding new variables to it.

Usage

```
kott.addvars(deskott, ...)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
...	tag = expr arguments defining columns to be added to deskott.

Details

This function adds to the data frame contained in deskott the *new* variables defined by the tag = expr arguments. A tag can be specified either by means of an identifier or by a character string; expr can be any expression that it makes sense to evaluate in the deskott environment.

For each argument tag = expr bound to the formal argument ... the added column will have *name* given by the tag value and *values* obtained by evaluating the expr expression on deskott. Any input expression unsupplied with a tag will be ignored and will therefore have no effect on the kott.addvars return value.

Variables to be added to the input replicated object have to be *new*: namely it is not possible to use kott.addvars to modify the values in a pre-existing deskott column.

Value

An object of the same class of deskott, containing new variables but supplied with exactly the same metadata.

Author(s)

Diego Zardetto

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Adding the new 'population' variable to estimate the number
# of final units in the population:
kdes2<-kott.addvars(kdes,population=1)
kottby(kdes2,~population)

# Recoding a qualitative variable:
kdes2<-kott.addvars(kdes,agerange=as.factor(ifelse(age5c==1,
                                        "young","not-young")))
kottby(kdes2,~agerange,est="mean")
kottby(kdes2,~income,~agerange,estimator="mean",conf.int=TRUE)

# Algebraic operations on numeric variables:
kdes2<-kott.addvars(kdes,q=income/z^2)
kottby(kdes2,~q)
```

---

kott.quantile *Estimation of quantiles*

Description

Calculates estimates, standard errors and confidence intervals for quantiles in subpopulations.

Usage

```
kott.quantile(deskott, y, probs = c(0.25,0.50,0.75), by = NULL,
              vartype = c("se", "cv", "cvpct", "var"),
              conf.int = FALSE, conf.lev = 0.95)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
y	Formula defining the variable of interest.
probs	Vector of probability values to be used to calculate the quantiles estimates. The default value selects the quartiles estimates.
by	Formula specifying the variables that define the "estimation domains". If NULL (the default option) estimates refer to the whole population.

vartype	character vector specifying the desired variability estimators. It is possible to choose one or more of: standard error (the default), coefficient of variation, percent coefficient of variation, or variance.
conf.int	Boolean (logical) value to request confidence intervals for the estimates: the default is FALSE.
conf.lev	Probability specifying the desired confidence level: the default value is 0.95.

Details

This function calculates weighted estimates for the quantiles of a quantitative variable using suitable weights depending on the class of deskott: calibrated weights for class kott.cal.design and direct weights otherwise. Standard errors are calculated using the extended DAGJK method [Kott 99-01].

The mandatory argument y identifies the variable of interest, that is the variable for which quantiles estimates are to be calculated. The deskott variable referenced by y must be numeric and must not contain any missing value (NA).

The optional argument probs specifies the probability values (0<=probs[i]<=1) for which quantiles estimates must be calculated; the default option selects quartiles estimates. If probs[i] is equal to 0 (1) the corresponding "estimate" produced by kott.quantile coincides with the smallest (largest) observed value for the y variable.

The optional argument by specifies the variables that define the "estimation domains", that is the subpopulations for which the estimates are to be calculated. If by=NULL (the default option), the estimates produced by kottby refer to the whole population. Estimation domains must be defined by a formula: for example the statement by=~B1:B2 selects as estimation domains the subpopulations determined by crossing the modalities of variables B1 and B2. The deskott variables referenced by by (if any) must be factor and must not contain any missing value (NA).

The conf.int argument allows to request the confidence intervals for the estimates. By default conf.int=FALSE, that is the confidence intervals are not provided.

Whenever confidence intervals are requested (i.e. conf.int=TRUE), the desired confidence level can be specified by means of the conf.lev argument. The conf.lev value must represent a probability (0<=conf.lev<=1) and its default is chosen to be 0.95. Given an input kott.design object with nrg random groups, kott.quantile builds the confidence intervals making use of a t distribution with nrg-1 degrees of freedom.

Value

The return value depends on the value of the input parameters. In the most general case, the function returns an object of class list (typically a list made up of data frames).

Warning

It may happen that, in certain subpopulations, some of the nrg replicate weights turn out to be all zero: for these replicates it is not possible to provide quantiles estimates. In these cases, kott.quantile (i) returns NaN for the corresponding standard errors and (ii) prints a warning message.

Note

Let $\hat{F}_y$ be the estimate of the cumulative distribution of the y variabile. If an observed value y^* exists such that $\hat{F}_y(y^*) = probs[i]$ than the i-th quantile estimate provided by kott.quantile equals y^*. If this is not the case, the kott.quantile function (i) finds the two observed values

y^- and y^+ ($y^- < y^+$) such that the corresponding values $\hat{F}_y(y^-)$ and $\hat{F}_y(y^+)$ are the closest to $probs[i]$, (ii) linearly interpolates $\hat{F}_y$ between $\hat{F}_y(y^-)$ and $\hat{F}_y(y^+)$ and (iii) estimates the i-th quantile by inverting the linear approximation in the point $probs[i]$.

The rigorous results of [kott 99-01] show that the DAGJK variance estimator for a given estimator $\hat{\theta}$ is correct provided that PSUs are sampled with replacement and that $\hat{\theta}$ is a smooth function of total estimators. As a result, it is not possible to guarantee that the DAGJK quantile variance estimator provided by `kott.quantile` is not biased.

Author(s)

Diego Zardetto

References

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

`kottby` for estimating totals and means, `kott.ratio` for estimating ratios between totals, `kott.regcoef` for estimating regression coefficients and `kottby.user` for calculating estimates based on user-defined estimators.

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Estimate of the deciles of the income variable for
# the whole population:
kott.quantile(kdes,~income,probs=seq(0.1,0.9,0.1))

# Estimate of the median of income by age5c:
kott.quantile(kdes,~income,probs=0.5,by=~age5c,conf.int=TRUE)

# "Estimate" of the minimum and maximum of income by sex
# (notice the value of SE):
kott.quantile(kdes,~income,probs=c(0,1),by=~sex)
```

`kott.ratio` *Estimation of ratios between totals*

Description

Calculates estimates, standard errors and confidence intervals for ratios between totals in subpopulations.

