Package ‘survey’

March 20, 2024

Title Analysis of Complex Survey Samples


Version 4.4-2

Author Thomas Lumley, Peter Gao, Ben Schneider

Maintainer ``Thomas Lumley'' <t.lumley@auckland.ac.nz>

License GPL-2 | GPL-3

Depends R (>= 4.1.0), grid, methods, Matrix, survival

Imports stats, graphics, splines, lattice, minqa, numDeriv, mitools

>= 2.4, Rcpp (>= 0.12.8)

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder R.rsp

Suggests foreign, MASS, KernSmooth, hexbin, RSQLite, quantreg, parallel, CompQuadForm, DBI, AER, SUMMER (>= 1.4.0), R.rsp

URL http://r-survey.r-forge.r-project.org/survey/

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-03-20 15:30:02 UTC

R topics documented:

anova.svyglm .................................................. 3
api ............................................................ 6
as.fpc ........................................................ 9
as.svreps .......
### R topics documented:

- `bootweights` ................................................................. 14
- `brrweights` ................................................................. 15
- `calibrate` ................................................................. 18
- `compressWeights` ......................................................... 24
- `confint.svyglm` ......................................................... 25
- `crowd` ................................................................. 26
- `dimnames.DBIsvydesign` ............................................. 27
- `election` ................................................................. 28
- `estweights` ................................................................. 29
- `fpc` ................................................................. 31
- `ftable.svystat` ......................................................... 32
- `hadamard` ................................................................. 33
- `hospital` ................................................................. 35
- `HR` ................................................................. 36
- `make.calfun` ............................................................... 37
- `marginpred` ............................................................... 38
- `mu284` ................................................................. 39
- `myco` ................................................................. 40
- `newsvyquantile` ......................................................... 41
- `nhanes` ................................................................. 44
- `nonresponse` .............................................................. 45
- `oldsvyquantile` ......................................................... 46
- `open.DBIsvydesign` ..................................................... 49
- `paley` ................................................................. 50
- `pchisqsum` ............................................................... 52
- `poisson_sampling` ...................................................... 53
- `postStratify` ............................................................ 54
- `psrsq` ................................................................. 56
- `rake` ................................................................. 57
- `regTermTest` ............................................................ 59
- `salamander` .............................................................. 61
- `scd` ................................................................. 62
- `SE` ................................................................. 63
- `smoothArea` ............................................................. 64
- `smoothUnit` ............................................................. 66
- `stratsample` ............................................................. 68
- `subset.survey.design` .................................................. 69
- `surveyoptions` .......................................................... 70
- `surveysummary` ........................................................ 71
- `svrepdesign` ........................................................... 75
- `svrVar` ................................................................. 78
- `svy.varcoef` ........................................................... 79
- `svyby` ................................................................. 80
- `svycdf` ................................................................. 83
- `svyciprop` ............................................................... 85
- `svycontrast` ............................................................ 87
- `svycoplot` .............................................................. 89
- `svycoxph` ............................................................... 90
anova.svyglm

Model comparison for glms.

Description

A method for the anova function, for use on svyglm and svycoxph objects. With a single model argument it produces a sequential anova table, with two arguments it compares the two models.
Usage

## S3 method for class 'svyglm'
anova(object, object2 = NULL, test = c("F", "Chisq"),
       method = c("LRT", "Wald"), tolerance = 1e-05, ..., force = FALSE)

## S3 method for class 'svycoxph'
anova(object, object2 = NULL, test = c("F", "Chisq"),
       method = c("LRT", "Wald"), tolerance = 1e-05, ..., force = FALSE)

## S3 method for class 'svyglm'
AIC(object, ..., k = 2, null_has_intercept = TRUE)

## S3 method for class 'svyglm'
BIC(object, ..., maximal)

## S3 method for class 'svyglm'
extractAIC(fit, scale, k = 2, ..., null_has_intercept = TRUE)

## S3 method for class 'svyglm'
extractAIC(fit, scale, k = 2, ..., null_has_intercept = TRUE)

Arguments

object, fit A svyglm or svycoxph object.
object2 Optionally, another svyglm or svycoxph object.
test Use (linear combination of) F or chi-squared distributions for p-values. F is usually preferable.
method Use weighted deviance difference (LRT) or Wald tests to compare models
tolerance For models that are not symbolically nested, the tolerance for deciding that a term is common to the models.
... For AIC and BIC, optionally more svyglm objects
scale not used
null_has_intercept Does the null model for AIC have an intercept or not? Must be FALSE if any of the models are intercept-only.
force Force the tests to be done by explicit projection even if the models are symbolically nested (eg, for debugging)
maximal A svyglm model that object (and ... if supplied) are nested in.
k Multiplier for effective df in AIC. Usually 2. There is no choice of k that will give BIC

Details

The reference distribution for the LRT depends on the misspecification effects for the parameters being tested (Rao and Scott, 1984). If the models are symbolically nested, so that the relevant parameters can be identified just by manipulating the model formulas, anova is equivalent to regTermTest. If the models are nested but not symbolically nested, more computation using the design matrices is needed to determine the projection matrix on to the parameters being tested. In the examples below, model1 and model2 are symbolically nested in model0 because model0 can be obtained just by deleting terms from the formulas. On the other hand, model2 is nested in model1.
but not symbolically nested: knowing that the model is nested requires knowing what design matrix columns are produced by stype and as.numeric(stype). Other typical examples of models that are nested but not symbolically nested are linear and spline models for a continuous covariate, or models with categorical versions of a variable at different resolutions (e.g., smoking yes/no or smoking never/former/current).

A saddlepoint approximation is used for the LRT with numerator df greater than 1.

AIC is defined using the Rao-Scott approximation to the weighted loglikelihood (Lumley and Scott, 2015). It replaces the usual penalty term \( p \), which is the null expectation of the log likelihood ratio, by the trace of the generalised design effect matrix, which is the expectation under complex sampling. For computational reasons everything is scaled so the weights sum to the sample size.

BIC is a BIC for the (approximate) multivariate Gaussian models on regression coefficients from the maximal model implied by each submodel (i.e., the models that say some coefficients in the maximal model are zero) (Lumley and Scott, 2015). It corresponds to comparing the models with a Wald test and replacing the sample size in the penalty by an effective sample size. For computational reasons, the models must not only be nested, the names of the coefficients must match.

extractAIC for a model with a Gaussian link uses the actual AIC based on maximum likelihood estimation of the variance parameter as well as the regression parameters.

Value

Object of class seqanova.svyglm if one model is given, otherwise of class regTermTest or regTermTestLRT

Note

At the moment, AIC works only for models including an intercept.

References


See Also

regTermTest, pchisqsum

Examples

data(api)
dclus2<-svydesign(id=~dnum+snum, weights=~pw, data=apiclus2)

model0<-svyglm(I(sch.wide=="Yes")~ell+meals+mobility, design=dclus2, family=quasibinomial())
model1<-svyglm(I(sch.wide=="Yes")~ell+meals+mobility+as.numeric(stype),
    design=dclus2, family=quasibinomial())
model2<-svyglm(I(sch.wide=="Yes")~ell+meals+mobility+stype, design=dclus2, family=quasibinomial())
api

Student performance in California schools

Description

The Academic Performance Index is computed for all California schools based on standardised testing of students. The data sets contain information for all schools with at least 100 students and for various probability samples of the data.

Usage

data(api)

Format

The full population data in apipop are a data frame with 6194 observations on the following 37 variables.
cds  Unique identifier
stype  Elementary/Middle/High School
name  School name (15 characters)
name  School name (40 characters)
num  School number
dname  District name
dnum  District number
cname  County name
cnum  County number
flag  reason for missing data
ptest  percentage of students tested
api00  API in 2000
api99  API in 1999
target  target for change in API
growth  Change in API
sch.wide  Met school-wide growth target?
comp.imp  Met Comparable Improvement target
both  Met both targets
awards  Eligible for awards program
meal  Percentage of students eligible for subsidized meals
ell  ‘English Language Learners’ (percent)
yr.rnd  Year-round school
mobility  percentage of students for whom this is the first year at the school
acs.k3  average class size years K-3
acs.46  average class size years 4-6
acs.core  Number of core academic courses
pct.resp  percent where parental education level is known
not.hsg  percent parents not high-school graduates
hsg  percent parents who are high-school graduates
some.col  percent parents with some college
col.grad  percent parents with college degree
grad.sch  percent parents with postgraduate education
avg.ed  average parental education level
full  percent fully qualified teachers
emer  percent teachers with emergency qualifications
enroll  number of students enrolled
api.stu  number of students tested.

The other data sets contain additional variables pw for sampling weights and fpc to compute finite population corrections to variance.
Details

apipop is the entire population, apisrs is a simple random sample, apiclus1 is a cluster sample of school districts, apistrat is a sample stratified by stype, and apiclus2 is a two-stage cluster sample of schools within districts. The sampling weights in apiclus1 are incorrect (the weight should be 757/15) but are as obtained from UCLA.

Source

Data were obtained from the survey sampling help pages of UCLA Academic Technology Services; these pages are no longer on line.

References

The API program has been discontinued at the end of 2018, and the archive page at the California Department of Education is now gone. The Wikipedia article has links to past material at the Internet Archive. https://en.wikipedia.org/wiki/Academic_Performance_Index_(California_public_schools)

Examples

library(survey)

data(api)

mean(apipop$api00)

sum(apipop$enroll, na.rm=TRUE)

# stratified sample

dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)

summary(dstrat)

svymean(~api00, dstrat)

svytotal(~enroll, dstrat, na.rm=TRUE)

# one-stage cluster sample

dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

summary(dclus1)

svymean(~api00, dclus1)

svytotal(~enroll, dclus1, na.rm=TRUE)

# two-stage cluster sample

dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)

summary(dclus2)

svymean(~api00, dclus2)

svytotal(~enroll, dclus2, na.rm=TRUE)

# two-stage 'with replacement'

dclus2wr<-svydesign(id=~dnum+snum, weights=~pw, data=apiclus2)

summary(dclus2wr)

svymean(~api00, dclus2wr)

svytotal(~enroll, dclus2wr, na.rm=TRUE)

# convert to replicate weights
rclus1<-as.svrepdesign(dclus1)
summary(rclus1)
svymean(~api00, rclus1)
svytotal(~enroll, rclus1, na.rm=TRUE)

# post-stratify on school type
pop.types<-xtabs(~stype, data=apiipop)
rclus1p<-postStratify(rclus1, ~stype, pop.types)
dclus1p<-postStratify(dclus1, ~stype, pop.types)
summary(dclus1p)
summary(rclus1p)
svymean(~api00, dclus1p)
svytotal(~enroll, dclus1p, na.rm=TRUE)
svymean(~api00, rclus1p)
svytotal(~enroll, rclus1p, na.rm=TRUE)

as.fpc

Package sample and population size data

Description
This function creates an object to store the number of clusters sampled within each stratum (at each stage of multistage sampling) and the number of clusters available in the population. It is called by svydesign, not directly by the user.

Usage
as.fpc(df, strata, ids, pps=FALSE)

Arguments
df A data frame or matrix with population size information
strata A data frame giving strata at each stage
ids A data frame giving cluster ids at each stage
pps if TRUE, fpc information may vary within a stratum and must be specified as a proportion rather than a population sizes

Details
The population size information may be specified as the number of clusters in the population or as the proportion of clusters sampled.

Value
An object of class survey_fpc
as.svrepdesign

See Also
svydesign, svyrecvar

---

Convert a survey design to use replicate weights

**Description**

Creates a replicate-weights survey design object from a traditional strata/cluster survey design object. JK1 and JKn are jackknife methods, BRR is Balanced Repeated Replicates and Fay is Fay’s modification of this, bootstrap is Canty and Davison’s bootstrap, subbootstrap is Rao and Wu’s \((n-1)\) bootstrap, and mrbbootstrap is Preston’s multistage rescaled bootstrap. With a svyimputationList object, the same replicate weights will be used for each imputation if the sampling weights are all the same and separate.replicates=FALSE.

**Usage**

```r
as.svrepdesign(design, ...)
```

## Default S3 method:

```r
as.svrepdesign(design, type = c("auto", "JK1", "JKn", "BRR", "bootstrap", "subbootstrap", "mrbbootstrap", "Fay"),
               fay.rho = 0, fpc = NULL, fpctype = NULL, ..., compress = TRUE,
               mse = getOption("survey.replicates.mse"))
```

## S3 method for class 'svyimputationList'

```r
as.svrepdesign(design, type = c("auto", "JK1", "JKn", "BRR", "bootstrap", "subbootstrap", "mrbbootstrap", "Fay"),
               fay.rho = 0, fpc = NULL, fpctype = NULL, separate.replicates = FALSE, ..., compress = TRUE,
               mse = getOption("survey.replicates.mse"))
```

**Arguments**

- `design`: Object of class survey.design or svyimputationList. Must not have been post-stratified/raked/calibrated in R
- `type`: Type of replicate weights. "auto" uses JKn for stratified, JK1 for unstratified designs
- `fay.rho`: Tuning parameter for Fay’s variance method
- `fpc`, `fpctype`,... Passed to jk1weights, jknweights, brrweights, bootweights, subbootweights, or mrbweights.
- `separate.replicates`: Compute replicate weights separately for each design (useful for the bootstrap types, which are not deterministic
- `compress`: Use a compressed representation of the replicate weights matrix.
- `mse`: if TRUE, compute variances from sums of squares around the point estimate, rather than the mean of the replicates
Value

Object of class svyrep.design.

References


See Also

brrweights, svydesign, svrepdesign, bootweights, subbootweights, mrbweights

Examples

data(scd)
scddes<svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA, nest=TRUE, fpc=rep(5,6))
scdnofpc<svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA, nest=TRUE)
# convert to BRR replicate weights
scd2brr <- as.svrepdesign(scdnofpc, type="BRR")
scd2fay <- as.svrepdesign(scdnofpc, type="Fay", fay.rho=0.3)
# convert to JKn weights
scd2jkn <- as.svrepdesign(scdnofpc, type="JKn")
# convert to JKn weights with finite population correction
scd2jknf <- as.svrepdesign(scddes, type="JKn")
## with user-supplied hadamard matrix
scd2brr1 <- as.svrepdesign(scdnofpc, type="BRR", hadamard.matrix=paley(11))
svyratio(~alive, ~arrests, design=scd2brr)
svyratio(~alive, ~arrests, design=scd2brr1)
svyratio(~alive, ~arrests, design=scd2fay)
svyratio(~alive, ~arrests, design=scd2jkn)
svyratio(~alive, ~arrests, design=scd2jknf)
data(api)
## one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
## convert to JK1 jackknife
rclus1<-as.svrepdesign(dclus1)
## convert to bootstrap
bclus1<-as.svrepdesign(dclus1,type="bootstrap", replicates=100)
svymean(~api00, dclus1)
svytotal(~enroll, dclus1)
svymean(~api00, rclus1)
svytotal(~enroll, rclus1)
svymean(~api00, bclus1)
svytotal(~enroll, bclus1)

dclus2<-svydesign(id = ~dnum + snum, fpc = ~fpc1 + fpc2, data = apiclus2)
mrbclus2<-as.svrepdesign(dclus2, type="mrb",replicates=100)
svytotal(~api00+stype, dclus2)
svytotal(~api00+stype, mrbclus2)

---

as.svydesign2  Update to the new survey design format

Description

The structure of survey design objects changed in version 2.9, to allow standard errors based on multistage sampling. as.svydesign converts an object to the new structure and .svycheck warns if an object does not have the new structure.

You can set options(survey.want.obsolete=TRUE) to suppress the warnings produced by .svycheck and options(survey.ultimate.cluster=TRUE) to always compute variances based on just the first stage of sampling.

Usage

as.svydesign2(object)
.svycheck(object)

Arguments

object produced by svydesign

Value

Object of class survey.design2

See Also

svydesign, svyrecvar
Description

Draws a barplot or dotplot based on results from a survey analysis. The default barplot method already works for results from svytable.

Usage

```r
## S3 method for class 'svystat'
barplot(height, ...)
## S3 method for class 'svrepstat'
barplot(height, ...)
## S3 method for class 'svby'
barplot(height, beside=TRUE, ...)

## S3 method for class 'svystat'
dotchart(x,...,pch=19)
## S3 method for class 'svrepstat'
dotchart(x,...,pch=19)
## S3 method for class 'svby'
dotchart(x,...,pch=19)
```

Arguments

- `height`, `x`: Analysis result
- `beside`: Grouped, rather than stacked, bars
- `...`: Arguments to barplot or dotchart
- `pch`: Overrides the default in dotchart.default

Examples

```r
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

a<-svymean(~stype, dclus1)
barplot(a)
barplot(a, names.arg=c("Elementary","High","Middle"), col="purple",
       main="Proportions of school level")

b<-svyby(~enroll+api.stu, ~stype, dclus1, svymean)
barplot(b,beside=TRUE,legend=TRUE)
dotchart(b)
```
bootweights

Compute survey bootstrap weights

Description

Bootstrap weights for infinite populations (‘with replacement’ sampling) are created by sampling with replacement from the PSUs in each stratum. subbootweights() samples n-1 PSUs from the n available (Rao and Wu), bootweights samples n (Canty and Davison).

For multistage designs or those with large sampling fractions, mrbweights implements Preston’s multistage rescaled bootstrap. The multistage rescaled bootstrap is still useful for single-stage designs with small sampling fractions, where it reduces to a half-sample replicate method.