Usage

```
kott.ratio(deskott, num, den, by = NULL,
           vartype = c("se", "cv", "cvpct", "var"),
           conf.int = FALSE, conf.lev = 0.95)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
num	Formula defining the numerator variables for the ratio estimator.
den	Formula defining the denominator variables for the ratio estimator.
by	Formula specifying the variables that define the "estimation domains". If NULL (the default option) estimates refer to the whole population.
vartype	character vector specifying the desired variability estimators. It is possible to choose one or more of: standard error (the default), coefficient of variation, percent coefficient of variation, or variance.
conf.int	Boolean (logical) value to request confidence intervals for the estimates: the default is FALSE.
conf.lev	Probability specifying the desired confidence level: the default value is 0.95.

Details

This function calculates weighted estimates for ratios between totals of quantitative variables using suitable weights depending on the class of deskott: calibrated weights for class kott.cal.design and direct weights otherwise. Standard errors are calculated using the extended DAGJK method [Kott 99-01].

The mandatory argument num (den) identifies the variables whose totals appear as the numerator (denominator) in the ratio estimator: the corresponding formula must be of the type num=~num1+...+numk (den=~den1+...+denl). The function calculates estimates for ratios between homologous variables in num and den; if num and den contain a different number of variables the shortest argument will be tacitly recycled. The deskott variables referenced by num (den) must be numeric and must not contain any missing value (NA).

The optional argument by specifies the variables that define the "estimation domains", that is the subpopulations for which the estimates are to be calculated. If by=NULL (the default option), the estimates produced by kottby refer to the whole population. Estimation domains must be defined by a formula: for example the statement by=~B1:B2 selects as estimation domains the subpopulations determined by crossing the modalities of variables B1 and B2. The deskott variables referenced by by (if any) must be factor and must not contain any missing value (NA).

The conf.int argument allows to request the confidence intervals for the estimates. By default conf.int=FALSE, that is the confidence intervals are not provided.

Whenever confidence intervals are requested (i.e. conf.int=TRUE), the desired confidence level can be specified by means of the conf.lev argument. The conf.lev value must represent a probability (0<=conf.lev<=1) and its default is chosen to be 0.95. Given an input kott.design object with nrg random groups, kott.ratio builds the confidence intervals making use of a t distribution with nrg-1 degrees of freedom.

Value

The return value depends on the value of the input parameters. In the most general case, the function returns an object of class list (typically a list made up of data frames).

Warning

It is possible that, in certain subpopulations, the estimate of the total of some den variables turns out to be zero for the original sample in deskott and/or for some of its nrg replicates. In these cases, kott.ratio (i) returns NaN for the estimates and/or for the corresponding standard errors and (ii) prints a warning message.

Author(s)

Diego Zardetto

References

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

kottby for estimating totals and means, kott.quantile for estimating quantiles, kott.regcoef for estimating regression coefficients and kottby.user for calculating estimates based on user-defined estimators.

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Estimate of the ratios y1/x1, y2/x2 e y3/x3 by marstat:
kott.ratio(kdes,~y1+y2+y3,~x1+x2+x3,by=~marstat)

# Estimate of the ratios z/x1, z/x2 e z/x3
# for the whole population (notice the recycling rule):
kott.ratio(kdes,~z,~x1+x2+x3,conf.int=TRUE)

# Estimators of means can be thought as
# ratio estimators:
kottby(kdes,~income,estimator="mean")
kott.ratio(kott.addvars(kdes,population=1),num=~income,den=~population)
```

---

kott.regcoef *Estimation of linear regression coefficients*

Description

Calculates estimates, standard errors and confidence intervals for regression coefficients in subpopulations.

Usage

```
kott.regcoef(deskott, model, by = NULL,
             vartype = c("se", "cv", "cvpct", "var"),
             conf.int = FALSE, conf.lev = 0.95)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
model	Formula giving a symbolic description of the linear model.
by	Formula specifying the variables that define the "estimation domains". If NULL (the default option) estimates refer to the whole population.
vartype	character vector specifying the desired variability estimators. It is possible to choose one or more of: standard error (the default), coefficient of variation, percent coefficient of variation, or variance.
conf.int	Boolean (logical) value to request confidence intervals for the estimates: the default is FALSE.
conf.lev	Probability specifying the desired confidence level: the default value is 0.95.

Details

This function calculates weighted estimates of linear regression coefficients using suitable weights depending on the class of deskott: calibrated weights for class kott.cal.design and direct weights otherwise. Standard errors are calculated using the extended DAGJK method [Kott 99-01].

The mandatory argument model specifies, by means of a symbolic formula, the linear regression model whose coefficients are to be estimated. model must have the form response ~ terms where response is the (numeric) response variable and terms represents a series of terms which specifies a linear predictor for response. Variables referenced by model must not contain any missing value (NA).

The optional argument by specifies the variables that define the "estimation domains", that is the subpopulations for which the estimates are to be calculated. If by=NULL (the default option), the estimates produced by kottby refer to the whole population. Estimation domains must be defined by a formula: for example the statement by=~B1:B2 selects as estimation domains the subpopulations determined by crossing the modalities of variables B1 and B2. The deskott variables referenced by by (if any) must be factor and must not contain any missing value (NA).

The conf.int argument allows to request the confidence intervals for the estimates. By default conf.int=FALSE, that is the confidence intervals are not provided.

Whenever confidence intervals are requested (i.e. conf.int=TRUE), the desired confidence level can be specified by means of the conf.lev argument. The conf.lev value must represent a probability (0<=conf.lev<=1) and its default is chosen to be 0.95. Given an input kott.design object with nrg random groups and a regression model with p predictors plus an intercept term, kott.regcoef builds the confidence intervals making use of a t distribution with nrg-p-1 degrees of freedom.

Value

The return value depends on the value of the input parameters. In the most general case, the function returns an object of class list (typically a list made up of data frames).

Author(s)

Diego Zardetto

References

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

kottby for estimating totals and means, kott.ratio for estimating ratios between totals, kott.quantile for estimating quantiles and kottby.user for calculating estimates based on user-defined estimators.

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)


# A model with one predictor and no intercept:
kott.regcoef(kdes,income~z-1)

# ...compare with ratio estimator:
kott.ratio(kott.addvars(kdes,income.mult.z=income*z,z2=z^2),~income.mult.z,~z2)


# A model with a factor term and no intercept:
kott.regcoef(kdes,income~age5c-1)

# ...compare with mean estimator in subpopulations:
kottby(kdes,~income,~age5c,estimator="mean")

# ...and with regression coefficients (for a different model)
# in subpopulations:
kott.regcoef(kdes,income~1,~age5c)


# An awkward model with many coefficients:
kott.regcoef(kdes,income~z:age5c+x3+marstat-1)
```

---

kottby *Estimation of totals and means*

Description

Calculates estimates, standard errors and confidence intervals for totals and means in subpopulations.

Usage

```
kottby(deskott, y, by = NULL, estimator = c("total", "mean"),
       vartype = c("se", "cv", "cvpct", "var"),
       conf.int = FALSE, conf.lev = 0.95)
```

Arguments

deskott	Object of class kott.design containing the replicated survey data.
y	Formula defining the variables of interest.
by	Formula specifying the variables that define the "estimation domains". If NULL (the default option) estimates refer to the whole population.
estimator	character specifying the desired estimator: it may be "total" (the default) or "mean".
vartype	character vector specifying the desired variability estimators. It is possible to choose one or more of: standard error (the default), coefficient of variation, percent coefficient of variation, or variance.
conf.int	Boolean (logical) value to request confidence intervals for the estimates: the default is FALSE.
conf.lev	Probability specifying the desired confidence level: the default value is 0.95.

Details

This function calculates weighted estimates for totals and means using suitable weights depending on the class of deskott: calibrated weights for class kott.cal.design and direct weights otherwise. Standard errors are calculated using the extended DAGJK method [Kott 99-01].

The mandatory argument y identifies the variables of interest, that is the variables for which estimates are to be calculated. The corresponding formula must be of the type y=~var1+...+varn. The deskott variables referenced by y must be numeric or factor and must not contain any missing value (NA). It is admissible to specify for y "mixed" formulas that simultaneously contain quantitative (numeric) variables and qualitative (factor) variables.

The optional argument by specifies the variables that define the "estimation domains", that is the subpopulations for which the estimates are to be calculated. If by=NULL (the default option), the estimates produced by kottby refer to the whole population. Estimation domains must be defined by a formula: for example the statement by=~B1:B2 selects as estimation domains the subpopulations determined by crossing the modalities of variables B1 and B2. The deskott variables referenced by by (if any) must be factor and must not contain any missing value (NA).

The optional argument estimator makes it possible to select the desired estimator. If estimator="total" (the default option), kottby calculates, for a given variable of interest vark, the estimate of the total (when vark is numeric) or the estimate of the absolute frequency distribution (when vark is factor). Similarly, if estimator="mean", the function calculates the estimate of the mean (when vark is numeric) or the the estimate of the relative frequency distribution (when vark is factor).

The conf.int argument allows to request the confidence intervals for the estimates. By default conf.int=FALSE, that is the confidence intervals are not provided.

Whenever confidence intervals are requested (i.e. conf.int=TRUE), the desired confidence level can be specified by means of the conf.lev argument. The conf.lev value must represent a probability (0<=conf.lev<=1) and its default is chosen to be 0.95.

Value

The return value depends on the value of the input parameters. In the most general case, the function returns an object of class `list` (typically a list made up of data frames).

Note

The advantage of the DAGJK method over the traditional jackknife is that, unlike the latter, it remains computationally manageable even when dealing with "complex and big" surveys (tens of thousands of PSUs arranged in a large number of strata with widely varying sizes). In fact, the DAGJK method is known to provide, for a broad range of sampling designs and estimators, (near) unbiased standard error estimates even with a "small" number (e.g. a few tens) of replicate weights. On the other hand, if the number of replicates is not large, it seems defensible to use a *t distribution* (rather than a normal distribution) for calculating the confidence intervals. In line with what was proposed in [Kott 99-01], given an input `kott.design` object with `nrg` random groups, `kottby` builds the confidence intervals making use of a t distribution with `nrg-1` degrees of freedom.

Author(s)

Diego Zardetto

References

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

`kott.ratio` for estimating ratios between totals, `kott.quantile` for estimating quantiles, `kott.regcoef` for estimating regression coefficients and `kottby.user` for calculating estimates based on user-defined estimators.

Examples

```
data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)


# Estimate of the total of 3 quantitative variables for the whole
# population:
kottby(kdes,~y1+y2+y3)


# Estimate of the total of the same 3 variables by sex:
kottby(kdes,~y1+y2+y3,~sex)


# Estimate of the mean of the same 3 variables by marstat and sex:
kottby(kdes,~y1+y2+y3,~marstat:sex,estimator="mean")
```

```
# Estimate of the absolute frequency distribution of the qualitative
# variable age5c for the whole population:
kottby(kdes,~age5c)


# Estimate of the relative frequency distribution of the qualitative
# variable marstat by sex:
kottby(kdes,~marstat,~sex,estimator="mean")


# The same with confidence intervals at a confidence level of 0.9:
kottby(kdes,~marstat,~sex,estimator="mean",conf.int=TRUE,conf.lev=0.9)


# Quantitative and qualitative variables together: estimate of the
# total for y3 and of the absolute frequency distribution of marstat,
# by sex:
kottby(kdes,~y3+marstat,~sex)


# Lonely PSUs do not give rise to NaNs in the standard errors:
kdes.lpsu<-kottdesign(data=example,ids=~towcod+famcod,strata=~stratum,
           weights=~weight,nrg=15)
kottby(kdes.lpsu,~x1+x2+x3)
```

`kottby.user` *Estimation for user-defined estimators*

Description

Calculates estimates, standard errors and confidence intervals for user-defined estimators (even non-analytic) in subpopulations.

Usage

```
kottby.user(deskott, by = NULL, user.estimator, na.replace = NULL,
            vartype = c("se", "cv", "cvpct", "var"),
            conf.int = FALSE, conf.lev = 0.95,
            df = attr(deskott, "nrg") - 1, ...)

global(deskott)
```

Arguments

`deskott`	Object of class `kott.design` containing the replicated survey data.
`by`	Formula specifying the variables that define the "estimation domains". If `NULL` (the default option) estimates refer to the whole population.
`user.estimator`	R function to compute the value of the desired estimator on the original survey sample (see also 'Details' and 'Defining a user estimator function').
`na.replace`	Value to be used to replace any `NA`s in the output estimates (see 'Details').

vartype	character vector specifying the desired variability estimators. It is possible to choose one or more of: standard error (the default), coefficient of variation, percent coefficient of variation, or variance.
conf.int	Boolean (logical) value to request confidence intervals for the estimates: the default is FALSE.
conf.lev	Probability specifying the desired confidence level: the default value is 0.95.
df	Degrees of freedom for the t distribution used to build confidence intervals (see 'Details').
...	Additional parameters (if any) to be passed to the user.estimator function.

Details

The kottby.user function is designed to fully exploit the versatility of the DAGJK [Kott 99-01] replication method. It is intended to provide the user with a user-friendly tool for calculating estimates, standard errors and confidence intervals for estimators defined by the user themselves. As is obvious, weighted estimates for the *"user-defined estimator"* are computed using suitable weights depending on the class of deskott: calibrated weights for class kott.cal.design and direct weights otherwise.

The optional argument by specifies the variables that define the "estimation domains", that is the subpopulations for which the estimates are to be calculated. If by=NULL (the default option), the estimates produced by kottby refer to the whole population. Estimation domains must be defined by a formula: for example the statement by=~B1:B2 selects as estimation domains the subpopulations determined by crossing the modalities of variables B1 and B2. The deskott variables referenced by by (if any) must be factor and must not contain any missing value (NA).

The mandatory argument user.estimator is used to specify the calculation method for the "user-defined estimator". In more precise terms: the value bound to the formal argument user.estimator must be a function (an R object of class function, even anonymous) able to compute the value of the required estimator on the sample data frame contained in deskott. It is not necessary for the user.estimator function's return value to be a single numerical value (it can be a vector, a matrix, an array, ...). In any case, it will be tacitly coerced to array by kottby.user. More detailed indications on how the user.estimator function must be constructed can be found in the 'Defining a user estimator function' section below.

The optional argument na.replace makes it possible to specify a value to be used to replace any missing values generated by user.estimator in the kottby.user function output. By default na.replace=NULL and the missing values are returned as NAs.

The conf.int argument allows to request the confidence intervals for the estimates. By default conf.int=FALSE, that is the confidence intervals are not provided.

Whenever confidence intervals are requested (i.e. conf.int=TRUE), the desired confidence level can be specified by means of the conf.lev argument. The conf.lev value must represent a probability (0<=conf.lev<=1) and its default is chosen to be 0.95.

Given an input kott.design object with nrg random groups, *by default* kottby.user builds the confidence intervals making use of a t distribution with nrg-1 degrees of freedom. Indeed the argument df has a default value of nrg-1. Notice, however, that this default value should be used only when the user-defined function user.estimator estimates a univariate parameter of interest. As an example, if user.estimator were designed to estimate regression coefficients for a multiple linear regression with p predictors and no intercept, the right choice would be df = nrg-p.

The special argument ... (*dot-dot-dot*) allows to specify additional parameters to be passed to the user-defined user.estimator function.

Value

The return value depends on the value of the input parameters. In the most general case, the function returns an object of class `list` (typically a list made up of data frames).

Defining a user estimator function

In order to be correctly invoked by `kottby.user`, the function that codifies the "user-defined estimator" must comply with specific *syntactical* restrictions. On the other hand there is not any constraint (at least in principle) on the *semantics* of the function, that is on "what it calculates".
The fundamental constraint is that the function's formal arguments list meets some minimal requirements. Suppose, for simplicity, that the function bound to the `user.estimator` formal argument is named `user.estfun`; than its structure must necessarily be of the following type:

`user.estfun=function(data,weights,etc){body}` [1]

The structure [1] has to be interpreted as follows: `user.estfun` body must contain all the instructions that would make it possible to compute the required estimator on the sample data contained in the `data` data frame using the weights contained in its `weights` column. The `"etc"` symbol represents in [1] any other `user.estfun`'s formal arguments whose actual values can be specified, when invoking `kottby.user`, using its special argument `...` (*dot-dot-dot*).