Usage

bootweights(strata, psu, replicates = 50, fpc = NULL, fpctype = c("population", "fraction", "correction"), compress = TRUE)

subbootweights(strata, psu, replicates = 50, compress = TRUE)
mrbweights(clusters, stratas, fpcs, replicates = 50, multicore = getOption("survey.multicore"))

Arguments

- strata: Identifier for sampling strata (top level only)
- stratas: data frame of strata for all stages of sampling
- psu: Identifier for primary sampling units
- clusters: data frame of identifiers for sampling units at each stage
- replicates: Number of bootstrap replicates
- fpc: Finite population correction (top level only)
- fpctype: Is fpc the population size, sampling fraction, or 1-sampling fraction?
- fpcs: survey_fpc object with population and sample size at each stage
- compress: Should the replicate weights be compressed?
- multicore: Use the multicore package to generate the replicates in parallel

Value

A set of replicate weights

warning

With multicore=TRUE the resampling procedure does not use the current random seed, so the results cannot be exactly reproduced even by using set.seed()
Note

These bootstraps are strictly appropriate only when the first stage of sampling is a simple or stratified random sample of PSUs with or without replacement, and not (eg) for PPS sampling. The functions will not enforce simple random sampling, so they can be used (approximately) for data that have had non-response corrections and other weight adjustments. It is preferable to apply these adjustments after creating the bootstrap replicate weights, but that may not be possible with public-use data.

References


See Also

as.svrepdesign

brrweights

Compute replicate weights

Description

Compute replicate weights from a survey design. These functions are usually called from as.svrepdesign rather than directly by the user.

Usage

brrweights(strata, psu, match = NULL,
small = c("fail","split","merge"),
large = c("split","merge","fail"),
fay.rho=0, only.weights=FALSE,
compress=TRUE, hadamard.matrix=NULL)

jk1weights(psu,fpc=NULL,
fpctype=c("population","fraction","correction"),
compress=TRUE)

jknweights(strata,psu, fpc=NULL,
fpctype=c("population","fraction","correction"),
compress=TRUE,
lonely.psu=getOption("survey.lonely.psu"))
Arguments

strata     Stratum identifiers
psu        PSU (cluster) identifier
match      Optional variable to use in matching.
small      How to handle strata with only one PSU
large      How to handle strata with more than two PSUs
fpc        Optional population (stratum) size or finite population correction
fpctype    How fpc is coded.
fay.rho    Parameter for Fay's extended BRR method
only.weights If TRUE return only the matrix of replicate weights
compress   If TRUE, store the replicate weights in compressed form
hadamard.matrix Optional user-supplied Hadamard matrix for brrweights
lonely.psu Handling of non-certainty single-PSU strata

Details

JK1 and JKn are jackknife schemes for unstratified and stratified designs respectively. The finite population correction may be specified as a single number, a vector with one entry per stratum, or a vector with one entry per observation (constant within strata). When fpc is a vector with one entry per stratum it may not have names that differ from the stratum identifiers (it may have no names, in which case it must be in the same order as unique(strata)). To specify population stratum sizes use fpctype="population", to specify sampling fractions use fpctype="fraction" and to specify the correction directly use fpctype="correction"

The only reason not to use compress=TRUE is that it is new and there is a greater possibility of bugs. It reduces the number of rows of the replicate weights matrix from the number of observations to the number of PSUs.

In BRR variance estimation each stratum is split in two to give half-samples. Balanced replicated weights are needed, where observations in two different strata end up in the same half stratum as often as in different half-strata. BRR, strictly speaking, is defined only when each stratum has exactly two PSUs. A stratum with one PSU can be merged with another such stratum, or can be split to appear in both half samples with half weight. The latter approach is appropriate for a PSU that was deterministically sampled.

A stratum with more than two PSUs can be split into multiple smaller strata each with two PSUs or the PSUs can be merged to give two superclusters within the stratum.

When merging small strata or grouping PSUs in large strata the match variable is used to sort PSUs before merging, to give approximate matching on this variable.

If you want more control than this you should probably construct your own weights using the Hadamard matrices produced by hadamard.
Value

For `brrweights` with `only.weights=FALSE` a list with elements

- `weights` two-column matrix indicating the weight for each half-stratum in one particular set of split samples
- `wstrata` New stratum variable incorporating merged or split strata
- `strata` Original strata for distinct PSUs
- `psu` Distinct PSUs
- `nnpairs` Dimension of Hadamard matrix used in BRR construction
- `sampler` function returning replicate weights
- `compress` Indicates whether the `sampler` returns per PSU or per observation weights

For `jk1weights` and `jknweights` a data frame of replicate weights and the `scale` and `rscale` arguments to `svrVar`.

References

Levy and Lemeshow "Sampling of Populations". Wiley.

Shao and Tu "The Jackknife and Bootstrap". Springer.

See Also

`hadamard`, `as.svrepdesign`, `svrVar`, `surveyoptions`

Examples

data(scd)
scdnofpcc<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA, nest=TRUE)

## convert to BRR replicate weights
scd2brrr <- as.svrepdesign(scdnofpcc, type="BRR")
svymean(~alive, scd2brr)
svyratio(~alive, ~arrests, scd2brr)

## with user-supplied hadamard matrix
scd2brrl <- as.svrepdesign(scdnofpcc, type="BRR", hadamard.matrix=paley(11))
svymean(~alive, scd2brrl)
svyratio(~alive, ~arrests, scd2brrl)
Calibration, generalized raking, or GREG estimators generalise post-stratification and raking by calibrating a sample to the marginal totals of variables in a linear regression model. This function reweights the survey design and adds additional information that is used by `svyrecvar` to reduce the estimated standard errors.

### Usage

```r
calibrate(design,...)
## S3 method for class 'survey.design2'
# calibrate(design, formula, population,
#   aggregate.stage=NULL, stage=0, variance=NULL,
#   bounds=c(-Inf,Inf), calfun=c("linear","raking","logit"),
#   maxit=50, epsilon=1e-7, verbose=FALSE, force=FALSE, trim=NULL,
#   bounds.const=FALSE, sparse=FALSE,...)
## S3 method for class 'svyrep.design'
# calibrate(design, formula, population, compress=NA,
#   aggregate.index=NULL, variance=NULL, bounds=c(-Inf,Inf),
#   calfun=c("linear","raking","logit"),
#   maxit=50, epsilon=1e-7, verbose=FALSE, force=FALSE, trim=NULL,
#   bounds.const=FALSE, sparse=FALSE,...)
## S3 method for class 'twophase'
# calibrate(design, phase=2, formula, population,
#   calfun=c("linear","raking","logit","rrz"),...)
# grake(mm,ww,calfun,eta=rep(0,NCOL(mm)),population,epsilon,
#   verbose,maxit,variance=NULL)
# cal_names(formula,design,...)
```

### Arguments

- `design`: Survey design object
- `formula`: Model formula for calibration model, or list of formulas for each margin
- `population`: Vectors of population column totals for the model matrix in the calibration model, or list of such vectors for each cluster, or list of tables for each margin. Required except for two-phase designs
- `compress`: compress the resulting replicate weights if TRUE or if NA and weights were previously compressed
- `stage`: See Details below
- `variance`: Coefficients for variance in calibration model (heteroskedasticity parameters) (see Details below)
aggregate.stage
An integer. If not \texttt{NULL}, make calibration weights constant within sampling units at this stage.

aggregate.index
A vector or one-sided formula. If not \texttt{NULL}, make calibration weights constant within levels of this variable

bounds
Bounds for the calibration weights, optional except for \texttt{calfun="logit"}

bounds.const
Should be \texttt{TRUE} if bounds have been specified as constant values rather than multiplicative values

trim
Weights outside this range will be trimmed to these bounds.

... Options for other methods

calfun
Calibration function: see below

maxit
Number of iterations

epsilon
Tolerance in matching population total. Either a single number or a vector of the same length as \texttt{population}

verbose
Print lots of uninteresting information

force
Return an answer even if the specified accuracy was not achieved

phase
Phase of a two-phase design to calibrate (only \texttt{phase=2} currently implemented.)

mm
Model matrix

ww
Vector of weights

eta
Starting values for iteration

sparse
Use sparse matrices for faster computation

Details
The \texttt{formula} argument specifies a model matrix, and the \texttt{population} argument is the population column sums of this matrix. The function \texttt{cal_names} shows what the column names of this model matrix will be.

For the important special case where the calibration totals are (possibly overlapping) marginal tables of factor variables, as in classical raking, the \texttt{formula} and \texttt{population} arguments may be lists in the same format as the input to \texttt{rake}.

If the \texttt{population} argument has a names attribute it will be checked against the names produced by \texttt{model.matrix(formula)} and reordered if necessary. This protects against situations where the (locale-dependent) ordering of factor levels is not what you expected.

Numerical instabilities may result if the sampling weights in the design object are wrong by multiple orders of magnitude. The code now attempts to rescale the weights first, but it is better for the user to ensure that the scale is reasonable.

The \texttt{calibrate} function implements linear, bounded linear, raking, bounded raking, and logit calibration functions. All except unbounded linear calibration use the Newton-Raphson algorithm described by Deville et al (1993). This algorithm is exposed for other uses in the \texttt{grake} function. Unbounded linear calibration uses an algorithm that is less sensitive to collinearity. The calibration function may be specified as a string naming one of the three built-in functions or as an object of class \texttt{calfun}, allowing user-defined functions. See \texttt{make.calfun} for details.
The bounds argument can be specified as global upper and lower bounds e.g. \( \text{bounds}=c(0.5, 2) \) or as a list with lower and upper vectors e.g. \( \text{bounds}=\text{list}(\text{lower}, \text{upper}) \). This allows for individual boundary constraints for each unit. The lower and upper vectors must be the same length as the input data. The bounds can be specified as multiplicative values or constant values. If constant, \( \text{bounds}.\text{const} \) must be set to \text{TRUE}.

Calibration with bounds, or on highly collinear data, may fail. If \text{force}=\text{TRUE} the approximately calibrated design object will still be returned (useful for examining why it failed). A failure in calibrating a set of replicate weights when the sampling weights were successfully calibrated will give only a warning, not an error.

When calibration to the desired set of bounds is not possible, another option is to trim weights. To do this set \text{bounds} to a looser set of bounds for which calibration is achievable and set \text{trim} to the tighter bounds. Weights outside the bounds will be trimmed to the bounds, and the excess weight distributed over other observations in proportion to their sampling weight (and so this may put some other observations slightly over the trimming bounds). The projection matrix used in computing standard errors is based on the feasible bounds specified by the \text{bounds} argument. See also \text{trimWeights}, which trims the final weights in a design object rather than the calibration adjustments.

For two-phase designs \text{calfun}="rrz" estimates the sampling probabilities using logistic regression as described by Robins et al (1994). \text{estWeights} will do the same thing.

Calibration may result in observations within the last-stage sampling units having unequal weight even though they necessarily are sampled together. Specifying \text{aggregate.stage} ensures that the calibration weight adjustments are constant within sampling units at the specified stage; if the original sampling weights were equal the final weights will also be equal. The algorithm is as described by Vanderhoeft (2001, section III.D). Specifying \text{aggregate.index} does the same thing for replicate weight designs; a warning will be given if the original weights are not constant within levels of \text{aggregate.index}.

In a model with two-stage sampling, population totals may be available for the PSUs actually sampled, but not for the whole population. In this situation, calibrating within each PSU reduces with second-stage contribution to variance. This generalizes to multistage sampling. The stage argument specifies which stage of sampling the totals refer to. Stage 0 is full population totals, stage 1 is totals for PSUs, and so on. The default, stage=NULL is interpreted as stage 0 when a single population vector is supplied and stage 1 when a list is supplied. Calibrating to PSU totals will fail (with a message about an exactly singular matrix) for PSUs that have fewer observations than the number of calibration variables.

The variance in the calibration model may depend on covariates. If variance=NULL the calibration model has constant variance. If variance is not NULL it specifies a linear combination of the columns of the model matrix and the calibration variance is proportional to that linear combination. Alternatively variance can be specified as a vector of values the same length as the input data specifying a heteroskedasticity parameter for each unit.

The design matrix specified by formula (after any aggregation) must be of full rank, with one exception. If the population total for a column is zero and all the observations are zero the column will be ignored. This allows the use of factors where the population happens to have no observations at some level.

In a two-phase design, population may be omitted when phase=2, to specify calibration to the phase-one sample. If the two-phase design object was constructed using the more memory-efficient
method="approx" argument to `twophase`, calibration of the first phase of sampling to the population is not supported.

Value

A survey design object.

References


Lumley T, Shaw PA, Dai JY (2011) "Connections between survey calibration estimators and semiparametric models for incomplete data" International Statistical Review. 79:200-220. (with discussion 79:221-232)


See Also

`postStratify`, `rake` for other ways to use auxiliary information

`twophase` and vignette("epi") for an example of calibration in two-phase designs


`make.calfun` for user-defined calibration distances.

`trimWeights` to trim final weights rather than calibration adjustments.

Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

```
cal_names(~stype, dclus1)
```

```
pop.totals<-c(~(Intercept)~6194, stypeH=755, stypeM=1018)
```

```
## For a single factor variable this is equivalent to
## postStratify

(dclus1g<calibrate(dclus1, ~stype, pop.totals))
```
```r
svymean(~api00, dclus1g)
svytotal(~enroll, dclus1g)
svytotal(~stype, dclus1g)

## Make weights constant within school district
dclus1agg<-calibrate(dclus1, ~stype, pop.totals, aggregate=1)
svymean(~api00, dclus1agg)
svytotal(~enroll, dclus1agg)
svytotal(~stype, dclus1agg)

## Now add sch.wide
cal_names(~stype+sch.wide, dclus1)
dclus1g2 <- calibrate(dclus1, ~stype+sch.wide, c(pop.totals, sch.wideYes=5122))
svymean(~api00, dclus1g2)
svytotal(~enroll, dclus1g2)
svytotal(~stype, dclus1g2)

## Finally, calibrate on 1999 API and school type
cal_names(~stype+api99, dclus1)
dclus1g3 <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069))
svymean(~api00, dclus1g3)
svytotal(~enroll, dclus1g3)
svytotal(~stype, dclus1g3)

## Same syntax with replicate weights
rclus1<-as.svrepdesign(dclus1)
(rclus1g3 <- calibrate(rclus1, ~stype+api99, c(pop.totals, api99=3914069)))
svymean(~api00, rclus1g3)
svytotal(~enroll, rclus1g3)
svytotal(~stype, rclus1g3)

(rclus1agg3 <- calibrate(rclus1, ~stype+api99, c(pop.totals, api99=3914069), aggregate.index=~dnum))
svymean(~api00, rclus1agg3)
svytotal(~enroll, rclus1agg3)
svytotal(~stype, rclus1agg3)

###
## Bounded weights
range(weights(dclus1g3)/weights(dclus1))
dclus1g3b <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069), bounds=c(0.6,1.6))
range(weights(dclus1g3b)/weights(dclus1))
svymean(~api00, dclus1g3b)
svytotal(~enroll, dclus1g3b)
```
calibrate

svytotal(~stype, dclus1g3b)

## Individual boundary constraints as constant values
# the first weight will be bounded at 40, the rest free to move
bnds <- list(
  lower = rep(-Inf, nrow(apiclus1)),
  upper = c(40, rep(Inf, nrow(apiclus1)-1)))
head(weights(dclus1g3))
dclus1g3b1 <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069),
  bounds=bnds, bounds.const=TRUE)
head(weights(dclus1g3b1))
svytotal(~api.stu, dclus1g3b1)

## trimming
dclus1tr <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069),
  bounds=c(0.5,2), trim=c(2/3,3/2))
svymean(~api00+api99+enroll, dclus1tr)
svytotal(~stype,dclus1tr)
range(weights(dclus1tr)/weights(dclus1))
rclus1tr <- calibrate(rclus1, ~stype+api99, c(pop.totals, api99=3914069),
  bounds=c(0.5,2), trim=c(2/3,3/2))
svymean(~api00+api99+enroll, rclus1tr)
svytotal(~stype,rclus1tr)

## Input in the same format as rake() for classical raking
pop.table <- xtabs(~stype+sch.wide, apipop)
pop.table2 <- xtabs(~stype+comp.imp, apipop)
dclus1r<-rake(dclus1, list(~stype+sch.wide, ~stype+comp.imp),
  list(pop.table, pop.table2))
gclus1r<-calibrate(dclus1, formula=list(~stype+sch.wide, ~stype+comp.imp),
  population=list(pop.table, pop.table2),calfun="raking")
svymean(~api00+stype, dclus1r)
svymean(~api00+stype, gclus1r)

## generalised raking
dclus1g3c <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069),
  calfun="raking")
range(weights(dclus1g3c)/weights(dclus1))
(dclus1g3d <- calibrate(dclus1, ~stype+api99, c(pop.totals, api99=3914069),
  calfun=cal.logit, bounds=c(0.5,2.5)))
range(weights(dclus1g3d)/weights(dclus1))

## Ratio estimators are calibration estimators
dstrat<-svydesign(id=~1,strata=stype, weights=~pw, data=apistrat, fpc=~fpc)
svytotal(~api.stu,dstrat)
common<-svyratio(~api.stu, ~enroll, dstrat, separate=FALSE)
predict(common, total=3811472)
compressWeights

Compress replicate weight matrix

Description

Many replicate weight matrices have redundant rows, such as when weights are the same for all observations in a PSU. This function produces a compressed form. Methods for `as.matrix` and `as.vector` extract and expand the weights.