Sometimes users may need to employ "global" quantities in the body of the `user.estfun` function, that is, quantities that, even when dealing with sub-population estimates, *should not be re-calculated* for the sub-populations themselves (the latter being the standard `kottby.user` behaviour). This need is met by the `global` function: the user has only to reference, wherever the need arises, the `user.estfun` input data frame by means of the `global(data)` expression rather than the standard one `data`.
The `global` function only accepts `kott.design` class objects and can only be used within functions invoked by `user.estfun`. An example that clearly illustrates the utility of `global` is provided by the calculation of poverty estimates (see the `poverty` function documented in the 'Examples' section below).

Note

The freedom granted to the user in developing the `user.estimator` function has important consequences that are worth highlighting. The key point is that, since only the user knows the semantics of `user.estimator`, he must vouch for its correct functioning. In particular:
(i) The `kottby.user` function must be able to invoke the `user.estimator` function on the `deskott` sample data frame and, if necessary, on its subsets defined by the `by` variables. Consequently, when developing the function, the user must make sure that the instructions in its `body` refer to variables that are actually contained in that data frame. This check could not be done by the `kottby.user` caller function albeit at the expense of limiting the user's freedom in constructing his `user.estimator`;
(ii) In the same way, due to user's freedom in developing `user.estimator`, the `kottby.user` function cannot prevent the generation of missing values in its output. The usefulness of the `na.replace` parameter must, therefore, be considered as purely "cosmetic".

Author(s)

Diego Zardetto

References

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

kottby for estimating totals and means, kott.ratio for estimating ratios between totals, kott.quantile for estimating quantiles and kott.regcoef for estimating regression coefficients.

Examples

```
# Some examples of user-defined estimators and illustration
# of their use via kottby.user. Remember that R functions
# expressing user-defined estimators must comply with the
# condition indicated in [1]. The 3 functions that appear
# in the following examples ('ones', 'ratio' and 'poverty')
# are contained in the data.examples file.
# The 'poverty' function (also) illustrates the correct use
# of the 'global' function.

data(data.examples)

# Creation of a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)


# 1) Estimator of the number of final units in the population.
#    Use the name 'ones' to refer to the R function that
#    expresses the estimator and define it as follows:

#    ones <- function (d, w)
#    ######################################
#    #  Number of final units estimator.  #
#    ######################################
#    {
#        sum(d[, w])
#    }

#    Now using kottby.user is easy, for instance:
kottby.user(kdes,user.estimator=ones)


# 2) Estimator of ratios between totals (or means) for 2
#    quantitative variables. Use the name 'ratio' to refer
#    to the R function that expresses the estimator and
#    define it as follows (notice the use of the etc
#    arguments in [1]):

#    ratio <- function (d, w, num, den)
#    ##########################################
#    #  Ratio estimator for totals (or means)  #
#    #  of quantitative variables.             #
#    ##########################################
#    {
#        sum(d[, w] * d[, num])/sum(d[, w] * d[, den])
#    }
```

```
#    Calculating ratio estimates and standard errors
#    is easy (notice the use of the ... argument
#    of kottby.user):
kottby.user(kdes,user.estimator=ratio,num="y1",den="x1")


# 3) A non-analytic estimator: population percentage
#    with income below the poverty threshold (defined,
#    for the sake of simplicity, as 0.6 times the
#    average income for the whole population).
#    Call 'poverty' the estimator and define it as follows:

#    poverty <- function (d, w, y, threshold)
#    ########################################################################
#    #  Population percentage with income below the poverty threshold.  #
#    #  Suppose poverty threshold is defined as 0.6 times the average   #
#    #  income for the whole population.                                #
#    ########################################################################
#    {
#        if (missing(threshold)) {
#        # if I do want to take into account the variance of the poverty
#        # threshold letting it be re-calculated replicate by replicate.
#            d.global = global(d)
#            th.value = 0.6 * sum(d.global[, w] * d.global[, y])/sum(d.global[, w])
#        }
#        else {
#        # if I do not want to take into account the variance of the poverty
#        # threshold, I will supply its point estimate to the 'threshold' argument.
#            th.value = threshold
#        }
#        est = 100 * sum(d[d[, y] < th.value, w])/sum(d[, w])
#        est
#    }


#    3.1) First use: neglect the variance of the poverty threshold
#         and supply to 'threshold' (by means of the ... argument
#         of kottby.user) its point estimate obtained using kottby:
pov.line<-0.6*kottby(kdes,~income,estimator="mean")$mean
kottby.user(kdes,user.estimator=poverty,y="income",threshold=pov.line)

#    3.2) Second use: do take into account the variance of the poverty
#         threshold letting it be re-calculated replicate by replicate
#         (thus not supplying any actual value to 'threshold'):
kottby.user(kdes,user.estimator=poverty,y="income")

#    Notice that the standard error estimate for the 'poverty' estimator
#    obtained in 3.2) cannot be calculated analytically by Taylor
#    linearization.

#    Notice the use of the 'global' function in the body of 'poverty':
#    since the poverty status of each final unit depends on a global
#    value (that is, the average income for the whole population)
#    'global' is used to prevent, whenever a sub-population poverty
#    estimate is needed, this global value being calculated locally
#    i.e. within the sub-population itself.
#    In fact:
```

```
pov.line<-0.6*kottby(kdes,~income,estimator="mean")$mean
kdes2<-kott.addvars(kdes,pov.status=as.factor(ifelse(income<pov.line,
                                                     "poor","not-poor")))
kottby.user(kdes2,by=~pov.status,user.estimator=poverty,y="income")

#    If the 'global' function were not used in 'poverty'
#    the poverty threshold would be calculated relative to
#    each individual sub-population:

poverty2 <- function (d, w, y, threshold)
###############################################
#  Whithout relying on the 'global' function  #
###############################################
{
    if (missing(threshold)) {
        th.value = 0.6 * sum(d[, w] * d[, y])/sum(d[, w])
    }
    else {
        th.value = threshold
    }
    est = 100 * sum(d[d[, y] < th.value, w])/sum(d[, w])
    est
}

kottby.user(kdes2,by=~pov.status,user.estimator=poverty2,y="income")

#    This means that without 'global' a non-null fraction of poors
#    would be paradoxically estimated for the "non-poors" sub-population
#    (and, conversely, a non-null fraction of non-poors among the "poors").
```

`kottcalibrate` *Calibration of replicate weights*

Description

Adds to a `kott.design` object the calibrated weights columns (one for each replicate weight, plus one for the initial weights).

Usage

```
kottcalibrate(deskott, df.population,
        calmodel = if (inherits(df.population, "pop.totals"))
                       attr(df.population, "calmodel"),
        partition = if (inherits(df.population, "pop.totals"))
                        attr(df.population, "partition") else FALSE,
        calfun = c("linear", "raking", "logit"),
        bounds = c(-Inf, Inf), aggregate.stage = NULL, maxit = 50,
        epsilon = 1e-07, force.rep = FALSE)
```

Arguments

`deskott`	Object of class `kott.design` containing the replicated survey data.
`df.population`	Data frame containing the known population totals for the auxiliary variables.

calmodel	Formula defining the linear structure of the calibration model.
partition	Formula specifying the variables that define the "calibration domains" for the model (see 'Details'); FALSE (the default) implies no calibration domains.
calfun	character specifying the distance function for the calibration process; the default is "linear".
bounds	Allowed range for the ratios between calibrated and initial weights; the default is c(-Inf,Inf).
aggregate.stage	An integer: if specified, causes the calibrated weights to be constant within sampling units at this stage.
maxit	Maximum number of iterations for the Newton-Raphson algorithm; the default is 50.
epsilon	Tolerance for the relative differences between the population totals and the corresponding estimates based on the claibrated weights; the default is 10^-7.
force.rep	If TRUE, whenever the calibration algorithm does not converge for a given set of replicate weights, forces the function to return a value (see 'Details'); the default is FALSE.

Details

This function creates an object of class kott.cal.design. A kott.cal.design object is made up by the union of the (calibrated) replicated survey data and the metadata describing the sampling design. kott.cal.design objects make it possible to estimate the variance of calibration estimators [Deville, Sarndal 92] using the extended *"Delete-A-Group Jackknife"* method [Kott 2008].

The mandatory argument calmodel symbolically defines the calibration model you want to use, that is - in the language of the generalised regression estimator - the assisting linear regression model underlying the calibration problem [Wilkinson, Rogers 73]. More specifically, the calmodel formula identifies the auxiliary variables and the constraints for the calibration problem. For example, calmodel=~(X+Z):C+(A+B):D defines the calibration problem in which constraints are imposed: (i) on the auxiliary (quantitative) variables X and Z within the subpopulations identified by the (qualitative) classification variable C and, at the same time, (ii) on the absolute frequency of the (qualitative) variables A and B within the subpopulations identified by the (qualitative) classification variable D.
The deskott variables referenced by calmodel must be numeric or factor and must not contain any missing value (NA).

Problems for which one or more qualitative variables can be *"factorised"* in the formula that specifies the calibration model, are particularly interesting. These variables split the population into non-overlapping subpopulations known as *"calibration domains"* for the model. An example is provided by the statement calmodel=~(A+B+X+Z):D in which the variable that identifies the calibration domains is D; similarly, the formula calmodel=~(A+B+X+Z):D1:D2 identifies as calibration domains the subpopulations determined by crossing the modalities of D1 and D2. The interest in models of this kind lies in the fact that the *global* calibration problem they describe can, actually, be broken down into *local* subproblems, one per calibration domain, which can be solved separately [Vanderhoeft 01]. Thus, for example, the global problem defined by calmodel=~(A+B+X+Z):D is equivalent to the sequence of problems defined by the *"reduced model"* calmodel=~A+B+X+Z in each of the domains identified by the modalities of D. The opportunity to separately solve the subproblems related to different calibration domains achieves a significant reduction in computation complexity: the gain increases with increasing survey data size and (most importantly) with increasing auxiliary variables number.

The optional argument partition makes it possible to choose, in cases in which the calibration problem can be factorised, whether to solve the problem globally or iteratively (that is, separately for each calibration domain). The global solution (which is the default option) can be selected invoking the kottcalibrate function with partition=FALSE. To request the iterative solution - a strongly recommended option when dealing with a lot of auxiliary variables and big data sizes - it is necessary to specify via partition the variables defining the calibration domains for the model. If a formula is passed through the partition argument (for example: partition=~D1:D2), the program checks that calmodel actually describes a "reduced model" (for example: calmodel=~X+Z+A+B), that is it does not reference any of the partition variables; if this is not the case, the program stops and prints an error message.
The deskott variables referenced by partition (if any) must be factor and must not contain any missing value (NA).

The mandatory argument df.population is used to specify the known totals of the auxiliary variables referenced by calmodel within the subpopulations (if any) identified by partition. These known totals must be stored in a data frame whose structure (i) depends on the values of calmodel and partition and (ii) must conform to a standard. In order to facilitate understanding of and compliance with this standard, the **EVER** package provides the user with two functions: pop.template and population.check. The pop.template function is able to guide the user in constructing the known totals data frame for a specific calibration problem, while the population.check function allows to check whether a known totals data frame conforms to the standard required by kottcalibrate. In any case, if the df.population data frame does not comply with the standard, the kottcalibrate function stops and prints an error message: the meaning of the message should help the user diagnose the cause of the problem.

The calfun argument identifies the distance function to be used in the calibration process. Three built-in functions are provided: "linear", "raking", and "logit". The default is "linear", which corresponds to the euclidean metric.

The bounds argument allows to add *"range constraints"* to the calibration problem. To be precise, the interval defined by bounds will contain the values of the ratios between final (calibrated) and initial (direct) weights. The default value is c(-Inf,Inf), i.e. no range constraints are imposed. These constraints are optional unless the "logit" function is selected: in the latter case the range defined by bounds has to be finite.

The value passed by the aggregate.stage argument must be an integer between 1 and the number of sampling stages of deskott. If specified, causes the calibrated weights to be constant within sampling units selected at the aggregate.stage stage (actually this is only ensured if the initial weights had already this property, as is sometimes the case in multistage cluster sampling). If not specified, the calibrated weights may differ even for sampling units with identical initial weights. The same holds if some final units belonging to the same cluster selected at the stage aggregate.stage fall in distinct calibration domains (i.e. if the domains defined by partition "cut across" the aggregate.stage-stage clusters).

The maxit argument sets the maximum number of iteration for the Newton-Raphson algorithm that is used to solve the calibration problem. The default value is 50.

The epsilon argument determines the convergence criterion for the optimisation algorithm: it fixes the maximum allowed value for the relative differences between the population totals and the corresponding estimates based on the claibrated weights. The default value is 10^-7.

If the number of replicates for deskott (the input object of class kott.design) is nrg, the function kottcalibrate is in charge of solving nrg+1 distinct calibration problems. In fact, the calibrated weights calculated by kottcalibrate must ensure that the known population totals are exactly reproduced not only by the original sample, but also by all its nrg replicates. Should this requirement fail, the DAGJK method would end up with a biased variance estimator [Kott 2008]. It is, however, possible (more likely when range constraints are imposed) that, for some of the nrg+1

distinct calibration problems and for the given values of epsilon and maxit, the solving algorithm does not converge. In this case kottcalibrate by default stops and prints an error message. On the contrary if force.rep = TRUE, *provided that the failure to converge pertains only to the replicate weights*, the function is forced to return the best approximation achieved for the corresponding calibrated weights. When this occurs, DAGJK standard errors estimates built on the object returned by kottcalibrate will be biased.

Value

An object of class kott.cal.design. The data frame it contains includes (in addition to the data already stored in deskott) the calibrated weights columns (one for each replicate weight, plus one for the initial weights, nrg+1 in all). The names of these columns are obtained by pasting the name of the initial weights column with the string ".cal" and the indices NULL,1,2,...,nrg.
The kott.cal.design class is a specialisation of the kott.design class; this means that an object created by kottcalibrate inherits from the data.frame class and you can use on it every method defined on that class.

Calibration process diagnostics

If the number of replicates for deskott is nrg, the function kottcalibrate is in charge of solving nrg+1 distinct calibration problems. When, dealing with a factorisable calibration problem, the user selects the iterative solution, each one of the above mentioned problems is split into as many *sub-problems* as the number of subpopulations defined by partition. A calibration process with such a complex structure needs some ad hoc tool for error diagnostics. For this purpose, every call to kottcalibrate creates, by side effect, a dedicated data structure named kottcal.status into the .GlobalEnv. kottcal.status is a list with two components: the first, "call", identifies the call to kottcalibrate that generated the list, the second, return.code, is a matrix each element of which identifies the return code of a specific calibration sub-problem. The meaning of the return codes is as follows:

-1 not yet tackled sub-problem;

0 solved sub-problem (convergence achieved);

1 unsolved sub-problem (no convergence): output forced.