Usage

```r
compressWeights(rw, ...)  # S3 method for class 'svyrep.design'
compressWeights(rw, ...)  # S3 method for class 'repweights_compressed'
as.matrix(x, ...)  # S3 method for class 'repweights_compressed'
as.vector(x, ...)  # S3 method for class 'repweights_compressed'
```

Arguments

- `rw` A set of replicate weights or a `svyrep.design` object
- `x` A compressed set of replicate weights
- `...` For future expansion

Value

An object of class `repweights_compressed` or a `svyrep.design` object with `repweights` element of class `repweights_compressed`

See Also

`jknweights`, `as.svrepdesign`
Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rcclus1c<-as.svrepdesign(dclus1,compress=TRUE)
rcclus1<-as.svrepdesign(dclus1,compress=FALSE)

confint.svyglm

Confidence intervals for regression parameters

Description

Computes confidence intervals for regression parameters in svyglm objects. The default is a Wald-type confidence interval, adding and subtracting a multiple of the standard error. The method="likelihood" is an interval based on inverting the Rao-Scott likelihood ratio test. That is, it is an interval where the working model deviance is lower than the threshold for the Rao-Scott test at the specified level.

Usage

## S3 method for class 'svyglm'
confint(object, parm, level = 0.95, method = c("Wald", "likelihood"), ddf = NULL, ...)

Arguments

object svyglm object
parm numeric or character vector indicating which parameters to construct intervals for.
level desired coverage
method See description above
ddf Denominator degrees of freedom for "likelihood" method, to use a t distribution rather than normal. If NULL, use object$df.residual
... for future expansion

Value

A matrix of confidence intervals

References


See Also

confint
Examples

data(api)
dclus2<-svydesign(id=~dnum+snum, fpc=~fpcl+fpc2, data=apiclus2)

m<-svyglm(I(comp.imp=="Yes")~stype*emer+ell, design=dclus2, family=quasibinomial)
confint(m)
confint(m, method="like", ddf=NULL, parm=c("ell","emer"))

crowd

Household crowding

Description

A tiny dataset from the VPLX manual.

Usage

data(crowd)

Format

A data frame with 6 observations on the following 5 variables.

- **rooms**: Number of rooms in the house
- **person**: Number of people in the household
- **weight**: Sampling weight
- **cluster**: Cluster number
- **stratum**: Stratum number

Source

Manual for VPLX, Census Bureau.

Examples

data(crowd)

## Example 1-1
i1.1<-as.svrepdesign(svydesign(id=~cluster, weight=~weight,data=crowd))
i1.1<-update(i1.1, room.ratio=rooms/person,
overcrowded=factor(person>rooms))
svymean(~rooms+person+room.ratio,i1.1)
svytotal(~rooms+person+room.ratio,i1.1)
svymean(~rooms+person+room.ratio,subset(i1.1,overcrowded==TRUE))
svytotal(~rooms+person+room.ratio,subset(i1.1,overcrowded==TRUE))

## Example 1-2
i1.2 <- as.svrepdesign(svymean(~rooms+person,i1.2)
svytotal(~rooms+person,i1.2)

**Description**

`dimnames` returns variable names and row names for the data variables in a design object and `dim` returns dimensions. For multiple imputation designs there is a third dimension giving the number of imputations. For database-backed designs the second dimension includes variables defined by `update`. The first dimension excludes observations with zero weight.

**Usage**

```r
## S3 method for class 'survey.design'
dim(x)
## S3 method for class 'svyimputationList'
dim(x)
## S3 method for class 'survey.design'
dimnames(x)
## S3 method for class 'DBIsvydesign'
dimnames(x)
## S3 method for class 'svyimputationList'
dimnames(x)
```

**Arguments**

- `x` Design object

**Value**

A vector of numbers for `dim`, a list of vectors of strings for `dimnames`.

**See Also**

`update.DBIsvydesign`, `with.svyimputationList`

**Examples**

```r
data(api)
dclus1 <- svydesign(ids=~dnum,weights=~pw, data=apiclus1,fpc=~fpc)
dim(dclus1)
dimnames(dclus1)
colnames(dclus1)
```
US 2004 presidential election data at state or county level

Description

A sample of voting data from US states or counties (depending on data availability), sampled with probability proportional to number of votes. The sample was drawn using Tille’s splitting method, implemented in the "sampling" package.

Usage

data(election)

Format

election is a data frame with 4600 observations on the following 8 variables.

County  A factor specifying the state or country
TotPrecincts  Number of precincts in the state or county
PrecinctsReporting  Number of precincts supplying data
Bush  Votes for George W. Bush
Kerry  Votes for John Kerry
Nader  Votes for Ralph Nader
votes  Total votes for those three candidates
p  Sampling probability, proportional to votes

election_pps is a sample of 40 counties or states taken with probability proportional to the number of votes. It includes the additional column wt with the sampling weights.
election_insample indicates which rows of election were sampled.
election_jointprob are the pairwise sampling probabilities and election_jointHR are approximate pairwise sampling probabilities using the Hartley-Rao approximation.

Source

.

Examples

data(election)

## high positive correlation between totals
plot(Bush*Kerry, data=election, log="xy")

## high negative correlation between proportions
plot(I(Bush/votes)-I(Kerry/votes), data=election)

## Variances without replacement
## Horvitz-Thompson type
estweights

Estimated weights for missing data

Description

Creates or adjusts a two-phase survey design object using a logistic regression model for second-phase sampling probability. This function should be particularly useful in reweighting to account for missing data.

Usage

estWeights(data, formula, ...)  
## S3 method for class 'twophase'
estWeights(data, formula = NULL, working.model = NULL, ...)  
## S3 method for class 'data.frame'
estWeights(data, formula = NULL, working.model = NULL, subset = NULL, strata = NULL, ...)

Arguments

data   twophase design object or data frame
formula  Predictors for estimating weights
working.model  Model fitted to complete (ie phase 1) data
subset  Subset of data frame with complete data (ie phase 1). If NULL use all complete cases
strata  Stratification (if any) of phase 2 sampling
...  for future expansion
Details

If `data` is a data frame, `estWeights` first creates a two-phase design object. The `strata` argument is used only to compute finite population corrections, the same variables must be included in `formula` to compute stratified sampling probabilities.

With a two-phase design object, `estWeights` estimates the sampling probabilities using logistic regression as described by Robins et al (1994) and adds information to the object to enable correct sandwich standard errors to be computed.

An alternative to specifying `formula` is to specify `working.model`. The estimating functions from this model will be used as predictors of the sampling probabilities, which will increase efficiency to the extent that the working model and the model of interest estimate the same parameters (Kulich & Lin 2004).

The effect on a two-phase design object is very similar to `calibrate`, and is identical when `formula` specifies a saturated model.

Value

A two-phase survey design object.

References


Lumley T, Shaw PA, Dai JY (2011) "Connections between survey calibration estimators and semi-parametric models for incomplete data" International Statistical Review. 79:200-220. (with discussion 79:221-232)

See Also

`postStratify`, `calibrate`, `twophase`

Examples

data(airquality)

## ignoring missingness, using model-based standard error
summary(lm(log(Ozone)~Temp+Wind, data=airquality))

## Without covariates to predict missingness we get
## same point estimates, but different (sandwich) standard errors
daq<-estWeights(airquality, formula=~1,subset=~I(!is.na(Ozone)))
summary(svyglm(log(Ozone)~Temp+Wind,design=daq))

## Reweighting based on weather, month
d2aq<-estWeights(airquality, formula=~Temp+Wind+Month,
Description

The fpc data frame has 8 rows and 6 columns. It is artificial data to illustrate survey sampling estimators.

Usage

data(fpc)

Format

This data frame contains the following columns:

stratid  Stratum ids
psuid   Sampling unit ids
weight  Sampling weights
nh      number sampled per stratum
Nh      population size per stratum
x       data

Source

https://www.stata-press.com/data/r7/fpc.dta

Examples

data(fpc)
fpc

withoutfpc<-svydesign(weights=~weight, ids=~psuid, strata=~stratid, variables=~x, 
data=fpc, nest=TRUE)
withoutfpc
svymean(~x, withoutfpc)

withfpc<-svydesign(weights=~weight, ids=~psuid, strata=~stratid, 
fpc=~Nh, variables=~x, data=fpc, nest=TRUE)
withfpc
svymean(~x, withfpc)
## Other equivalent forms

```r
withfpc<-svydesign(prob=~I(1/weight), ids=~psuid, strata=~stratid,
                      fpc=~Nh, variables=~x, data=fpc, nest=TRUE)
svymean(~x, withfpc)

withfpc<-svydesign(weights=~weight, ids=~psuid, strata=~stratid,
                      fpc=~I(nh/Nh), variables=~x, data=fpc, nest=TRUE)
svymean(~x, withfpc)

withfpc<-svydesign(weights=~weight, ids=~interaction(stratid,psuid),
                      strata=~stratid, fpc=~I(nh/Nh), variables=~x, data=fpc)
svymean(~x, withfpc)

withfpc<-svydesign(ids=~psuid, strata=~stratid, fpc=~Nh,
                      variables=~x, data=fpc, nest=TRUE)
svymean(~x, withfpc)

withfpc<-svydesign(ids=~psuid, strata=~stratid, fpc=~I(nh/Nh),
                      variables=~x, data=fpc, nest=TRUE)
svymean(~x, withfpc)
```

---

**ftable.svystat**

*Lay out tables of survey statistics*

### Description

Reformat the output of survey computations to a table.

### Usage

```r
## S3 method for class 'svystat'
ftable(x, rownames,...)
## S3 method for class 'svrepstat'
ftable(x, rownames,...)
## S3 method for class 'svyby'
ftable(x,...)
```

### Arguments

- `x` Output of functions such as svymean, svrepmean, svyby
hadamard

rownames List of vectors of strings giving dimension names for the resulting table (see examples)

... Arguments for future expansion

Value

An object of class "ftable"

See Also

ftable

Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

a<-svymean(~interaction(stype,comp.imp), design=dclus1)
b<-ftable(a, rownames=list(stype=c("E","H","M"),comp.imp=c("No","Yes")))
b

a<-svymean(~interaction(stype,comp.imp), design=dclus1, deff=TRUE)
b<-ftable(a, rownames=list(stype=c("E","H","M"),comp.imp=c("No","Yes")))
round(100*b,1)

rclus1<-as.svrepdesign(dclus1)
a<-svytotal(~interaction(stype,comp.imp), design=rclus1)
b<-ftable(a, rownames=list(stype=c("E","H","M"),comp.imp=c("No","Yes")))
b
round(b)

a<-svyby(~api99 + api00, ~stype + sch.wide, rclus1, svymean, keep.var=TRUE)
ftable(a)
print(ftable(a),digits=2)

b<-svyby(~api99 + api00, ~stype + sch.wide, rclus1, svymean, keep.var=TRUE, deff=TRUE)
print(ftable(b),digits=2)

d<-svyby(~api99 + api00, ~stype + sch.wide, rclus1, svymean, keep.var=TRUE, vartype=c("se","cvpct"))
round(ftable(d),1)

hadamard

Hadamard matrices

Description

Returns a Hadamard matrix of dimension larger than the argument.
hadamard

Usage

hadamard(n)

Arguments

n  lower bound for size

Details

For most \( n \) the matrix comes from paley. The \( 36 \times 36 \) matrix is from Plackett and Burman (1946) and the \( 28 \times 28 \) is from Sloane’s library of Hadamard matrices.

Matrices of dimension every multiple of 4 are thought to exist, but this function doesn’t know about all of them, so it will sometimes return matrices that are larger than necessary. The excess is at most 4 for \( n < 180 \) and at most 5\% for \( n > 100 \).

Value

A Hadamard matrix

Note

Strictly speaking, a Hadamard matrix has entries +1 and -1 rather than 1 and 0, so \( 2 \times \text{hadamard}(n) - 1 \) is a Hadamard matrix

References

Sloane NJA. A Library of Hadamard Matrices http://neilsloane.com/hadamard/


See Also

brrweights, paley

Examples

par(mfrow=c(2,2))
## Sylvester-type
image(hadamard(63),main=quote("Sylvester: "*64==2^6))
## Paley-type
image(hadamard(59),main=quote("Paley: "*60==59+1))
## from NJ Sloane's library
image(hadamard(27),main=quote("Stored: "*28))
## For n=90 we get 96 rather than the minimum possible size, 92.
image(hadamard(90),main=quote("Constructed: "*96==2^3%*%(11+1))


Sample of obstetric hospitals

Description

The hospital data frame has 15 rows and 5 columns.

Usage

data(hospital)

Format

This data frame contains the following columns:

- **hospno** Hospital id
- **oblevel** level of obstetric care
- **weighta** Weights, as given by the original reference
- **tothosp** total hospitalisations
- **births** births
- **weightats** Weights, as given in the source

Source

Previously at http://www.ats.ucla.edu/stat/books/sop/hospsamp.dta

References


Examples

data(hospital)

hospdes<-svydesign(strata=~oblevel, id=~hospno, weights=~weighta, fpc=~tothosp, data=hospital)
hosprep<-as.svrepdesign(hospdes)

svytotal(~births, design=hospdes)
svytotal(~births, design=hosprep)
Wrappers for specifying PPS designs

Description

The Horvitz-Thompson estimator and the Hartley-Rao approximation require information in addition to the sampling probabilities for sampled individuals. These functions allow this information to be supplied.

Usage

```r
HR(psum=NULL, strata = NULL)
ppsmat(jointprob, tolerance = 1e-04)
ppscov(probcov, weighted=FALSE)
```

Arguments

- `psum`: The sum of squared sampling probabilities for the population, divided by the sample size, as a single number or as a vector for stratified sampling
- `strata`: Stratum labels, of the same length as `psum`, if `psum` is a vector
- `jointprob`: Matrix of pairwise sampling probabilities for the sampled individuals
- `tolerance`: Tolerance for deciding that the covariance of sampling indicators is zero
- `probcov`: Covariance of the sampling indicators (often written 'Delta'), or weighted covariance if `weighted=TRUE`
- `weighted`: If TRUE, the `probcov` argument is the covariance divided by pairwise sampling probabilities

Value

An object of class `HR`, `ppsmat`, `ppsdelta`, or `ppsdcheck` suitable for supplying as the `pps` argument to `svydesign`.

See Also

- `election` for examples of PPS designs

Examples

```r
HR(0.1)
```
make.calfun

Description

Create calibration metric for use in calibrate. The function $F$ is the link function described in section 2 of Deville et al. To create a new calibration metric, specify $F - 1$ and its derivative. The package provides cal.linear, cal.raking, cal.logit, which are standard, and cal.sinh from the CALMAR2 macro, for which $F$ is the derivative of the inverse hyperbolic sine.

Usage

make.calfun(Fm1, dF, name)

Arguments

Fm1 Function $F - 1$ taking a vector $u$ and a vector of length 2, bounds.
dF Derivative of Fm1 wrt u: arguments u and bounds
name Character string to use as name

Value

An object of class “calfun”

References


See Also

calibrate

Examples

str(cal.linear)
cal.linear$Fm1
cal.linear$dF

hellinger <- make.calfun(Fm1=function(u, bounds) ((1-u/2)^-2)-1,
dF= function(u, bounds) (1-u/2)^-3 ,
name="hellinger distance")

hellinger
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
marginpred

Standardised predictions (predictive margins) for regression models.

Description

Reweights the design (using calibrate) so that the adjustment variables are uncorrelated with the variables in the model, and then performs predictions by calling predict. When the adjustment model is saturated this is equivalent to direct standardization on the adjustment variables.

The svycoxph and svykmlist methods return survival curves.

Usage

marginpred(model, adjustfor, predictat, ...)  
## S3 method for class 'svycoxph'
marginpred(model, adjustfor, predictat, se=FALSE, ...)  
## S3 method for class 'svykmlist'
marginpred(model, adjustfor, predictat, se=FALSE, ...)  
## S3 method for class 'svyglm'
marginpred(model, adjustfor, predictat, ...)

Arguments

model A regression model object of a class that has a marginpred method
adjustfor Model formula specifying adjustment variables, which must be in the design object of the model
predictat A data frame giving values of the variables in model to predict at
se Estimate standard errors for the survival curve (uses a lot of memory if the sample size is large)
... Extra arguments, passed to the predict method for model

See Also

svypredmeans for the method of Graubard and Korn implemented in SUDAAN.
calibrate
predict.svycoxph
Examples

```r
## generate data with apparent group effect from confounding
tset.seed(42)
df<-data.frame(x=rnorm(100))
df$time<-rexp(100)*exp(df$x-1)
df$status<-1
df$group<-(df$x+rnorm(100))>0
des<-svydesign(id=-1, data=df)
newdf<-data.frame(group=c(FALSE, TRUE), x=c(0, 0))

## Cox model
m0<-svycoxph(Surv(time, status)~group, design=des)
m1<-svycoxph(Surv(time, status)~group+x, design=des)

## conditional predictions, unadjusted and adjusted
cpred0<-predict(m0, type="curve", newdata=newdf, se=TRUE)
cpred1<-predict(m1, type="curve", newdata=newdf, se=TRUE)

## adjusted marginal prediction
mpred<-marginpred(m0, adjustfor=~x, predictat=newdf, se=TRUE)

plot(cpred0)
lines(cpred0[[1]], col="red")
lines(cpred0[[2]], col="red")
lines(mpred[[1]], col="blue")
lines(mpred[[2]], col="blue")

## Kaplan--Meier
s2<-svykm(Surv(time, status>0)~group, design=des)
p2<-marginpred(s2, adjustfor=~x, predictat=newdf, se=TRUE)
plot(s2)
lines(p2[[1]], col="green")
lines(p2[[2]], col="green")

## logistic regression
logisticm <- svyglm(group~time, family=quasibinomial, design=des)
newdf$time<-c(0.1, 0.8)
logisticpred <- marginpred(logisticm, adjustfor=~x, predictat=newdf)
```

Description

The MU284 population comes from Sarndal et al, and the complete data are available from Statlib. These data are a two-stage sample from the population, analyzed on page 143 of the book.

Usage

data(mu284)
Format

A data frame with 15 observations on the following 5 variables.

- id1: identifier for PSU
- n1: number of PSUs in population
- id2: identifier for second-stage unit
- y1: variable to be analysed
- n2: number of second-stage units in this PSU

Source

(downloaded from StatLib, which is no longer active)

Examples

data(mu284)
(dmu284<-svydesign(id=~id1+id2,fpc=~n1+n2, data=mu284))
(ytotal<-svytotal(~y1, dmu284))
vcov(ytotal)

myco

Association between leprosy and BCG vaccination

Description

These data are in a paper by JNK Rao and colleagues, on score tests for complex survey data. External information (not further specified) suggests the functional form for the Age variable.

Usage

data("myco")

Format

A data frame with 516 observations on the following 6 variables.

- Age: Age in years at the midpoint of six age strata
- Scar: Presence of a BCG vaccination scar
- n: Sampled number of cases (and thus controls) in the age stratum
- Ncontrol: Number of non-cases in the population
- wt: Sampling weight
- leprosy: case status 0/1
Details

The data are a simulated stratified case-control study drawn from a population study conducted in a region of Malawi (Clayton and Hills, 1993, Table 18.1). The goal was to examine whether BCG vaccination against tuberculosis protects against leprosy (the causative agents are both species of _Mycobacterium_). Rao et al have a typographical error: the number of non-cases in the population in the 25-30 age stratum is given as 4981 but 5981 matches both the computational output and the data as given by Clayton and Hills.

Source


Examples

data(myco)
dmyco<-svydesign(id=~1, strata=~interaction(Age,leprosy),weights=~wt,data=myco)

m_full<-svyglm(leprosy~I((Age+7.5)^-2)+Scar, family=quasibinomial, design=dmyco)
m_age<-svyglm(leprosy~I((Age+7.5)^-2), family=quasibinomial, design=dmyco)
anova(m_full,m_age)

## unweighted model does not match
m_full
glm(leprosy~I((Age+7.5)^-2)+Scar, family=binomial, data=myco)

---

newsvyquantile  
Quantiles under complex sampling.

Description

Estimates quantiles and confidence intervals for them. This function was completely re-written for version 4.1 of the survey package, and has a wider range of ways to define the quantile. See the vignette for a list of them.

Usage

ewsvyquantile(x, design, quantiles, ...)  
## S3 method for class 'survey.design'
svyquantile(x, design, quantiles, alpha = 0.05,  
interval.type = c("mean", "beta","xlogit", "asin","score"),  
na.rm = FALSE, ci=TRUE, se = ci,  
qrule=c("math","school","shahvaish","hf1","hf2","hf3",  
"hf4","hf5","hf6","hf7","hf8","hf9"),  
df = NULL, ...)

svyquantile(x, design, quantiles, alpha = 0.05,
    interval.type = c("mean", "beta", "xlogit", "asin", "quantile"),
    na.rm = FALSE, ci = TRUE, se = ci,
    qrule = c("math", "school", "shahvaish", "hf1", "hf2", "hf3",
              "hf4", "hf5", "hf6", "hf7", "hf8", "hf9"),
    df = NULL, return.replicates = FALSE,...)

Arguments

x A one-sided formula describing variables to be used
design Design object
quantiles Numeric vector specifying which quantiles are requested
alpha Specified confidence interval coverage
interval.type See Details below
na.rm Remove missing values?
se Return an estimated confidence interval and standard error?
qrule Rule for defining the quantiles: either a character string specifying one of the
        built-in rules, or a function
df Degrees of freedom for confidence interval estimation: NULL specifies degf(design)
return.replicates Return replicate estimates of the quantile (only for interval.type="quantile")
... For future expansion

Details

The \( p \)th quantile is defined as the value where the estimated cumulative distribution function is equal
to \( p \). As with quantiles in unweighted data, this definition only pins down the quantile to an interval
between two observations, and a rule is needed to interpolate. The default is the mathematical
definition, the lower end of the quantile interval; `qrule="school"` uses the midpoint of the quantile
interval; `"hf1"` to `"hf9"` are weighted analogues of type=1 to 9 in `quantile`. See the vignette
"Quantile rules" for details and for how to write your own.

By default, confidence intervals are estimated using Woodruff’s (1952) method, which involves
computing the quantile, estimating a confidence interval for the proportion of observations be-
low the quantile, and then transforming that interval using the estimated CDF. In that context, the
interval.type argument specifies how the confidence interval for the proportion is computed,
matching `svyciprop`. In contrast to `oldsvyquantile`, NaN is returned if a confidence interval end-
point on the probability scale falls outside \([0,1]\).

There are two exceptions. For `svydesign` objects, `interval.type="score"` asks for the Francisco
& Fuller confidence interval based on inverting a score test. According to Dorfmann & Valliant,
this interval has inferior performance to the "beta" and "logit" intervals; it is provided for compatibility.

For replicate-weight designs, `interval.type="quantile"` ask for an interval based directly on the
replicates of the quantile. This interval is not valid for jackknife-type replicates, though it should
perform well for bootstrap-type replicates, BRR, and SDR.
The df argument specifies degrees of freedom for a t-distribution approximation to distributions of means. The default is the design degrees of freedom. Specify df=Inf to use a Normal distribution (eg, for compatibility).

When the standard error is requested, it is estimated by dividing the confidence interval length by the number of standard errors in a t confidence interval with the specified alpha. For example, with alpha=0.05 and df=Inf the standard error is estimated as the confidence interval length divided by 2*1.96.

Value

An object of class "newsvyquantile", except that with a replicate-weights design and interval.type="quantile" and return.replicates=TRUE it's an object of class "svrepstat"

References


Woodruff RS (1952) Confidence intervals for medians and other position measures. JASA 57, 622-627.

See Also

vignette("qrule", package = "survey") oldsvyquantile quantile

Examples

data(api)
## population
quantile(api$api00,c(.25,.5,.75))

## one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rclus1<-as.svrepdesign(dclus1)
bclus1<-as.svrepdesign(dclus1,type="boot")

svyquantile(~api00, dclus1, c(.25,.5,.75))
svyquantile(~api00, dclus1, c(.25,.5,.75),interval.type="beta")

svyquantile(~api00, rclus1, c(.25,.5,.75))
svyquantile(~api00, rclus1, c(.25,.5,.75),interval.type="quantile")
svyquantile(~api00, bclus1, c(.25,.5,.75),interval.type="quantile")
svyquantile(~api00+ell, dclus1, c(.25,.5,.75), qrule="math")
svyquantile(~api00+ell, dclus1, c(.25,.5,.75), qrule="school")
svyquantile(~api00+ell, dclus1, c(.25,.5,.75), qrule="hf8")

---

**nhanes**  
*Cholesterol data from a US survey*

---

**Description**

Data extracted from NHANES 2009-2010 on high cholesterol.

**Usage**

```r
data(nhanes)
```

**Format**

A data frame with 8591 observations on the following 7 variables.

- **SDMVPSU**  Primary sampling units
- **SDMVSTRA**  Sampling strata
- **WTMEC2YR**  Sampling weights
- **HI_CHOL**  Numeric vector: 1 for total cholesterol over 240mg/dl, 0 under 240mg/dl
- **race** 1=Hispanic, 2=non-Hispanic white, 3=non-Hispanic black, 4=other
- **agecat**  Age group: (0,19] (19,39] (39,59] (59,Inf]
- **RIAGENDR**  Gender: 1=male, 2=female

**Source**


**Examples**

```r
data(nhanes)
design <- svydesign(id=~SDMVPSU, strata=~SDMVSTRA, weights=~WTMEC2YR, nest=TRUE, data=nhanes)  
design
```
nonresponse

Experimental: Construct non-response weights

Description

Functions to simplify the construction of non-response weights by combining strata with small numbers or large weights.

Usage

nonresponse(sample.weights, sample.counts, population)

sparseCells(object, count=0, totalweight=Inf, nrweight=1.5)

neighbours(index, object)

joinCells(object, a, ...)

## S3 method for class 'nonresponse'

weights(object, ...)

Arguments

- sample.weights: table of sampling weight by stratifying variables
- sample.counts: table of sample counts by stratifying variables
- population: table of population size by stratifying variables
- object: object of class "nonresponse"
- count: Cells with fewer sampled units than this are "sparse"
- nrweight: Cells with higher non-response weight than this are "sparse"
- totalweight: Cells with average sampling weight times non-response weight higher than this are "sparse"
- index: Number of a cell whose neighbours are to be found
- a, ...: Cells to join

Details

When a stratified survey is conducted with imperfect response it is desirable to rescale the sampling weights to reflect the nonresponse. If some strata have small sample size, high non-response, or already had high sampling weights it may be desirable to get less variable non-response weights by averaging non-response across strata. Suitable strata to collapse may be similar on the stratifying variables and/or on the level of non-response.

nonresponse() combines stratified tables of population size, sample size, and sample weight into an object. sparseCells identifies cells that may need combining. neighbours describes the cells adjacent to a specified cell, and joinCells collapses the specified cells. When the collapsing is complete, use weights() to extract the nonresponse weights.

Value

nonresponse and joinCells return objects of class "nonresponse", neighbours and sparseCells return objects of class "nonresponseSubset"
Examples

data(api)
## pretend the sampling was stratified on three variables
poptable<-xtabs(~sch.wide+comp.imp+stype,data=apipop)
sample.count<-xtabs(~sch.wide+comp.imp+stype,data=apiclus1)
sample.weight<-xtabs(pw~sch.wide+comp.imp+stype, data=apiclus1)

## create a nonresponse object
nr<-nonresponse(sample.weight,sample.count, poptable)

## sparse cells
sparseCells(nr)

## Look at neighbours
neighbours(3,nr)
neighbours(11,nr)

## Collapse some contiguous cells
nr1<-joinCells(nr,3,5,7)

## sparse cells now
sparseCells(nr1)
nr2<-joinCells(nr1,3,11,8)

## one relatively sparse cell
sparseCells(nr2)
## but nothing suitable to join it to
neighbours(3,nr2)

## extract the weights
weights(nr2)

---

oldsvyquantile

 Deprecated implementation of quantiles

Description

Compute quantiles for data from complex surveys. oldsvyquantile is the version of the function from before version 4.1 of the package, available for backwards compatibility. See svyquantile for the current version.

Usage

## S3 method for class 'survey.design'
oldsvyquantile(x, design, quantiles, alpha=0.05,
ci=FALSE, method = "linear", f = 1,
interval.type=c("Wald","score","betaWald"), na.rm=FALSE,se=ci,
oldsvyquantile

```
# S3 method for class 'svyrep.design'
oldsvyquantile(x, design, quantiles,
    method = "linear", interval.type = c("probability", "quantile"), f = 1,
    return.replicates = FALSE, ties = c("discrete", "rounded"), na.rm = FALSE,
    alpha = 0.05, df = NULL,...)
```

**Arguments**

- `x` A formula, vector or matrix
- `design` survey.design or svyrep.design object
- `quantiles` Quantiles to estimate
- `method` see `approxfun`
- `f` see `approxfun`
- `ci` Compute a confidence interval? (relatively slow; needed for `svyby`)
- `se` Compute standard errors from the confidence interval length?
- `alpha` Level for confidence interval
- `interval.type` See Details below
- `ties` See Details below
- `df` Degrees of freedom for a t-distribution. Inf requests a Normal distribution, NULL uses `degeff`. Not relevant for type="betaWald"
- `return.replicates` Return the replicate means?
- `na.rm` Remove NAs?
- `...` arguments for future expansion

**Details**

The definition of the CDF and thus of the quantiles is ambiguous in the presence of ties. With `ties="discrete"` the data are treated as genuinely discrete, so the CDF has vertical steps at tied observations. With `ties="rounded"` all the weights for tied observations are summed and the CDF interpolates linearly between distinct observed values, and so is a continuous function. Combining `interval.type="betaWald"` and `ties="discrete"` is (close to) the proposal of Shah and Vaish(2006) used in some versions of SUDAAN.

Interval estimation for quantiles is complicated, because the influence function is not continuous. Linearisation cannot be used directly, and computing the variance of replicates is valid only for some designs (eg BRR, but not jackknife). The `interval.type` option controls how the intervals are computed.

For survey.design objects the default is `interval.type="Wald"`. A 95% Wald confidence interval is constructed for the proportion below the estimated quantile. The inverse of the estimated CDF is used to map this to a confidence interval for the quantile. This is the method of Woodruff (1952). For "betaWald" the same procedure is used, but the confidence interval for the proportion is computed using the exact binomial cdf with an effective sample size proposed by Korn & Graubard (1998).
If `interval.type="score"` we use a method described by Binder (1991) and due originally to Francisco and Fuller (1986), which corresponds to inverting a robust score test. At the upper and lower limits of the confidence interval, a test of the null hypothesis that the cumulative distribution function is equal to the target quantile just rejects. This was the default before version 2.9. It is much slower than "Wald", and Dorfman & Valliant (1993) suggest it is not any more accurate.

Standard errors are computed from these confidence intervals by dividing the confidence interval length by \(2\times qnorm(\alpha/2)\).

For replicate-weight designs, ordinary replication-based standard errors are valid for BRR and Fay’s method, and for some bootstrap-based designs, but not for jackknife-based designs. `interval.type="quantile"` gives these replication-based standard errors. The default, `interval.type="probability"` computes confidence on the probability scale and then transforms back to quantiles, the equivalent of `interval.type="Wald"` for `survey.design` objects (with `alpha=0.05`).

There is a `confint` method for `svyquantile` objects; it simply extracts the pre-computed confidence interval.

**Value**

returns a list whose first component is the quantiles and second component is the confidence intervals. For replicate weight designs, returns an object of class `svyrepstat`.

**Author(s)**

Thomas Lumley

**References**


Woodruff RS (1952) Confidence intervals for medians and other position measures. *JASA* 57, 622-627.

**See Also**

`svykm` for quantiles of survival curves

`svyciprop` for confidence intervals on proportions.
Examples

```r
data(api)
## population
quantile(api00$api00, c(.25, .5, .75))

## one-stage cluster sample
dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
oldsvyquantile(~api00, dclus1, c(.25, .5, .75), ci = TRUE)
oldsvyquantile(~api00, dclus1, c(.25, .5, .75), ci = TRUE, interval.type = "betaWald")
oldsvyquantile(~api00, dclus1, c(.25, .5, .75), ci = TRUE, df = NULL)

dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
(qapi <- oldsvyquantile(~api00, dclus1, c(.25, .5, .75), ci = TRUE, interval.type = "score")
SE(qapi)

# stratified sample
dstrat <- svydesign(id = 1, strata = ~stype, weights = ~pw, data = apistrat, fpc = ~fpc)
oldsvyquantile(~api00, dstrat, c(.25, .5, .75), ci = TRUE)

# stratified sample, replicate weights
# interval="probability" is necessary for jackknife weights
rstrat <- as.svrepdesign(dstrat)
oldsvyquantile(~api00, rstrat, c(.25, .5, .75), interval.type = "probability")

# BRR method
data(scd)
repweights <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0, 1), c(0, 1, 0, 1, 0, 1),
                       c(0, 1, 0, 1, 1, 0))
scdrep <- svrepdesign(data = scd, type = "BRR", repweights = repweights)
oldsvyquantile(~arrests + alive, design = scdrep, quantile = 0.5, interval.type = "quantile")
oldsvyquantile(~arrests + alive, design = scdrep, quantile = 0.5, interval.type = "quantile", df = NULL)
```

---

**open.DBIsvydesign**

*Open and close DBI connections*

**Description**

A database-backed survey design object contains a connection to a database. This connection will be broken if the object is saved and reloaded, and the connection should ideally be closed with `close` before quitting R (although it doesn’t matter for SQLite connections). The connection can be reopened with `open`.

**Usage**

```r
## S3 method for class 'DBIsvydesign'
```
open(con, ...)  
## S3 method for class 'DBIsvydesign'  
close(con, ...)

**Arguments**

- **con** Object of class DBIsvydesign
- **...** Other options, to be passed to dbConnect or dbDisconnect.

**Value**

The same survey design object with the connection opened or closed.

**See Also**

- svydesign
- DBI package

**Examples**

```r
## Not run:  
library(RSQLite)  
dclus1<-svydesign(id=~dnum, weights=~pw, fpc=~fpc,  
data="apiclus1",dbtype="SQLite",  
dbname=system.file("api.db",package="survey"))

dclus1  
close(dclus1)  
dclus1  
try(svymean(~api00, dclus1))

dclus1<-open(dclus1)  
open(dclus1)  
svymean(~api00, dclus1)

## End(Not run)
```

---

**paley**

**Paley-type Hadamard matrices**

**Description**

Computes a Hadamard matrix of dimension \((p + 1) \times 2^k\), where \(p\) is a prime, and \(p+1\) is a multiple of 4, using the Paley construction. Used by hadamard.

**Usage**

```r
paley(n, nmax = 2 * n, prime=NULL, check=!is.null(prime))

is.hadamard(H, style=c("0/1","+-"), full.orthogonal.balance=TRUE)
```
Arguments

- **n**: Minimum size for matrix
- **nmax**: Maximum size for matrix. Ignored if prime is specified.
- **prime**: Optional. A prime at least as large as n, such that prime+1 is divisible by 4.
- **check**: Check that the resulting matrix is of Hadamard type
- **H**: Matrix
- **style**: "0/1" for a matrix of 0s and 1s, "+-" for a matrix of ±1.
- **full.orthogonal.balance**: Require full orthogonal balance?