Recall that the latter return code may only occur if force.rep = TRUE.
In case of error, users can exploit kottcal.status to identify the sub-problem from which the error stemmed, hence taking a step forward to eliminate it.

Author(s)

Diego Zardetto

References

Deville, J.C., Sarndal, C.E. (1992) *"Calibration Estimators in Survey Sampling"*, Journal of the American Statistical Association, Vol. 87, No. 418, pp.376-382.

Kott, Phillip S. (2008) *"Building a Better Delete-a-Group Jackknife for a Calibration Estimator"*, NASS Research Report, NASS: Washington, DC.

Wilkinson, G.N., Rogers, C.E. (1973) *"Symbolic Description of Factorial Models for Analysis of Variance"*, Journal of the Royal Statistical Society, series C (Applied Statistics), Vol. 22, pp. 181-191.

Vanderhoeft, C. (2001) *"Generalized Calibration at Statistic Belgium"*, Statistics Belgium Working Paper n. 3, http://www.statbel.fgov.be/studies/paper03_en.asp.

Lumley, T. (2006) *"survey: analysis of complex survey samples"*, http://cran.at.r-project.org/web/packages/survey/index.html.

Scannapieco, M., Zardetto, D., Barcaroli, G. (2007) *"La Calibrazione dei Dati con R: una Sperimentazione sull'Indagine Forze di Lavoro ed un Confronto con GENESEES/SAS"*, Contributi Istat n. 4., http://www.istat.it/dati/pubbsci/contributi/Contributi/contr_2007/2007_4.pdf.

See Also

desc for a concise description of kott.design objects, kottby, kott.ratio, kott.regcoef, kott.quantile and kottby.user for calculating estimates and standard errors, pop.template for constructing known totals data frames in compliance with the standard required by kottcalibrate, population.check to check that the known totals data frame satisfies that standard, bounds.hint to obtain an hint for range restricted calibration.

Examples

```
# Calibration of a kott.design object according to different calibration
# models (the known totals data frames pop01, ..., pop05p and the bounds
# vector are contained in the data.examples file).
# For the examples relating to calibration models that can be factorised
# both a global and an iterative solution are given.

data(data.examples)

# Creation of the object to be calibrated:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)


# 1) Calibration on the total number of units in the population
#   (totals in pop01):
kdescal01<-kottcalibrate(deskott=kdes,df.population=pop01,calmodel=~1,
           calfun="logit",bounds=bounds,aggregate.stage=2)

# Checking the result (the function 'ones' is contained
# in the data.examples file):
kottby.user(kdescal01,user.estimator=ones)


# 2) Calibration on the marginal distributions of sex and marstat
#    (totals in pop02):
kdescal02<-kottcalibrate(deskott=kdes,df.population=pop02,
           calmodel=~sex+marstat-1,calfun="logit",bounds=bounds,
           aggregate.stage=2)

# Checking the result:
kottby(kdescal02,~sex+marstat)


# 3) Calibration (global solution) on the joint distribution of sex
#    and marstat (totals in pop03):
kdescal03<-kottcalibrate(deskott=kdes,df.population=pop03,
           calmodel=~marstat:sex-1,calfun="logit",bounds=bounds)

# Checking the result:
kottby(kdescal03,~sex,~marstat) # or: kottby(kdescal03,~marstat,~sex)
```

```
# which, obviously, is not respected by kdescal02 (notice the size of SE):
kottby(kdescal02,~sex,~marstat)


# 3.1) Again a calibration on the joint distribution of sex and marstat
#      but, this time, with the iterative solution (partition=~sex,
#      totals in pop03p):
kdescal03p<-kottcalibrate(deskott=kdes,df.population=pop03p,
            calmodel=~marstat-1,partition=~sex,calfun="logit",
            bounds=bounds)

# Checking the result:
kottby(kdescal03p,~sex,~marstat)


# 4) Calibration (global solution) on the totals for the quantitative
#    variables x1, x2 and x3 in the subpopulations defined by the
#    regcod variable (totals in pop04):
kdescal04<-kottcalibrate(deskott=kdes,df.population=pop04,
           calmodel=~(x1+x2+x3-1):regcod,calfun="logit",
           bounds=bounds,aggregate.stage=2)

# Checking the result:
kottby(kdescal04,~x1+x2+x3,~regcod)


# 4.1) Same problem with the iterative solution (partition=~regcod,
#      totals in pop04p):
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calmodel=~x1+x2+x3-1,partition=~regcod,calfun="logit",
            bounds=bounds,aggregate.stage=2)

# Checking the result:
kottby(kdescal04p,~x1+x2+x3,~regcod)


# 5) Calibration (global solution) on the total for the quantitative
#    variable x1 and on the marginal distribution of the qualitative
#    variable age5c, in the subpopulations defined by crossing sex
#    and marstat (totals in pop05):
kdescal05<-kottcalibrate(deskott=kdes,df.population=pop05,
           calmodel=~(age5c+x1-1):sex:marstat,calfun="logit",
           bounds=bounds,force.rep=TRUE)

# Calibration process diagnostics:
kottcal.status

# Checking the result:
kottby(kdescal05,~age5c+x1,~sex:marstat)


# 5.1) Same problem with the iterative solution (partition=~sex:marstat,
#      totals in pop05p):
kdescal05p<-kottcalibrate(deskott=kdes,df.population=pop05p,
            calmodel=~age5c+x1-1,partition=~sex:marstat,
            calfun="logit",bounds=bounds,force.rep=TRUE)
```

```
# Calibration process diagnostics:
kottcal.status

# Checking the result:
kottby(kdescal05p,~age5c+x1,~sex:marstat)


# Notice that 3.1 e 5.1) do not impose the aggregate.stage=2
# condition. This condition cannot, in fact, be fulfilled because
# in both cases the domains defined by partition "cut across"
# the kdes second stage clusters (households). To compare the results,
# the same choice was also made for 3) e 5).
```

`kottdesign` *Delete-A-Group Jackknife replication*

Description

Adds to a data frame of survey data the replicate weights calculated according to the *"Delete-A-Group Jackknife"* (DAGJK) method.

Usage

```
kottdesign(data, ids, strata = FALSE, weights, nrg,
           self.rep.str = FALSE, check.data = FALSE,
           aux = FALSE)
```

Arguments

`data`	Data frame of survey data.
`ids`	Formula identifying clusters selected at subsequent sampling stages (PSUs, SSUs, ...).
`strata`	Formula identifying the stratification variable; `FALSE` (the default) implies no stratification.
`weights`	Formula identifying the initial weights for the sampling units.
`nrg`	Number of "random groups" (and replicate weights) you want to create.
`self.rep.str`	Formula identifying self-representing strata (SR), if any; `FALSE` (the default) means no SR strata.
`check.data`	Boolean (`logical`) value to check the correct nesting of `data` clusters; the default is `FALSE`.
`aux`	If `TRUE` adds columns of auxiliary information to the output data frame.

Details

This function creates an object of class `kott.design`. A `kott.design` object is made up by the union of the replicated survey data and the metadata describing the sampling design. The metadata (stored as attributes of the object) are used to enable and guide processing and analyses provided by other functions in the **EVER** package (such as `kottcalibrate`, `kottby`, `desc`, ...).

The data, ids, weights and nrg arguments are mandatory, while strata, check.data and aux arguments are optional. The data variables that are referenced by ids, weights and strata (if specified) must not contain any missing value (NA).

The ids argument specifies the cluster identifiers. It is possible to specify a multi-stage sampling design by simply using a formula with the identifiers of clusters selected at subsequent sampling stages. For example, ids=~id.PSU+id.SSU declares a two-stage sampling in which the first stage units are identified by the id.PSU variable and second stage ones by the id.SSU variable.