Details

The Paley construction gives a Hadamard matrix of order p+1 if p is prime and p+1 is a multiple of 4. This is then expanded to order \((p + 1) \times 2^k\) using the Sylvester construction.

paley knows primes up to 7919. The user can specify a prime with the prime argument, in which case a matrix of order \(p + 1\) is constructed.

If check=TRUE the code uses is.hadamard to check that the resulting matrix really is of Hadamard type, in the same way as in the example below. As this test takes \(n^3\) time it is preferable to just be sure that prime really is prime.

A Hadamard matrix including a row of 1s gives BRR designs where the average of the replicates for a linear statistic is exactly the full sample estimate. This property is called full orthogonal balance.

Value

For paley, a matrix of zeros and ones, or NULL if no matrix smaller than nmax can be found.

For is.hadamard, TRUE if H is a Hadamard matrix.

References


See Also

hadamard

Examples

```r
M <- paley(11)

is.hadamard(M)
## internals of is.hadamard(M)
H <- 2*M - 1
## HH^T is diagonal for any Hadamard matrix
H%*%t(H)
```
**pchisqsum**

*Distribution of quadratic forms*

**Description**

The distribution of a quadratic form in p standard Normal variables is a linear combination of p chi-squared distributions with 1 df. When there is uncertainty about the variance, a reasonable model for the distribution is a linear combination of F distributions with the same denominator.

**Usage**

```r
pchisqsum(x, df, a, lower.tail = TRUE, 
          method = c("satterthwaite", "integration","saddlepoint"))
pFsum(x, df, a, ddf=Inf,lower.tail = TRUE, 
       method = c("saddlepoint","integration","satterthwaite"), ...)
```

**Arguments**

- **x**  
  Observed values

- **df**  
  Vector of degrees of freedom

- **a**  
  Vector of coefficients

- **ddf**  
  Denominator degrees of freedom

- **lower.tail**  
  lower or upper tail?

- **method**  
  See Details below

- **...**  
  arguments to pchisqsum

**Details**

The "satterthwaite" method uses Satterthwaite's approximation, and this is also used as a fallback for the other methods. The accuracy is usually good, but is more variable depending on a than the other methods and is anticonservative in the right tail (eg for upper tail probabilities less than $10^{-5}$). The Satterthwaite approximation requires all $a>0$.

"integration" requires the CompQuadForm package. For pchisqsum it uses Farebrother's algorithm if all $a>0$. For pFsum or when some $a<0$ it inverts the characteristic function using the algorithm of Davies (1980). These algorithms are highly accurate for the lower tail probability, but they obtain the upper tail probability by subtraction from 1 and so fail completely when the upper tail probability is comparable to machine epsilon or smaller.

If the CompQuadForm package is not present, a warning is given and the saddlepoint approximation is used.

"saddlepoint" uses Kuonen's saddlepoint approximation. This is moderately accurate even very far out in the upper tail or with some $a=0$ and does not require any additional packages. The relative error in the right tail is uniformly bounded for all $x$ and decreases as $p$ increases. This method is implemented in pure R and so is slower than the "integration" method.

The distribution in pFsum is standardised so that a likelihood ratio test can use the same x value as in pchisqsum. That is, the linear combination of chi-squareds is multiplied by $ddf$ and then divided by an independent chi-squared with $ddf$ degrees of freedom.
Description

Specify a design where units are sampled independently from the population, with known probabilities. This design is often used theoretically, but is rarely used in practice because the sample size is variable. This function calls `ppscov` to specify a sparse sampling covariance matrix.
Usage

poisson_sampling(p)

Arguments

p Vector of sampling probabilities

Value

Object of class ppsdcheck

See Also

ppscov, svydesign

Examples

data(api)
apipop$prob<-with(apiipop, 200*api00/sum(api00))
insample<-as.logical(rbinom(nrow(apiipop),1,apiipop$prob))
apipois<-apiipop[insample,]
despois<svydesign(id=~1, prob=~prob, pps=poisson_sampling(apiois$prob), data=apiois)
svytotal(~api00, despois)

## SE formula
sqrt(sum( (apiois$api00*weights(despois))^2*(1-apiois$prob)))

---

postStratify Post-stratify a survey

Description

Post-stratification adjusts the sampling and replicate weights so that the joint distribution of a set of post-stratifying variables matches the known population joint distribution. Use rake when the full joint distribution is not available.

Usage

postStratify.design, strata, population, partial = FALSE, ...)
## S3 method for class 'svyrep.design'
postStratify.design, strata, population, partial = FALSE, compress=NULL,...)
## S3 method for class 'survey.design'
postStratify.design, strata, population, partial = FALSE, ...)
**postStratify**

- **Arguments**
  - `design`: A survey design with replicate weights
  - `strata`: A formula or data frame of post-stratifying variables, which must not contain missing values.
  - `population`: A table, `xtabs` or `data.frame` with population frequencies
  - `partial`: If `TRUE`, ignore population strata not present in the sample
  - `compress`: Attempt to compress the replicate weight matrix? When `NULL` will attempt to compress if the original weight matrix was compressed
  - `...`: arguments for future expansion

- **Details**
  The population totals can be specified as a table with the strata variables in the margins, or as a data frame where one column lists frequencies and the other columns list the unique combinations of strata variables (the format produced by `as.data.frame` acting on a `table` object). A table must have named dimnames to indicate the variable names.

Compressing the replicate weights will take time and may even increase memory use if there is actually little redundancy in the weight matrix (in particular if the post-stratification variables have many values and cut across PSUs).

If a `svydesign` object is to be converted to a replication design the post-stratification should be performed after conversion.

The variance estimate for replication designs follows the same procedure as Valliant (1993) described for estimating totals. Rao et al (2002) describe this procedure for estimating functions (and also the GREG or g-calibration procedure, see `calibrate`)

- **Value**
  A new survey design object.

- **Note**
  If the sampling weights are already post-stratified there will be no change in point estimates after `postStratify` but the standard error estimates will decrease to correctly reflect the post-stratification.

- **References**

- **See Also**
  - `rake`, `calibrate` for other things to do with auxiliary information
  - `compressWeights` for information on compressing weights
Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rclus1<-as.svrepdesign(dclus1)

svymean(~api00, rclus1)
svytotal(~enroll, rclus1)

# post-stratify on school type
pop.types <- data.frame(stype=c("E","H","M"), Freq=c(4421,755,1018))
#or: pop.types <- xtabs(~stype, data=apipop)
#or: pop.types <- table(stype=apipop$stype)

rclus1p<-postStratify(rclus1, ~stype, pop.types)
syntax(rclus1p)
svymean(~api00, rclus1p)
svytotal(~enroll, rclus1p)

## and for svydesign objects
dclus1p<-postStratify(dclus1, ~stype, pop.types)
syntax(dclus1p)
svymean(~api00, dclus1p)
svytotal(~enroll, dclus1p)

---

psrsq

Pseudo-Rsquareds

Description

Compute the Nagelkerke and Cox–Snell pseudo-r-squared statistics, primarily for logistic regression. A generic function with methods for glm and svyglm. The method for svyglm objects uses the design-based estimators described by Lumley (2017).

Usage

psrsq(object, method = c("Cox-Snell", "Nagelkerke"), ...)

Arguments

object

A regression model (glm or svyglm)

method

Which statistic to compute

... For future expansion

Value

Numeric value
Rake uses iterative post-stratification to match marginal distributions of a survey sample to known population margins.

Usage

rake(design, sample.margins, population.margins, control = list(maxit = 10, epsilon = 1, verbose=FALSE), compress=NULL)

Arguments

- **design**: A survey object
- **sample.margins**: list of formulas or data frames describing sample margins, which must not contain missing values
- **population.margins**: list of tables or data frames describing corresponding population margins
- **control**: maxit controls the number of iterations. Convergence is declared if the maximum change in a table entry is less than epsilon. If epsilon<1 it is taken to be a fraction of the total sampling weight.
- **compress**: If design has replicate weights, attempt to compress the new replicate weight matrix? When NULL, will attempt to compress if the original weight matrix was compressed
Details

The sample margins should be in a format suitable for postStratify.

Raking (aka iterative proportional fitting) is known to converge for any table without zeros, and for any table with zeros for which there is a joint distribution with the given margins and the same pattern of zeros. The ‘margins’ need not be one-dimensional.

The algorithm works by repeated calls to postStratify (iterative proportional fitting), which is efficient for large multiway tables. For small tables calibrate will be faster, and also allows raking to population totals for continuous variables, and raking with bounded weights.

Value

A raked survey design.

See Also

postStratify, compressWeights

calibrate for other ways to use auxiliary information.

Examples

data(api)
dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rclus1 <- as.svrepdesign(dclus1)

svymean(~api00, rclus1)
svytotal(~enroll, rclus1)

## population marginal totals for each stratum
pop.types <- data.frame(stype=c("E","H","M"), Freq=c(4421,755,1018))
pop.schwide <- data.frame(sch.wide=c("No","Yes"), Freq=c(1072,5122))

rclus1r <- rake(rclus1, list(~stype,~sch.wide), list(pop.types, pop.schwide))

svymean(~api00, rclus1r)
svytotal(~enroll, rclus1r)

## marginal totals correspond to population
xtabs(~stype, apipop)
svytable(~stype, rclus1r, round=TRUE)
xtabs(~sch.wide, apipop)
svytable(~sch.wide, rclus1r, round=TRUE)

## joint totals don't correspond
xtabs(~stype+sch.wide, apipop)
svytable(~stype+sch.wide, rclus1r, round=TRUE)

## Do it for a design without replicate weights
dclus1r<-rake(dclus1, list(~stype,~sch.wide), list(pop.types, pop.schwide))

svymean(~api00, dclus1r)
svytotal(~enroll, dclus1r)

## compare to raking with calibrate()

dclus1gr <- calibrate(dclus1, ~stype+sch.wide, pop=c(6194, 755, 1018, 5122),
calfun="raking")

svymean(~stype+api00, dclus1r)
svymean(~stype+api00, dclus1gr)

## compare to joint post-stratification
## (only possible if joint population table is known)
##
pop.table <- xtabs(~stype+sch.wide, apipop)

rclus1ps <- postStratify(rclus1, ~stype+sch.wide, pop.table)
svytable(~stype+sch.wide, rclus1ps, round=TRUE)

svymean(~api00, rclus1ps)
svytotal(~enroll, rclus1ps)

## Example of raking with partial joint distributions

pop.imp <- data.frame(comp.imp=c("No","Yes"), Freq=c(1712,4482))
dclus1r2 <- rake(dclus1, list(~stype+sch.wide, ~comp.imp),
                  list(pop.table, pop.imp))

svymean(~api00, dclus1r2)

## compare to calibrate() syntax with tables

dclus1r2 <- calibrate(dclus1, formula=list(~stype+sch.wide, ~comp.imp),
                      population=list(pop.table, pop.imp), calfun="raking")

svymean(~api00, dclus1r2)

---

**regTermTest**

**Wald test for a term in a regression model**

**Description**

Provides Wald test and working Wald and working likelihood ratio (Rao-Scott) test of the hypothesis that all coefficients associated with a particular regression term are zero (or have some other specified values). Particularly useful as a substitute for `anova` when not fitting by maximum likelihood.

**Usage**

```r
regTermTest(model, test.terms, null=NULL, df=NULL,
            method=c("Wald","WorkingWald","LRT"),
            lrt.approximation="saddlepoint")
```

**Arguments**

- **model**
  A model object with `coef` and `vcov` methods
test.terms Character string or one-sided formula giving name of term or terms to test
null Null hypothesis values for parameters. Default is zeros
df Denominator degrees of freedom for an F test. If NULL these are estimated from the model. Use Inf for a chi-squared test.
method If "Wald", the Wald-type test; if "LRT" the Rao-Scott test based on the estimated log likelihood ratio; If "WorkingWald" the Wald-type test using the variance matrix under simple random sampling
lrt.approximation method for approximating the distribution of the LRT and Working Wald statistic; see pchisqsum.

Details
The Wald test uses a chisquared or F distribution. The two working-model tests come from the (mis-specified) working model where the observations are independent and the weights are frequency weights. For categorical data, this is just the model fitted to the estimated population crosstabulation. The Rao-Scott LRT statistic is the likelihood ratio statistic in this model. The working Wald test statistic is the Wald statistic in this model. The working-model tests do not have a chi-squared sampling distribution: we use a linear combination of chi-squared or F distributions as in pchisqsum. I believe the working Wald test is what SUDAAN refers to as a "Satterthwaite adjusted Wald test".

To match other software you will typically need to use lrt.approximation="satterthwaite"

Value
An object of class regTermTest or regTermTestLRT.

Note
The "LRT" method will not work if the model had starting values supplied for the regression coefficients. Instead, fit the two models separately and use anova(model1, model2, force=TRUE)

References

See Also
anova, vcov, contrasts, pchisqsum
salamander

Examples

data(esoph)
model1 <- glm(cbind(ncases, ncontrols) ~ agegp + tobgp * alcgp, data = esoph, family = binomial())
anova(model1)

regTermTest(model1,"tobgp")
regTermTest(model1,"tobgp:alcgp")
regTermTest(model1, ~alcgp+tobgp:alcgp)

data(api)
dclus2<-svydesign(id=~dnum+snum, weights=~pw, data=apiclus2)
model2<-svyglm(I(sch.wide=="Yes")~ell+meals+mobility, design=dclus2, family=quasibinomial())
regTermTest(model2, ~ell)
regTermTest(model2, ~ell, df=NULL)
regTermTest(model2, ~ell, method="LRT", df=Inf)
regTermTest(model2, ~ell+meals, method="LRT", df=NULL)

regTermTest(model2, ~ell+meals, method="WorkingWald", df=NULL)

salamander

Salamander mating data set from McCullagh and Nelder (1989)

Description

This data set presents the outcome of three experiments conducted at the University of Chicago in 1986 to study interbreeding between populations of mountain dusky salamanders (McCullagh and Nelder, 1989, Section 14.5). The analysis here is from Lumley (1998, section 5.3)

Usage

data(salamander)

Format

A data frame with the following columns:

Mate  Whether the salamanders mated (1) or did not mate (0).
Cross  Cross between female and male type. A factor with four levels: R/R, R/W, W/R, and W/W. The type of the female salamander is listed first and the male is listed second. Rough Butt is represented by R and White Side is represented by W. For example, Cross=W/R indicates a White Side female was crossed with a Rough Butt male.
Male  Identification number of the male salamander. A factor.
Female  Identification number of the female salamander. A factor.
References


Examples

data(salamander)
salamander$mixed<-with(salamander, Cross=="W/R" | Cross=="R/W")
salamander$RWvsWR<-with(salamander, ifelse(mixed,
((Cross=="R/W")-(Cross=="W/R"))/2,
0))
xsalamander<-xdesign(id=list(~Male, ~Female), data=salamander,
overlap= "unbiased")

## Adjacency matrix
## Blocks 1 and 2 are actually the same salamanders, but
## it's traditional to pretend they are independent.
image(xsalamander$adjacency)

## R doesn't allow family=binomial(identity)
success <- svyglm(Mate~mixed+RWvsWR, design=xsalamander,
family=quasi(link="identity", variance="mu(1-mu)")
summary(success)

---

scd Survival in cardiac arrest

Description

These data are from Section 12.2 of Levy and Lemeshow. They describe (a possibly apocryphal) study of survival in out-of-hospital cardiac arrest. Two out of five ambulance stations were sampled from each of three emergency service areas.

Usage

data(scd)

Format

This data frame contains the following columns:

- **ESA** Emergency Service Area (strata)
- **ambulance** Ambulance station (PSU)
- **arrests** estimated number of cardiac arrests
- **alive** number reaching hospital alive

Source

Examples

data(scd)

## survey design objects
scddes<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA,
nest=TRUE, fpc=rep(5,6))
scdnofpc<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA,
nest=TRUE)

# convert to BRR replicate weights
scd2brr <- as.svrepdesign(scdnofpc, type="BRR")
# or to Rao-Wu bootstrap
scd2boot <- as.svrepdesign(scdnofpc, type="subboot")

# use BRR replicate weights from Levy and Lemeshow
repweights<-2*cbind(c(1,0,1,0,1,0), c(1,0,0,1,0,1), c(0,1,1,0,0,1),
c(0,1,0,1,1,0))
scdrep<-svrepdesign(data=scd, type="BRR", repweights=repweights)

# ratio estimates
svyratio(~alive, ~arrests, design=scddes)
svyratio(~alive, ~arrests, design=scdnofpc)
svyratio(~alive, ~arrests, design=scd2brr)
svyratio(~alive, ~arrests, design=scd2boot)
svyratio(~alive, ~arrests, design=scdrep)

# or a logistic regression
summary(svyglm(cbind(alive,arrests-alive)~1, family=quasibinomial, design=scdnofpc))
summary(svyglm(cbind(alive,arrests-alive)~1, family=quasibinomial, design=scdrep))

# Because no sampling weights are given, can't compute design effects
# without replacement: use deff="replace"
svymean(~alive+arrests, scddes, deff=TRUE)
svymean(~alive+arrests, scddes, deff="replace")

Extract standard errors

Description

Extracts standard errors from an object. The default method is for objects with a `vcov` method.

Usage

SE(object, ...)
## Default S3 method:
SE(object,...)
## S3 method for class 'svrepstat'
SE(object,...)
Arguments

object An object

Arguments for future expansion

Value

Vector of standard errors.

See Also

vcov

smoothArea Small area estimation via basic area level model

Description

Generates small area estimates by smoothing direct estimates using an area level model

Usage

svysmoothArea(
  formula,
  domain,
  design = NULL,
  adj.mat = NULL,
  X.domain = NULL,
  direct.est = NULL,
  domain.size = NULL,
  transform = c("identity", "logit", "log"),
  pc.u = 1,
  pc.alpha = 0.01,
  pc.u.phi = 0.5,
  pc.alpha.phi = 2/3,
  level = 0.95,
  n.sample = 250,
  var.tol = 1e-10,
  return.samples = FALSE,
  ...
)

Arguments

formula An object of class 'formula' describing the model to be fitted. If direct.est is specified, the right hand side of the formula is not necessary.

domain One-sided formula specifying factors containing domain labels

design An object of class "svydesign" containing the data for the model
adj.mat  Adjacency matrix with rownames matching the domain labels. If set to NULL, the IID spatial effect will be used.

X.domain  Data frame of areal covariates. One of the column names needs to match the name of the domain variable, in order to be linked to the data input. Currently only supporting time-invariant covariates.

direct.est  Data frame of direct estimates, with first column containing the domain variable, second column containing direct estimate, and third column containing the variance of direct estimate.

domain.size  Data frame of domain sizes. One of the column names needs to match the name of the domain variable, in order to be linked to the data input and there must be a column names 'size' containing domain sizes.

transform  Optional transformation applied to the direct estimates before fitting area level model. The default option is no transformation, but logit and log are implemented.

pc.u  Hyperparameter U for the PC prior on precisions. See the INLA documentation for more details on the parameterization.

pc.alpha  Hyperparameter alpha for the PC prior on precisions.

pc.u.phi  Hyperparameter U for the PC prior on the mixture probability phi in BYM2 model.

pc.alpha.phi  Hyperparameter alpha for the PC prior on the mixture probability phi in BYM2 model.

level  The specified level for the posterior credible intervals

n.sample  Number of draws from posterior used to compute summaries

var.tol  Tolerance parameter; if variance of an area's direct estimator is below this value, that direct estimator is dropped from model

return.samples  If TRUE, return matrix of posterior samples of area level quantities

... for future methods

Details

The basic area level model is a Bayesian version of the Fay-Herriot model (Fay & Herriot,1979). It treats direct estimates of small area quantities as response data and explicitly models differences between areas using covariate information and random effects. The Fay-Herriot model can be viewed as a two-stage model: in the first stage, a sampling model represents the sampling variability of a direct estimator and in the second stage, a linking model describes the between area differences in small area quantities. More detail is given in section 4 of Mercer et al (2015).

Value

A svysae object

References

smoothUnit

Smooth via basic unit level model

Description

Generates small area estimates by smoothing direct estimates using a basic unit level model. This model assumes sampling is ignorable (no selection bias). It's a Bayesian linear (family="gaussian") or generalised linear (family="binomial") mixed model for the unit-level data with individual-level covariates and area-level random effects.

See Also

The survey-sae vignette

Examples

```r
## artificial data from SUMMER package
## Uses too many cores for a CRAN example

## Not run:
hasSUMMER<-tryCatch({
  data("DemoData2",package="SUMMER")
  data("DemoMap2",package="SUMMER")
}, error=function(e) FALSE)
if (!isFALSE(hasSUMMER)){
  library(survey)
  des0 <- svydesign(ids = ~clustid+id, strata = ~strata,
    weights = ~weights, data = DemoData2, nest = TRUE)
  Xmat <- aggregate(age~region, data = DemoData2, FUN = mean)
  cts.cov.res <- svysmoothArea(tobacco.use ~ age,
    domain = ~region,
    design = des0,
    adj.mat = DemoMap2$Amat,
    X.domain = Xmat,
    pc.u = 1,
    pc.alpha = 0.01,
    pc.u.phi = 0.5,
    pc.alpha.phi = 2/3)
  print(cts.cov.res)
  plot(cts.cov.res)
}
## End(Not run)
```
smoothUnit

Usage

svysmoothUnit(
  formula,
  domain,
  design,
  family = c("gaussian", "binomial"),
  X.pop = NULL,
  adj.mat = NULL,
  domain.size = NULL,
  pc.u = 1,
  pc.alpha = 0.01,
  pc.u.phi = 0.5,
  pc.alpha.phi = 2/3,
  level = 0.95,
  n.sample = 250,
  return.samples = FALSE,
  X.pop.weights = NULL,
)

Arguments

formula An object of class 'formula' describing the model to be fitted.
domain One-sided formula specifying factors containing domain labels
design An object of class "survey.design" containing the data for the model
family of the response variable, currently supports 'binomial' (default with logit link
function) or 'gaussian'.
X.pop Data frame of population unit-level covariates. One of the column name needs
to match the domain specified, in order to be linked to the data input. Currently
only supporting time-invariant covariates.
adj.mat Adjacency matrix with rownames matching the domain labels. If set to NULL,
the IID spatial effect will be used.
domain.size Data frame of domain sizes. One of the column names needs to match the name
of the domain variable, in order to be linked to the data input and there must
be a column names 'size' containing domain sizes. The default option is no
transformation, but logit and log are implemented.
pc.u Hyperparameter U for the PC prior on precisions. See the INLA documentation
for more details on the parameterization.
pc.alpha Hyperparameter alpha for the PC prior on precisions.
pc.u.phi Hyperparameter U for the PC prior on the mixture probability phi in BYM2
model.
pc.alpha.phi Hyperparameter alpha for the PC prior on the mixture probability phi in BYM2
model.
level The specified level for the posterior credible intervals
n.sample Number of draws from posterior used to compute summaries
return.samples  If TRUE, return matrix of posterior samples of area level quantities
X.pop.weights  Optional vector of weights to use when aggregating unit level predictions
...  for future expansion

Value

A svysae object

References


See Also

The survey-sae vignette

---

**stratsample**

*Take a stratified sample*

Description

This function takes a stratified sample without replacement from a data set.

Usage

```r
stratsample(strata, counts)
```

Arguments

- `strata`  Vector of stratum identifiers; will be coerced to character
- `counts`  named vector of stratum sample sizes, with names corresponding to the values of `as.character(strata)`

Value

vector of indices into `strata` giving the sample

See Also

- `sample`
  The "sampling" package has many more sampling algorithms.

Examples

```r
data(api)
s<-stratsample(api$stype, c("E"=5,"H"=4,"M"=2))
table(api$stype[s])
```
subset.survey.design  Subset of survey

Description
Restrict a survey design to a subpopulation, keeping the original design information about number of clusters, strata. If the design has no post-stratification or calibration data the subset will use proportionately less memory.

Usage
```r
## S3 method for class 'survey.design'
subset(x, subset, ...)
## S3 method for class 'svyrep.design'
subset(x, subset, ...)
```

Arguments
- `x` A survey design object
- `subset` An expression specifying the subpopulation
- `...` Arguments not used by this method

Value
A new survey design object

See Also
- `svydesign`

Examples
```r
data(fpc)
dfpc<-svydesign(id=~psuid,strat=~stratid,weight=~weight,data=fpc,nest=TRUE)
dsub<-subset(dfpc, x>4)
supmarry(dsub)
svymean(~x, design=dsub)

## These should give the same domain estimates and standard errors
svyby(~x,-I(x>4), design=dfpc, svymean)
suppsummary(svyglm(x-I(x>4)+0, design=dfpc))

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rclus1<-as.svrepdesign(dclus1)
svymean(~enroll, subset(dclus1, sch.wide="Yes" & comp.imp="Yes"))
svymean(~enroll, subset(rclus1, sch.wide="Yes" & comp.imp="Yes"))
```
Description

This help page documents the options that control the behaviour of the survey package.

Details

All the options for the survey package have names beginning with "survey". Four of them control standard error estimation.

options("survey.replicates.mse") controls the default in svrepdesign and as.svrepdesign for computing variances. When options("survey.replicates.mse") is TRUE, the default is to create replicate weight designs that compute variances centered at the point estimate, rather than at the mean of the replicates. The option can be overridden by specifying the mse argument explicitly in svrepdesign and as.svrepdesign. The default is FALSE.

When options("survey.ultimate.cluster") is TRUE, standard error estimation is based on independence of PSUs at the first stage of sampling, without using any information about subsequent stages. When FALSE, finite population corrections and variances are estimated recursively. See svyrecvar for more information. This option makes no difference unless first-stage finite population corrections are specified, in which case setting the option to TRUE gives the wrong answer for a multistage study. The only reason to use TRUE is for compatibility with other software that gives the wrong answer.

Handling of strata with a single PSU that are not certainty PSUs is controlled by options("survey.lonely.psu"). The default setting is "fail", which gives an error. Use "remove" to ignore that PSU for variance computation, "adjust" to center the stratum at the population mean rather than the stratum mean, and "average" to replace the variance contribution of the stratum by the average variance contribution across strata. As of version 3.4-2 as.svrepdesign also uses this option.

The variance formulas for domain estimation give well-defined, positive results when a stratum contains only one PSU with observations in the domain, but are not unbiased. If options("survey.adjust.domain.lonely") is TRUE and options("survey.lonely.psu") is "average" or "adjust" the same adjustment for lonely PSUs will be used within a domain. Note that this adjustment is not available for replicate-weight designs, nor (currently) for raked, post-stratified, or calibrated designs.

The fourth option is options("survey.want.obsolete"). This controls the warnings about using the deprecated pre-2.9.0 survey design objects.

The behaviour of replicate-weight designs for self-representing strata is controlled by options("survey.drop.replicates"). When TRUE, various optimizations are used that take advantage of the fact that these strata do not contribute to the variance. The only reason ever to use FALSE is if there is a bug in the code for these optimizations.

The fifth option controls the use of multiple processors with the multicore package. This option should not affect the values computed by any of the survey functions. If TRUE, all functions that are able to use multiple processors will do so by default. Using multiple processors may speed up calculations, but need not, especially if the computer is short on memory. The best strategy is probably to experiment with explicitly requesting multicore=TRUE in functions that support it, to see if there is an increase in speed before setting the global option.
survey.use_rcpp controls whether the new C++ code for standard errors is used (vs the old R code). The factory setting is TRUE and the only reason to use FALSE is for comparisons.

---

**surveysummary**  
Summary statistics for sample surveys

**Description**

Compute means, variances, ratios and totals for data from complex surveys.

**Usage**

```r
## S3 method for class 'survey.design'
svymean(x, design, na.rm=FALSE, deff=FALSE, influence=FALSE, ...)
## S3 method for class 'survey.design2'
svymean(x, design, na.rm=FALSE, deff=FALSE, influence=FALSE, ...)
## S3 method for class 'twophase'
svymean(x, design, na.rm=FALSE, deff=FALSE, ...)
## S3 method for class 'svyrepproj'
svymean(x, design, na.rm=FALSE, rho=NULL,
        return.replicates=FALSE, deff=FALSE, ...)
## S3 method for class 'survey.design'
svyvar(x, design, na.rm=FALSE, ...)
## S3 method for class 'svyrepproj'
svyvar(x, design, na.rm=FALSE, rho=NULL,
        return.replicates=FALSE, ...)
## S3 method for class 'survey.design'
svytotal(x, design, na.rm=FALSE, deff=FALSE, ...)
## S3 method for class 'survey.design2'
svytotal(x, design, na.rm=FALSE, deff=FALSE, ...)
## S3 method for class 'twophase'
svytotal(x, design, na.rm=FALSE, deff=FALSE, ...)
## S3 method for class 'svyrepproj'
svytotal(x, design, na.rm=FALSE, rho=NULL,
        return.replicates=FALSE, deff=FALSE, ...)
## S3 method for class 'svystat'
coef(object, ...)
## S3 method for class 'svyrepproj'
coef(object, ...)
## S3 method for class 'svystat'
vcov(object, ...)
## S3 method for class 'svyrepproj'
vcov(object, ...)
## S3 method for class 'svystat'
confint(object, parm, level = 0.95, df = Inf, ...)
## S3 method for class 'svyrepproj'
confint(object, parm, level = 0.95, df = Inf, ...)
```
cv(object,...)
deff(object, quietly=FALSE,...)
make.formula(names)

Arguments

x  A formula, vector or matrix
design  survey.design or svyrep.design object
na.rm  Should cases with missing values be dropped?
influence  Should a matrix of influence functions be returned (primarily to support svyby)
rho  parameter for Fay’s variance estimator in a BRR design
return.replicates  Return the replicate means_totals?
deff  Return the design effect (see below)
object  The result of one of the other survey summary functions
quietly  Don’t warn when there is no design effect computed
estimate.only  Don’t compute standard errors (useful when svyvar is used to estimate the design effect)
parm  a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level  the confidence level required.
df  degrees of freedom for t-distribution in confidence interval, use degf(design) for number of PSUs minus number of strata
...  additional arguments to methods, not currently used
names  vector of character strings

Details

These functions perform weighted estimation, with each observation being weighted by the inverse of its sampling probability. Except for the table functions, these also give precision estimates that incorporate the effects of stratification and clustering.

Factor variables are converted to sets of indicator variables for each category in computing means and totals. Combining this with the interaction function, allows crosstabulations. See ftable.svystat for formatting the output.

With na.rm=TRUE, all cases with missing data are removed. With na.rm=FALSE cases with missing data are not removed and so will produce missing results. When using replicate weights and na.rm=FALSE it may be useful to set options(na.action="na.pass"), otherwise all replicates with any missing results will be discarded.

The svytotal and svreptotal functions estimate a population total. Use predict on svyratio and svyglm, to get ratio or regression estimates of totals.

svyvar estimates the population variance. The object returned includes the full matrix of estimated population variances and covariances, but by default only the diagonal elements are printed. To display the whole matrix use as.matrix(v) or print(v,covariance=TRUE).
The design effect compares the variance of a mean or total to the variance from a study of the same size using simple random sampling without replacement. Note that the design effect will be incorrect if the weights have been rescaled so that they are not reciprocals of sampling probabilities. To obtain an estimate of the design effect comparing to simple random sampling with replacement, which does not have this requirement, use `deff="replace"`. This with-replacement design effect is the square of Kish’s "deft".

The design effect for a subset of a design conditions on the size of the subset. That is, it compares the variance of the estimate to the variance of an estimate based on a simple random sample of the same size as the subset, taken from the subpopulation. So, for example, under stratified random sampling the design effect in a subset consisting of a single stratum will be 1.0.

The `cv` function computes the coefficient of variation of a statistic such as ratio, mean or total. The default method is for any object with methods for `SE` and `coef`.

`make.formula` makes a formula from a vector of names. This is useful because formulas as the best way to specify variables to the survey functions.

**Value**

Objects of class "svystat" or "svrepstat", which are vectors with a "var" attribute giving the variance and a "statistic" attribute giving the name of the statistic.

These objects have methods for `vcov`, `SE`, `coef`, `confint`, `svycontrast`.

**Author(s)**

Thomas Lumley

**See Also**

`svydesign`, `as.svrepdesign`, `svrepdesign` for constructing design objects.

`degf` to extract degrees of freedom from a design.

`svyquantile` for quantiles

`ftable.svystat` for more attractive tables

`sveciprop` for more accurate confidence intervals for proportions near 0 or 1.

`svytest` for comparing two means.

`svycontrast` for linear and nonlinear functions of estimates.

**Examples**

```r
data(api)

## one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

svymean(~api00, dclus1, deff=TRUE)
svymean(~factor(stype), dclus1)
svymean(~interaction(stype, comp.imp), dclus1)
svyquantile(~api00, dclus1, c(.25,.5,.75))
```
svytotal(~enroll, dclus1, deff=TRUE)
svratio(~api.stu, ~enroll, dclus1)

v<-svyvar(~api00+api99, dclus1)
v
print(v, cov=TRUE)
as.matrix(v)

# replicate weights - jackknife (this is slower)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw,  
data=apistrat, fpc=~fpc)
jkstrat<-as.svrepdesign(dstrat)

svymean(~api00, jkstrat)
svymean(~factor(stype), jkstrat)
svyvar(~api00+api99, jkstrat)

svyquantile(~api00, jkstrat, c(.25,.5,.75))
svytotal(~enroll, jkstrat)
svyratio(~api.stu, ~enroll, jkstrat)

# coefficients of variation
CV(svytotal(~enroll,dstrat))
CV(svratio(~api.stu, ~enroll, jkstrat))

# extracting information from the results
coef(svytotal(~enroll,dstrat))
vcov(svymean(~api00+api99, jkstrat))
SE(svymean(~enroll, dstrat))
confint(svymean(~api00+api00, dclus1))
confint(svymean(~api00+api00, dclus1), df=degf(dclus1))

# Design effect
svymean(~api00, dstrat, deff=TRUE)
svymean(~api00, dstrat, deff="replace")
svymean(~api00, jkstrat, deff=TRUE)
svymean(~api00, jkstrat, deff="replace")
(a<-svytotal(~enroll, dclus1, deff=TRUE))
deff(a)

## weights that are *already* calibrated to population size
sum(weights(dclus1))
nrow(apiipop)
cdclus1<- svydesign(id=~dnum, weights=~pw, data=apiclus1,  
fpc=~fpc,calibrate.formula=~1)
SE(svymean(~enroll, dclus1))

## not equal to SE(mean)
SE(svytotal(~enroll, dclus1))/nrow(apiipop)

## equal to SE(mean)
SE(svytotal(~enroll, cdclus1))/nrow(apiipop)
svrepdesign

---

**svrepdesign**

Specify survey design with replicate weights

---

**Description**

Some recent large-scale surveys specify replication weights rather than the sampling design (partly for privacy reasons). This function specifies the data structure for such a survey.