The strata argument identifies the stratification variable. The data variable referenced by strata (if specified) must be a factor. By default the sample is assumed to be non-stratified.

The weights argument identifies the initial (or direct) weights for the units included in the sample. The data variable referenced by weights must be numeric.

The nrg argument selects the number of "random groups" (and replicate weights) you want to create by means of the DAGJK method [Kott 98-99-01]. The value of nrg must be greater than 1 and less than or equal to the number of sampled PSUs (otherwise the function stops and prints an error message). If nrg equals the number of sampled PSUs, the DAGJK method "reduces" to (that is, it provides identical results to) the traditional stratified jackknife method. The advantage of the DAGJK method over the traditional jackknife is that, unlike the latter, it remains computationally manageable even when dealing with "complex and big" surveys (tens of thousands of PSUs arranged in a large number of strata with widely varying sizes). In fact, the DAGJK method is known to provide, for a broad range of sampling designs and estimators, (near) unbiased standard error estimates even with a "small" number (e.g. a few tens) of replicate weights.

When dealing with a multistage, stratified sampling design that includes *self-representing (SR) strata* (i.e. strata containing PSUs selected with probability 1), the main contribution to the variance of the SR strata arises from the second stage units (*"variance PSUs"*). In this instance, the user can exploit the self.rep.str argument to specify, by a formula, the data variable identifying the SR strata: as a result the function will build the variance PSUs and take care of them. When choosing this option, the user must ensure that the variable referenced by self.rep.str is logical (with value TRUE for SR strata and FALSE otherwise) or numeric (with value 1 for SR strata and 0 otherwise).
As an alternative, the user can attend to develop by himself the appropriate identifiers for the sampling units in ids. To be precise, the identifier for the PSUs (say id.PSU) must have, in the SR strata, values in correspondence 1:1 (for example they can be equal, provided this does not cause undesired duplications) with those of the SSUs identifier (say id.SSU).

The optional argument check.data allows to check the correct nesting of data clusters (PSUs, SSUs, ...). If check.data=TRUE the function checks that every unit selected at stage k+1 is associated to one and only one unit selected at stage k. For a stratified design the function checks also the correct nesting of clusters within strata.

The optional argument aux can usually be ignored: its default value selects the standard behaviour of the function. Invoking kottdesign with aux=TRUE can, on the other hand, prove useful for any user who wants to fully understand how the DAGJK method builds the replicate weights. If aux=TRUE, the output data frame contains auxiliary columns that provide: the number of PSUs per stratum, the number of PSUs per stratum and random group and the multiplicative coefficients that transform the initial weights into replicate weights.

Value

An object of class kott.design. The data frame it contains includes (in addition to the original survey data):

- A new column named *rgi* (random group index) giving the random group to which each sample unit belongs.

- The replicate weights columns (one per random group, `nrg` in all), the names of which are obtained by pasting the name of the initial weights column with the indices 1, 2, ..., `nrg`.

The `kott.design` class is a specialisation of the `data.frame` class; this means that an object created by `kottdesign` inherits from the `data.frame` class and you can use on it every method defined on that class.

Note

The **EVER** package implements the extended version of the DAGJK method [Kott 99-01]. It guarantees unbiased estimates of standard errors even when the number of PSUs sampled in some strata is small (that is, less than `nrg`).

The rigorous [Kott 98-99-01] results were derived under the hypothesis of with replacement selection of PSUs. This means that the DAGJK method cannot include finite population corrections (*fpc*): this restriction is fully reflected in the **EVER** package.

If only one PSU (*lonely PSU*) has been selected in some non-self-representative strata (NSR), the `kottdesign` function does not report an error message, rather a warning one. In fact, the extended DAGJK method automatically removes the contribution of strata containing lonely PSUs from the estimation of standard errors (obviously, their contribution remains when calculating the estimates). This is all the users have to remember, if they come across the warning message produced by `kottdesign`. Whenever the described behaviour seems to be undesirable, a viable alternative in order to eliminate the lonely PSUs is to collapse strata in a suitable manner. In such a case, the price to pay is the possibility of ending up with an over-estimation of the standard errors. As far as the strata collapsing strategie is concerned, the **EVER** package does not provide (in the current version) any support to the user.

Unlike the conventional jackknife method, the DAGJK is a stochastic replication method. If, having fixed the sampling design and the number of replicates, it is applied a number of times to the same sample data frame, generally a different random groups composition results. This means that repeated invocations of the `kottdesign` function, even if run with identical actual parameters, generate different `kott.design` objects (and, consequently, different standard error estimates). What has been stated obviously does not apply when `nrg` equals the number of sampled PSUs. If you really need it, you can however generate exactly the same results for subsequent applications of `kottdesign`: you have only to keep fixed the seed of R's random numbers generator (using the `set.seed` function).

Author(s)

Diego Zardetto.

References

Kott, Phillip S. (1998) *"Using the Delete-A-Group Jackknife Variance Estimator in NASS Surveys"*, RD Research Report No. RD-98-01, USDA, NASS: Washington, DC.

Kott, Phillip S. (1999) *"The Extended Delete-A-Group Jackknife"*. Bulletin of the International Statistical Instititute. 52nd Session. Contributed Papers. Book 2, pp. 167-168.

Kott, Phillip S. (2001) *"The Delete-A-Group Jackknife"*. Journal of Official Statistics, Vol.17, No.4, pp. 521-526.

See Also

desc for a concise description of kott.design objects, kottby, kott.ratio, kott.regcoef, kott.quantile and kottby.user for calculating estimates and standard errors, kottcalibrate for calibrating replicate weights.

Examples

```
# Creation of kott.design objects starting with survey data sampled
# with different sampling designs (actually the survey data frame is
# always the same: the examples serve the purpose of illustrating
# the syntax).

data(data.examples)

# Two-stage stratified cluster sampling design (notice the presence of
# lonely PSUs):
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~stratum,
      weights=~weight,nrg=15)
desc(kdes)


# The same using collapsed strata (SUPERSTRATUM variable) to remove
# lonely PSUs:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)
desc(kdes)


# Same design, but using the self.rep.str argument to identify
# the SR strata (actually towcod identifies the
# "variance PSUs" by construction):
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15,self.rep.str=~sr)
desc(kdes)


# Two stage cluster sampling (no stratification):
kdes<-kottdesign(data=example,ids=~towcod+famcod,weights=~weight,nrg=15)
desc(kdes)


# One-stage stratified cluster sampling:
kdes<-kottdesign(data=example,ids=~towcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)
desc(kdes)


# Stratified independent sampling design:
kdes<-kottdesign(data=example,ids=~key,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)
desc(kdes)
```

---

pop.template *Template data frame for known population totals*

Description

Construct a *"template"* data frame to store known population totals for a calibration problem.

Usage

```
pop.template(data, calmodel, partition = FALSE)
```

Arguments

data	Data frame of survey data.
calmodel	Formula defining the linear structure of the calibration model.
partition	Formula specifying the variables that define the "calibration domains" for the model. FALSE (the default) implies no calibration domains.

Details

This function creates an object of class pop.totals. A pop.totals object is made up by the union of a data frame whose structure conforms to the standard required by kottcalibrate for the known totals and the metadata describing the calibration problem.

The mandatory argument data must identify the survey data frame on which the calibration problem is defined (or, as an alternative, a kott.design object built upon that data frame).

The mandatory argument calmodel symbolically defines the calibration model you want to use: it identifies the auxiliary variables and the constraints for the calibration problem. The deskott variables referenced by calmodel must be numeric or factor and must not contain any missing value (NA).

The optional argument partition specifies the variables that define the calibration domains for the model. The default value (FALSE) means either that there are not calibration domains or that you want to solve the problem globally (even though it could be factorised). If a formula is passed through the partition argument the program checks that calmodel actually describes a "reduced model", that is it does not reference any of the partition variables; if this is not the case, the program stops and prints an error message. The deskott variables referenced by partition (if any) must be factor and must not contain any missing value (NA).

Value

An object of class pop.totals. The data frame it contains is a *"template"* in the sense that all the known totals it must be able to store are missing (NA). However, this data frame has a structure that complies with the standard required by kottcalibrate (provided the latter is invoked with the same calmodel and partition values used to create the template).
The operation of filling the template's NAs with the actual values of the corresponding population totals is, obviously, to be done by the user.

The pop.totals class is a specialisation of the data.frame class; this means that an object built by pop.template inherits from the data.frame class and you can use on it every method defined on that class.

Author(s)

Diego Zardetto

See Also

kottcalibrate for calibrating replicate weights, population.check to check that the known totals data frame satisfies the standard required by kottcalibrate.

Examples

```
# Creation of population totals template data frames for different
# calibration problems (if the calibration models can be factorised
# both a global and an iterative solution are given):

data(data.examples)

# 1) Calibration on the total number of units in the population:
pop.template(data=example,calmodel=~1)


# 2) Calibration on the total number of units in the population
#    and on the marginal distribution of marstat (notice that the
#    total for the first level "married" of the marstat factor
#    variable is missing because it can be deduced from
#    the remaining totals):
pop.template(data=example,calmodel=~marstat)


# 3) Calibration on the marginal distribution of marstat (you
#    must explicitly remove the intercept term in the
#    calibration model adding -1 to the calmodel formula):
pop.template(data=example,calmodel=~marstat-1)


# 4) Calibration (global solution) on the joint distribution of sex
#    and marstat:
pop.template(data=example,calmodel=~sex:marstat-1)

# 4.1) Calibration (iterative solution) on the joint distribution
#      of sex and marstat:
#      4.1.1) Using sex to define calibration domains:
pop.template(data=example,calmodel=~marstat-1,partition=~sex)

#      4.1.2) Using marstat to define calibration domains:
pop.template(data=example,calmodel=~sex-1,partition=~marstat)


# 5) Calibration (global solution) on the total for the quantitative
#    variable x1 and on the marginal distribution of the qualitative
#    variable age5c, in the subpopulations defined by crossing sex
#    and marstat:
pop.template(data=example,calmodel=~(age5c+x1-1):sex:marstat)

# 5.1) The same problem with iterative solutions:
#      5.1.1) Using sex to define calibration domains:
pop.template(data=example,calmodel=~(age5c+x1-1):marstat,partition=~sex)

#      5.1.2) Using marstat to define calibration domains:
pop.template(data=example,calmodel=~(age5c+x1-1):sex,partition=~marstat)
```

```
#      5.1.3) Using sex and marstat to define calibration domains:
pop.template(data=example,calmodel=~age5c+x1-1,partition=~sex:marstat)
```

`population.check` *Compliance test for known totals data frames*

Description

Checks whether a known population totals data frame conforms to the standard required by `kottcalibrate` for a specific calibration problem.

Usage

```
population.check(df.population, data, calmodel, partition = FALSE)
```

Arguments

`df.population`	Data frame of known population totals.
`data`	Data frame of survey data.
`calmodel`	Formula defining the linear structure of the calibration model.
`partition`	Formula specifying the variables that define the "calibration domains" for the model. `FALSE` (the default) implies no calibration domains.

Details

The behaviour of this function depends on the outcome of the test. If `df.population` is found to conform to the standard, the function first converts it into an object of class `pop.totals` and then invisibly returns it. Failing this, the function stops and prints an error message: the meaning of the message should help the user diagnose the cause of the problem.

The mandatory argument `df.population` identifies the known totals data frame for which compliance with the standard is to be checked.

The mandatory argument `data` identifies the survey data frame on which the calibration problem is defined (or, as an alternative, a `kott.design` object built upon that data frame).

The mandatory argument `calmodel` symbolically defines the calibration model you want to use: it identifies the auxiliary variables and the constraints for the calibration problem. The `data` variables referenced by `calmodel` must be `numeric` or `factor` and must not contain any missing value (`NA`).

The optional argument `partition` specifies the variables that define the calibration domains for the model. The default value (`FALSE`) means either that there are not calibration domains or that you want to solve the problem globally (even though it could be factorised). If a formula is passed through the `partition` argument the program checks that `calmodel` actually describes a "reduced model", that is it does not reference any of the partition variables; if this is not the case, the program stops and prints an error message. The `data` variables referenced by `partition` (if any) must be `factor` and must not contain any missing value (`NA`).

Value

An invisibile object of class `pop.totals`. The `pop.totals` class is a specialisation of the `data.frame` class; this means that an object built by `pop.template` inherits from the `data.frame` class and you can use on it every method defined on that class.

Note

The population.check function can be used to convert a known totals data frame that conforms to the standard required by kottcalibrate into an object of class pop.totals. The usefulness of this conversion lies in the fact that, once you have known totals with this "certified format", you can invoke kottcalibrate without specifying the values for the calmodel and partition arguments (this means that the function is able to extract them directly from the attributes of the pop.totals object).

Author(s)

Diego Zardetto

See Also

kottcalibrate for calibrating replicate weights, pop.template for the definition of the class pop.totals and to build a "template" data frame for known population totals.

Examples

```
data(data.examples)

# Suppose you have to calibrate the example survey data frame
# on the totals of x1 by sex and you want the iterative solution.
# Start creating a kott.design object:
kdes<-kottdesign(data=example,ids=~towcod+famcod,strata=~SUPERSTRATUM,
      weights=~weight,nrg=15)

# Then build a template data frame for the known totals:
pop<-pop.template(data=example,calmodel=~x1-1,partition=~sex)
pop
class(pop)

# Now fill NAs with the actual values for the population
# totals (suppose 123 for sex="f" and 456 for sex="m"):
pop[,"x1"]<-c(123,456)
pop
class(pop)

# Finally check if pop complies with the kottcalibrate standard:
population.check(df.population=pop,data=example,calmodel=~x1-1,
                 partition=~sex)


# If, despite keeping the content unchanged, we altered the
# structure of the data frame (for example, by changing the
# order of its rows)...
pop.mod<-pop ; pop.mod[1,]<-pop[2,] ; pop.mod[2,]<-pop[1,]
pop
pop.mod

# ... we would obtain an error:
## Not run: population.check(df.population=pop.mod,data=example,calmodel=~x1-1,
                           partition=~sex)
## End(Not run)
```

```
# Remember that, if the known totals have been converted
# into the pop.totals "format" by means of population.check,
# it is possible to invoke kottcalibrate without specifying
# calmodel and partition:

class(pop04p)
pop04p
kdescal04p<-kottcalibrate(deskott=kdes,df.population=pop04p,
            calfun="logit",bounds=bounds,aggregate.stage=2)


# ... this option is not allowed if the known totals
# are not of class pop.totals even if they conform to the
# standard:

pop04p.mod=data.frame(pop04p)
class(pop04p.mod)
pop04p.mod
## Not run: kottcalibrate(deskott=kdes,df.population=pop04p.mod,calfun="logit",
              bounds=bounds,aggregate.stage=2)
## End(Not run)
```

Index