**Usage**

```r
svrepdesign(variables, repweights, weights, data, degf=NULL,...)
```

---

**Arguments**

- **variables**
  - formula or data frame specifying variables to include in the design (default is all)
- **repweights**
  - formula or data frame specifying replication weights, or character string specifying a regular expression that matches the names of the replication weight variables
- **weights**
  - sampling weights
- **data**
  - data frame to look up variables in formulas, or character string giving name of database table
- **degf**
  - Design degrees of freedom; use NULL to have the function work this out for you
**Type of replication weights**

- **combined.weights**: TRUE if the `repweights` already include the sampling weights. This is usually the case.

- **rho**: Shrinkage factor for weights in Fay’s method

- **bootstrap.average**: For `type="bootstrap"`, if the bootstrap weights have been averaged, gives the number of iterations averaged over

- **scale, rscales**: Scaling constant for variance, see Details below

- **fpc, fpctype**: Finite population correction information

- **mse**: If TRUE, compute variances based on sum of squares around the point estimate, rather than the mean of the replicates

- **dbname**: name of database, passed to `DBI::dbConnect()`

- **dbtype**: Database driver: see Details

- **x**: survey design with replicate weights

- **...**: Other arguments to `image`

- **col**: Colors

- **type**: "rep" for only the replicate weights, "total" for the replicate and sampling weights combined.

**Details**

In the BRR method, the dataset is split into halves, and the difference between halves is used to estimate the variance. In Fay’s method, rather than removing observations from half the sample they are given weight \( \rho \) in one half-sample and \( 2-\rho \) in the other. The ideal BRR analysis is restricted to a design where each stratum has two PSUs, however, it has been used in a much wider class of surveys. The `scale` and `rscales` arguments will be ignored (with a warning) if they are specified.

The JK1 and JKn types are both jackknife estimators deleting one cluster at a time. JKn is designed for stratified and JK1 for unstratified designs.

The successive-difference weights in the American Community Survey automatically use `scale = 4/ncol(repweights)` and `rscales=rep(1, ncol(repweights))`. This can be specified as `type="ACS"` or `type="successive-difference"`. The `scale` and `rscales` arguments will be ignored (with a warning) if they are specified.

JK2 weights (type="JK2"), as in the California Health Interview Survey, automatically use `scale=1, rscales=rep(1, ncol(repweights))`. The `scale` and `rscales` arguments will be ignored (with a warning) if they are specified.

Averaged bootstrap weights ("mean bootstrap") are used for some surveys from Statistics Canada. Yee et al (1999) describe their construction and use for one such survey.

The variance is computed as the sum of squared deviations of the replicates from their mean. This may be rescaled: `scale` is an overall multiplier and `rscales` is a vector of replicate-specific multipliers for the squared deviations. That is, `rscales` should have one entry for each column of `repweights` if the replication weights incorporate the sampling weights (`combined.weights=TRUE`) or for `type="other"` these must be specified, otherwise they can be guessed from the weights.
A finite population correction may be specified for type="other", type="JK1" and type="JKn". fpc must be a vector with one entry for each replicate. To specify sampling fractions use fpctype="fraction" and to specify the correction directly use fpctype="correction".

The design degrees of freedom are returned by degf. By default they are computed from the numerical rank of the repweights. This is slow for very large data sets and you can specify a value instead.

repweights may be a character string giving a regular expression for the replicate weight variables. For example, in the California Health Interview Survey public-use data, the sampling weights are "rakedw0" and the replicate weights are "rakedw1" to "rakedw80". The regular expression "rakedw[1-9]" matches the replicate weight variables (and not the sampling weight variable).

data may be a character string giving the name of a table or view in a relational database that can be accessed through the DBI interface. For DBI interfaces dbtype should be the name of the database driver and dbname should be the name by which the driver identifies the specific database (eg file name for SQLite).

The appropriate database interface package must already be loaded (eg RSQLite for SQLite). The survey design object will contain the replicate weights, but actual variables will be loaded from the database only as needed. Use close to close the database connection and open to reopen the connection, eg, after loading a saved object.

The database interface does not attempt to modify the underlying database and so can be used with read-only permissions on the database.

To generate your own replicate weights either use as.svrepdesign on a survey.design object, or see brrweights, bootweights, jk1weights and jknweights.

The model.frame method extracts the observed data.

Value

Object of class svyrep.design, with methods for print, summary, weights, image.

Note

To use replication-weight analyses on a survey specified by sampling design, use as.svrepdesign to convert it.

References


See Also

as.svrepdesign, svydesign, brrweights, bootweights
svrVar

Compute variance from replicates

Description

Compute an appropriately scaled empirical variance estimate from replicates. The `mse` argument specifies whether the sums of squares should be centered at the point estimate (`mse=TRUE`) or the mean of the replicates. It is usually taken from the `mse` component of the design object.

Usage

```r
svrVar(thetas, scale, rscales, na.action=getOption("na.action"),
       mse=getOption("survey.replicates.mse"), coef)
```

Arguments

- `thetas`: matrix whose rows are replicates (or a vector of replicates)
- `scale`: Overall scaling factor
svy.varcoef

Description
Computes the sandwich variance estimator for a generalised linear model fitted to data from a complex sample survey. Designed to be used internally by svyglm.

Usage
svy.varcoef(glm.object, design)

Arguments
- glm.object: A glm object
- design: A survey.design object

Value
A variance matrix

Author(s)
Thomas Lumley

See Also
svyglm, svydesign, svyCprod
svyby

Survey statistics on subsets

Description

Compute survey statistics on subsets of a survey defined by factors.

Usage

svyby(formula, by, design, ..., deff=FALSE, keep.var=TRUE, keep.names=TRUE, verbose=FALSE, vartype=c("se","ci","ci","cv","cvpct","var"), drop.empty.groups=TRUE, covmat=FALSE, return.replicates=FALSE, na.rm.by=FALSE, na.rm.all=FALSE, stringsAsFactors=TRUE, multicore=getOption("survey.multicore"))

Arguments

formula,x A formula specifying the variables to pass to FUN (or a matrix, data frame, or vector)

by A formula specifying factors that define subsets, or a list of factors.

design A svydesign or svrepdesign object

FUN A function taking a formula and survey design object as its first two arguments.

... Other arguments to FUN. NOTE: if any of the names of these are partial matches to formula, by, or design, you must specify the formula, by, or design argument by name, not just by position.

deff Request a design effect from FUN
keep.var
If FUN returns a svystat object, extract standard errors from it

keep.names
Define row names based on the subsets

verbose
If TRUE, print a label for each subset as it is processed.

vartype
Report variability as one or more of standard error, confidence interval, coefficient of variation, percent coefficient of variation, or variance

drop.empty.groups
If FALSE, report NA for empty groups, if TRUE drop them from the output

na.rm.by
If true, omit groups defined by NA values of the by variables

na.rm.all
If true, check for groups with no non-missing observations for variables defined by formula and treat these groups as empty. Doesn’t make much sense without na.rm=TRUE

covmat
If TRUE, compute covariances between estimates for different subsets. Allows svycontrast to be used on output. Requires that FUN supports either return.replicates=TRUE or influence=TRUE

return.replicates
Only for replicate-weight designs. If TRUE, return all the replicates as the "replicates" attribute of the result

influence
Return the influence functions of the result

multicore
Use multicore package to distribute subsets over multiple processors?

stringsAsFactors
Convert any string variables in formula to factors before calling FUN, so that the factor levels will be the same in all groups (See Note below). Potentially slow.

parm
A specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.

level
The confidence level required.

df
Degrees of freedom for t-distribution in confidence interval, use degf(design) for number of PSUs minus number of strata

object
An object of class "svyby"

bys
One-sided formula with each term specifying a grouping (rather than being combined to give a grouping)

Details

The variance type "ci" asks for confidence intervals, which are produced by confint. In some cases additional options to FUN will be needed to produce confidence intervals, for example, svyquantile needs ci=TRUE or keep.var=FALSE.

unwtd.count is designed to be passed to svyby to report the number of non-missing observations in each subset. Observations with exactly zero weight will also be counted as missing, since that’s how subsets are implemented for some designs.

Parallel processing with multicore=TRUE is useful only for fairly large problems and on computers with sufficient memory. The multicore package is incompatible with some GUIs, although the Mac Aqua GUI appears to be safe.

The variant svybys creates a separate table for each term in bys rather than creating a joint table.
Value

An object of class "svyby": a data frame showing the factors and the results of FUN.

For unwtd.count, the unweighted number of non-missing observations in the data matrix specified by x for the design.

Note

The function works by making a lot of calls of the form FUN(formula, subset(design, by==i)), where formula is re-evaluated in each subset, so it is unwise to use data-dependent terms in formula. In particular, svyby(~factor(a), ~b, design=d, svymean), will create factor variables whose levels are only those values of a present in each subset. If a is a character variable then svyby(~a, ~b, design=d, svymean) creates factor variables implicitly and so has the same problem. Either use update.survey.design to add variables to the design object instead or specify the levels explicitly in the call to factor. The stringsAsFactors=TRUE option converts all character variables to factors, which can be slow, set it to FALSE if you have predefined factors where necessary.

Note

Asking for a design effect (deff=TRUE) from a function that does not produce one will cause an error or incorrect formatting of the output. The same will occur with keep.var=TRUE if the function does not compute a standard error.

See Also

svytable and ftable.svystat for contingency tables, ftable.svyby for pretty-printing of svyby

Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

svyby(~api99, ~stype, dclus1, svymean)
svyby(~api99, ~stype, dclus1, svyquantile, quantiles=0.5, ci=TRUE, vartype="ci")
## without ci=TRUE svyquantile does not compute standard errors
svyby(~api99, ~stype, dclus1, svyquantile, quantiles=0.5, keep.var=FALSE)
svyby(~api99, list(school.type=apiclus1$stype), dclus1, svymean)
svyby(~api99+api00, ~stype, dclus1, svymean, deff=TRUE, vartype="ci")
svyby(~api99+api00, ~stype+sch.wide, dclus1, svymean, keep.var=FALSE)
## report raw number of observations
svyby(~api99+api00, ~stype+sch.wide, dclus1, unwtd.count, keep.var=FALSE)

rclus1<-as.svrepdesign(dclus1)

svyby(~api99, ~stype, rclus1, svymean)
svyby(~api99, ~stype, rclus1, svyquantile, quantiles=0.5)
svyby(~api99, list(school.type=apiclus1$stype), rclus1, svymean, vartype="cv")
svyby(~enroll, ~stype, rclus1,svytotal, deff=TRUE)
svyby(~api99+api00, ~stype+sch.wide, rclus1, svymean, keep.var=FALSE)
##report raw number of observations
svyby(~api99+api00, ~stype+sch.wide, rclus1, unwtd.count, keep.var=FALSE)

## comparing subgroups using covmat=TRUE
mns<-svyby(~api99, ~stype, rclus1, svymean, covmat=TRUE)
vcov(mns)
svycontrast(mns, c(E = 1, M = -1))

str(svyby(~api99, ~stype, rclus1, svymean, return.replicates=TRUE))

tots<-svyby(~enroll, ~stype, dclus1, svytotal, covmat=TRUE)
vcov(tots)
svycontrast(tots, quote(E/H))

## comparing subgroups uses the delta method unless replicates are present
meanlogs<-svyby(~log(enroll), ~stype, svymean, design=rclus1, covmat=TRUE)
svycontrast(meanlogs, quote(exp(E-H)))
meanlogs<-svyby(~log(enroll), ~stype, svymean, design=rclus1, covmat=TRUE, return.replicates=TRUE)
svycontrast(meanlogs, quote(exp(E-H)))

## extractor functions
(a<-svyby(~enroll, ~stype, rclus1, svytotal, deff=TRUE, verbose=TRUE,
    vartype=c("se", "cv", "cvpct", "var")))
deff(a)
SE(a)
cv(a)
coef(a)
confint(a, df=degf(rclus1))

## ratio estimates
svyby(~api.stu, by=~stype, denominator=~enroll, design=dclus1, sivratio)
ratios<-svyby(~api.stu, by=~stype, denominator=~enroll, design=dclus1, sivratio, covmat=TRUE)
vcov(ratios)

## empty groups
svyby(~api00, ~comp.imp+sch.wide, design=dclus1, svymean)
svyby(~api00, ~comp.imp+sch.wide, design=dclus1, svymean, drop.empty.groups=FALSE)

## Multiple tables
svyby(~api00, ~comp.imp+sch.wide, design=dclus1, svymean)
Description

Estimates the population cumulative distribution function for specified variables. In contrast to \texttt{svyquantile}, this does not do any interpolation: the result is a right-continuous step function.

Usage

\begin{verbatim}
svycdf(formula, design, na.rm = TRUE,...)
## S3 method for class 'svycdf'
print(x,...)
## S3 method for class 'svycdf'
plot(x,xlab=NULL,...)
\end{verbatim}

Arguments

- \texttt{formula}: one-sided formula giving variables from the design object
- \texttt{design}: survey design object
- \texttt{na.rm}: remove missing data (case-wise deletion)?
- \texttt{...}: other arguments to \texttt{plot.stepfun}
- \texttt{x}: object of class \texttt{svycdf}
- \texttt{xlab}: a vector of x-axis labels or \texttt{NULL} for the default labels

Value

An object of class \texttt{svycdf}, which is a list of step functions (of class \texttt{stepfun})

See Also

\texttt{svyquantile, svyhist, plot.stepfun}

Examples

\begin{verbatim}
data(api)
dstrat <- svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
   fpc = ~fpc)
cdf.est<-svycdf(~enroll+api00+api99, dstrat)
cdf.est
## function
cdf.est[[1]]
## evaluate the function
cdf.est[[1]](800)
cdf.est[[2]](800)

## compare to population and sample CDFs.
opar<-par(mfrow=c(2,1))
cdf.pop<-ecdf(api$enroll)
cdf.samp<-ecdf(api$enroll)
plot(cdf.pop,main="Population vs sample", xlab="Enrollment")
lines(cdf.samp,col.points="red")
\end{verbatim}
svyciprop

Confidence intervals for proportions

Description

Computes confidence intervals for proportions using methods that may be more accurate near 0 and 1 than simply using confint(svymean()).

Usage

svyciprop(formula, design, method = c("logit", "likelihood", "asin", "beta", "mean","xlogit"), level = 0.95, df=degf(design),...)

Arguments

formula: Model formula specifying a single binary variable

design: survey design object

method: See Details below. Partial matching is done on the argument.

level: Confidence level for interval

df: denominator degrees of freedom, for all methods except "beta". Use Inf for confidence intervals based on a Normal distribution, and for "likelihood" and "logit" use NULL for the default method in glms (currently degf(design)-1, but this may be improved in the future)

...: For "mean" and "asin", this is passed to confint.svystat

Details

The "logit" method fits a logistic regression model and computes a Wald-type interval on the log-odds scale, which is then transformed to the probability scale.

The "likelihood" method uses the (Rao-Scott) scaled chi-squared distribution for the loglikelihood from a binomial distribution.

The "asin" method uses the variance-stabilising transformation for the binomial distribution, the arcsine square root, and then back-transforms the interval to the probability scale.

The "beta" method uses the incomplete beta function as in binom.test, with an effective sample size based on the estimated variance of the proportion. (Korn and Graubard, 1998)

The "xlogit" method uses a logit transformation of the mean and then back-transforms to the probability scale. This appears to be the method used by SUDAAN and SPSS COMPLEX SAMPLES.

The "mean" method is a Wald-type interval on the probability scale, the same as confint(svymean())
All methods undercover for probabilities close enough to zero or one, but "beta", "likelihood", "logit", and "logit" are noticeably better than the other two. None of the methods will work when the observed proportion is exactly 0 or 1.

The `confint` method extracts the confidence interval; the `vcov` and `SE` methods just report the variance or standard error of the mean.

**Value**

The point estimate of the proportion, with the confidence interval as an attribute

**References**


**See Also**

`s涯mean`, `yrbs`

**Examples**

data(api)
dclus1<-svydesign(id=~dnum, fpc=~fpc, data=apiclus1)

svyciprop(~I(ell==0), dclus1, method="li")
svyciprop(~I(ell==0), dclus1, method="lo")
svyciprop(~I(ell==0), dclus1, method="as")
svyciprop(~I(ell==0), dclus1, method="be")
svyciprop(~I(ell==0), dclus1, method="me")
svyciprop(~I(ell==0), dclus1, method="xl")

## reproduces Stata svy: mean
svyciprop(~I(ell==0), dclus1, method="me", df=degf(dclus1))
## reproduces Stata svy: prop
svyciprop(~I(ell==0), dclus1, method="lo", df=degf(dclus1))

rclus1<-as.svrepdesign(dclus1)
svyciprop(~I(emer==0), rclus1, method="li")
svyciprop(~I(emer==0), rclus1, method="lo")
svyciprop(~I(emer==0), rclus1, method="as")
svyciprop(~I(emer==0), rclus1, method="be")
svyciprop(~I(emer==0), rclus1, method="me")
svycontrast

Linear and nonlinear contrasts of survey statistics

Description

Computes linear or nonlinear contrasts of estimates produced by survey functions (or any object with coef and vcov methods).

Usage

svycontrast(stat, contrasts, add=FALSE, ...)

Arguments

stat
  object of class svrepstat or svystat
contrasts
  A vector or list of vectors of coefficients, or a call or list of calls
add
  keep all the coefficients of the input in the output?
...
  For future expansion

Details

If contrasts is a list, the element names are used as names for the returned statistics.

If an element of contrasts is shorter than coef(stat) and has names, the names are used to match up the vectors and the remaining elements of contrasts are assumed to be zero. If the names are not legal variable names (e.g. \(0.1\)) they must be quoted (e.g. "0.1")

If contrasts is a "call" or list of "call"s, and stat is a svrepstat object including replicates, the replicates are transformed and used to compute the variance. If stat is a svystat object or a svrepstat object without replicates, the delta-method is used to compute variances, and the calls must use only functions that deriv knows how to differentiate. If the names are not legal variable names they must be quoted with backticks (e.g. `0.1`).

If stats is a svyvar object, the estimates are elements of a matrix and the names are the row and column names pasted together with a colon.

Value

Object of class svrepstat or svystat or svyvar

See Also

regTermTest, svyglm
`svycontrast` Examples

```r
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

a <- svytotal(~api00+enroll+api99, dclus1)
svycontrast(a, list(avg=c(0.5,0,0.5), diff=c(1,0,-1)))
## if contrast vectors have names, zeroes may be omitted
svycontrast(a, list(avg=c(api00=0.5,api99=0.5), diff=c(api00=1,api99=-1)))

## nonlinear contrasts
svycontrast(a, quote(api00/api99))
svratio(~api00, ~api99, dclus1)

## Example: standardised skewness coefficient
moments<-svymean(~I(api00^3)+I(api00^2)*I(api00), dclus1)
svycontrast(moments, quote((\(I(api00^3)^{-3}*I(api00^2)*I(api00) + 3*I(api00)^2*I(api00)^{-2} - I(api00)^{-3})/ (I(api00^2)^{-1.5}))))

## Example: geometric means
## using delta method
meanlogs <- svymean(~log(api00)+log(api99), dclus1)
svycontrast(meanlogs, list(api00=quote(exp(`\(log(api00)`))), api99=quote(exp(`\(log(api99)`)))))

## using delta method
rclus1<-as.svrepdesign(dclus1)
meanlogs <- svymean(~log(api00)+log(api99), rclus1)
svycontrast(meanlogs, list(api00=quote(exp(`\(log(api00)`))), api99=quote(exp(`\(log(api99)`)))))

## why is add=TRUE useful?
(totals<-svyby(~enroll, ~stype, design=dclus1, svytotal, covmat=TRUE))
totals1<-svycontrast(totals, list(total=c(1,1,1)), add=TRUE)
svycontrast(totals1, list(quote(E/total), quote(H/total), quote(M/total)))

totals2<-svycontrast(totals, list(total=quote(E+H+M)), add=TRUE)
all.equal(as.matrix(totals1), as.matrix(totals2))

## more complicated svyby
means <- svyby(~api00+api99, ~stype+sch.wide, design=dclus1, svymean, covmat=TRUE)
svycontrast(means, quote(`\(E.No:api00`)-`\(E.No:api99`))
svycontrast(means, quote(`\(E.No:api00`)/`\(E.No:api99`))

## transforming replicates
meanlogs_r <- svymean(~log(api00)+log(api99), rclus1, return.replicates=TRUE)
svycontrast(meanlogs_r, list(api00=quote(exp(`\(log(api00)`))), api99=quote(exp(`\(log(api99)`)))))
```
## converting covariances to correlations

```r
vmat <- svyvar(~api00+ell,dclus1)
print(vmat,cov=TRUE)
cov2cor(as.matrix(vmat))[1,2]
svycontrast(vmat, quote(`api00:ell'/sqrt(`api00:api00`*`ell:ell`))))
```

---

svycoplot  

### Conditioning plots of survey data

**Description**

Draws conditioned scatterplots ('Trellis' plots) of survey data using hexagonal binning or transparency.

**Usage**

```r
svycoplot(formula, design, style = c("hexbin", "transparent"), basecol = "black", alpha = c(0, 0.8), hexscale=c("relative","absolute"), ...)
```

**Arguments**

- `formula`: A graph formula suitable for `xyplot`
- `design`: A survey design object
- `style`: Hexagonal binning or transparent color?
- `basecol`: The fully opaque 'base' color for creating transparent colors. This may also be a function; see `svyplot` for details
- `alpha`: Minimum and maximum opacity
- `hexscale`: Scale hexagons separate for each panel (relative) or across all panels (absolute)
- `...`: Other arguments passed to `grid.hexagons` or `xyplot`

**Value**

An object of class `trellis`

**Note**

As with all 'Trellis' graphs, this function creates an object but does not draw the graph. When used inside a function or non-interactively you need to `print()` the result to create the graph.

**See Also**

`svyplot`
Examples

```r
data(api)
dclus2 <- svydesign(id = ~dnum + snum, weights = ~pw,
                   data = apiclus2, fpc = ~fpc1 + fpc2)
svycoplot(api00 ~ api99 | sch.wide * comp.imp, design = dclus2, style = "hexbin")
svycoplot(api00 ~ api99 | sch.wide * comp.imp, design = dclus2, style = "hexbin", hexscale = "absolute")
svycoplot(api00 ~ api99 | sch.wide, design = dclus2, style = "trans")
svycoplot(api00 ~ meals | stype, design = dclus2,
          style = "transparent",
          basecol = function(d) c("darkred", "purple", "forestgreen") [as.numeric(d$stype)],
          alpha = c(0, 1))
```

svycoxph

**Survey-weighted Cox models.**

Description

Fit a proportional hazards model to data from a complex survey design.

Usage

```r
svycoxph(formula, design, subset=NULL, rescale=TRUE, ...)
## S3 method for class 'svycoxph'
predict(object, newdata, se=FALSE,
          type=c("lp", "risk", "terms", "curve"), ...)
## S3 method for class 'svycoxph'
AIC(object, ..., k = 2)
```

Arguments

- `formula` Model formula. Any `cluster()` terms will be ignored.
- `design` survey.design object. Must contain all variables in the formula
- `subset` Expression to select a subpopulation
- `rescale` Rescale weights to improve numerical stability
- `object` A svycoxph object
- `newdata` New data for prediction
- `se` Compute standard errors? This takes a lot of memory for type="curve"
- `type` "curve" does predicted survival curves. The other values are passed to predict.coxph()
- `...` For AIC, more models to compare the AIC of. For svycoxph, other arguments passed to coxph.
- `k` The penalty per parameter that would be used under independent sampling: AIC has k=2
The main difference between `svycoxph` function and the `robust=TRUE` option to `coxph` in the survival package is that this function accounts for the reduction in variance from stratified sampling and the increase in variance from having only a small number of clusters.

Note that `strata` terms in the model formula describe subsets that have a separate baseline hazard function and need not have anything to do with the stratification of the sampling.

The AIC method uses the same approach as `AIC.svyglm`, though the relevance of the criterion this optimises is a bit less clear than for generalised linear models.

The standard errors for predicted survival curves are available only by linearization, not by replicate weights (at the moment). Use `withReplicates` to get standard errors with replicate weights. Predicted survival curves are not available for stratified Cox models.

The standard errors use the delta-method approach of Williams (1995) for the Nelson-Aalen estimator, modified to handle the Cox model following Tsiatis (1981). The standard errors agree closely with `survfit.coxph` for independent sampling when the model fits well, but are larger when the model fits poorly. I believe the standard errors are equivalent to those of Lin (2000), but I don’t know of any implementation that would allow a check.

An object of class `svycoxph` for `svycoxph`, an object of class `svykmlist` for `predict(, type="curve")`.

The standard error calculation for survival curves uses memory proportional to the sample size times the square of the number of events.

Thomas Lumley

Binder DA. (1992) Fitting Cox’s proportional hazards models from survey data. Biometrika 79: 139-147

See Also

`coxph`, `predict.coxph`, `svykm` for estimation of Kaplan-Meier survival curves and for methods that operate on survival curves.
`regTermTest` for Wald and (Rao-Scott) likelihood ratio tests for one or more parameters.
## Examples

### Somewhat unrealistic example of nonresponse bias.
```r
data(pbc, package="survival")
pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
biasmodel <- glm(randomized~age*edema, data=pbc, family=binomial)
pbc$randprob <- fitted(biasmodel)
if (is.null(pbc$albumin)) pbc$albumin <- pbc$alb  ## pre 2.9.0

dpbc <- svydesign(id=~1, prob=~randprob, strata=~edema, data=subset(pbc, randomized))
rpbc <- as.svrepdesign(dpbc)

(model <- svycoxph(Surv(time, status>0)~log(bili)+protime+albumin, design=dpbc))

svycoxph(Surv(time, status>0)~log(bili)+protime+albumin, design=rpbc)
s <- predict(model, se=TRUE, type="curve",
newdata=data.frame(bili=c(3,9), protime=c(10,10), albumin=c(3.5,3.5)))
plot(s[[1]], ci=TRUE, col="sienna")
lines(s[[2]], ci=TRUE, col="royalblue")
quantile(s[[1]], ci=TRUE)
confint(s[[2]], parm=365*(1:5))
```

---

### svyCprod

**Computations for survey variances**

#### Description

Computes the sum of products needed for the variance of survey sample estimators. `svyCprod` is used for survey design objects from before version 2.9, `onestage` is called by `svyrecvar` for post-2.9 design objects.

#### Usage

```r
svyCprod(x, strata, psu, fpc, nPSU, certainty=NULL, postStrata=NULL,
lonely.psu=getOption("survey.lonely.psu"))
onestage(x, strata, clusters, nPSU, fpc,
lonely.psu=getOption("survey.lonely.psu"), stage=0, cal)
```

#### Arguments

- **x**: A vector or matrix
- **strata**: A vector of stratum indicators (may be NULL for `svyCprod`)
- **psu**: A vector of cluster indicators (may be NULL)
- **clusters**: A vector of cluster indicators
- **fpc**: A data frame (`svyCprod`) or vector (`onestage`) of population stratum sizes, or NULL
nPSU  Table (svyprod) or vector (onestage) of original sample stratum sizes (or NULL)
certainty  logical vector with stratum names as names. If TRUE and that stratum has a
degree of freedom correction and by a finite population correction (if supplied) and added across
postStrata  Post-stratification variables
lonely.psu One of "remove", "adjust", "fail", "certainty", "average". See Details
above
stage  Used internally to track the depth of recursion
cal  Used to pass calibration information at stages below the population

Details

The observations for each cluster are added, then centered within each stratum and the outer product
is taken of the row vector resulting for each cluster. This is added within strata, multiplied by a
degrees-of-freedom correction and by a finite population correction (if supplied) and added across
strata.

If there are fewer clusters (PSUs) in a stratum than in the original design extra rows of zeroes are
added to \( x \) to allow the correct subpopulation variance to be computed.

See postStratify for information about post-stratification adjustments.

The variance formula gives 0/0 if a stratum contains only one sampling unit. If the certainty
argument specifies that this is a PSU sampled with probability 1 (a "certainty" PSU) then it does
not contribute to the variance (this is correct only when there is no subsampling within the PSU –
otherwise it should be defined as a pseudo-stratum). If certainty is FALSE for this stratum or is
not supplied the result depends on lonely.psu.

The options are "fail" to give an error, "remove" or "certainty" to give a variance contribution
of 0 for the stratum, "adjust" to center the stratum at the grand mean rather than the stratum mean,
and "average" to assign strata with one PSU the average variance contribution from strata with
more than one PSU. The choice is controlled by setting options(survey.lonely.psu). If this is
not done the factory default is "fail". Using "adjust" is conservative, and it would often be better
to combine strata in some intelligent way. The properties of "average" have not been investigated
thoroughly, but it may be useful when the lonely PSUs are due to a few strata having PSUs missing
completely at random.

The "remove" and "certainty" options give the same result, but "certainty" is intended for
situations where there is only one PSU in the population stratum, which is sampled with certainty
(also called ‘self-representing’ PSUs or strata). With "certainty" no warning is generated for
strata with only one PSU. Ordinarily, svydesign will detect certainty PSUs, making this option
unnecessary.

For strata with a single PSU in a subset (domain) the variance formula gives a value that is well-
defined and positive, but not typically correct. If options("survey.adjust.domain.lonely") is
TRUE and options("survey.lonely.psu") is "adjust" or "average", and no post-stratification
or G-calibration has been done, strata with a single PSU in a subset will be treated like those with a
single PSU in the sample. I am not aware of any theoretical study of this procedure, but it should at
least be conservative.

Value

A covariance matrix
svycralpha

Author(s)
Thomas Lumley

References

See Also
svydesign, svyrecvar, surveyoptions, postStratify

dsycralpha | Cronbach's alpha

Description
Compute Cronbach’s alpha coefficient of reliability from survey data. The formula is equation (2) of Cronbach (1951) only with design-based estimates of the variances.

Usage
svycralpha(formula, design, na.rm = FALSE)

Arguments
formula | One-sided formula giving the variables that make up the total score
design | survey design object
na.rm | TRUE to remove missing values

Value
A number

References

Examples
data(api)
dstrat<-svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat, fpc = ~fpc)
svycralpha(~ell+mobility+avg.ed+emer+meals, dstrat)
svydesign

Survey sample analysis.

Description

Specify a complex survey design.

Usage

svydesign(ids, probs=NULL, strata = NULL, variables = NULL, fpc=NULL, data = NULL, nest = FALSE, check.strata = !nest, weights=NULL, pps=FALSE,...)

## Default S3 method:
svydesign(ids, probs=NULL, strata = NULL, variables = NULL, fpc=NULL, data = NULL, nest = FALSE, check.strata = !nest, weights=NULL, pps=FALSE, calibrate.formula=NULL, variance=c("HT","YG"),...)

## S3 method for class 'imputationList'
svydesign(ids, probs = NULL, strata = NULL, variables = NULL, fpc = NULL, data, nest = FALSE, check.strata = !nest, weights = NULL, pps=FALSE, calibrate.formula=NULL,...)

## S3 method for class 'character'
svydesign(ids, probs = NULL, strata = NULL, variables = NULL, fpc = NULL, data, nest = FALSE, check.strata = !nest, weights = NULL, pps=FALSE, calibrate.formula=NULL, dbtype = "SQLite", dbname, ...)

Arguments

ids Formula or data frame specifying cluster ids from largest level to smallest level, ~0 or ~1 is a formula for no clusters.

probs Formula or data frame specifying cluster sampling probabilities

strata Formula or vector specifying strata, use NULL for no strata

variables Formula or data frame specifying the variables measured in the survey. If NULL, the data argument is used.

fpc Finite population correction: see Details below

weights Formula or vector specifying sampling weights as an alternative to prob

data Data frame to look up variables in the formula arguments, or database table name, or imputationList object, see below

nest If TRUE, relabel cluster ids to enforce nesting within strata

check.strata If TRUE, check that clusters are nested in strata

pps "brewer" to use Brewer's approximation for PPS sampling without replacement. "overton" to use Overton's approximation. An object of class HR to use the Hartley-Rao approximation. An object of class ppsmat to use the Horvitz-Thompson estimator.
calibrate.formula
model formula specifying how the weights are *already* calibrated (raked, post-stratified).
dbtype
name of database driver to pass to dbDriver
dbname
name of database (e.g. file name for SQLite)
variance
For pps without replacement, use variance="YG" for the Yates-Grundy estimator instead of the Horvitz-Thompson estimator
...
for future expansion

Details

The svydesign object combines a data frame and all the survey design information needed to analyse it. These objects are used by the survey modelling and summary functions. The id argument is always required, the strata, fpc, weights and probs arguments are optional. If these variables are specified they must not have any missing values.

By default, svydesign assumes that all PSUs, even those in different strata, have a unique value of the id variable. This allows some data errors to be detected. If your PSUs reuse the same identifiers across strata then set nest=TRUE.

The finite population correction (fpc) is used to reduce the variance when a substantial fraction of the total population of interest has been sampled. It may not be appropriate if the target of inference is the process generating the data rather than the statistics of a particular finite population.

The finite population correction can be specified either as the total population size in each stratum or as the fraction of the total population that has been sampled. In either case the relevant population size is the sampling units. That is, sampling 100 units from a population stratum of size 500 can be specified as 500 or as 100/500=0.2. The exception is for PPS sampling without replacement, where the sampling probability (which will be different for each PSU) must be used.

If population sizes are specified but not sampling probabilities or weights, the sampling probabilities will be computed from the population sizes assuming simple random sampling within strata.

For multistage sampling the id argument should specify a formula with the cluster identifiers at each stage. If subsequent stages are stratified strata should also be specified as a formula with stratum identifiers at each stage. The population size for each level of sampling should also be specified in fpc. If fpc is not specified then sampling is assumed to be with replacement at the top level and only the first stage of cluster is used in computing variances. If fpc is specified but for fewer stages than id, sampling is assumed to be complete for subsequent stages. The variance calculations for multistage sampling assume simple or stratified random sampling within clusters at each stage except possibly the last.

For PPS sampling without replacement it is necessary to specify the probabilities for each stage of sampling using the fpc arguments, and an overall weight argument should not be given. At the moment, multistage or stratified PPS sampling without replacement is supported only with pps="brewer", or by giving the full joint probability matrix using ppsmat. [Cluster sampling is supported by all methods, but not subsampling within clusters].

The dim, "[", "<-" and na.action methods for survey.design objects operate on the data frame specified by variables and ensure that the design information is properly updated to correspond to the new data frame. With the "<-" method the new value can be a survey.design object instead of a data frame, but only the data frame is used. See also subset.survey.design for a simple way to select subpopulations.
The model.frame method extracts the observed data.

If the strata with only one PSU are not self-representing (or they are, but svydesign cannot tell based on fpc) then the handling of these strata for variance computation is determined by options("survey.lonely.psu"). See svyCprod for details.

data may be a character string giving the name of a table or view in a relational database that can be accessed through the DBI interfaces. For DBI interfaces dbtype should be the name of the database driver and dbname should be the name by which the driver identifies the specific database (e.g. file name for SQLite).

The appropriate database interface package must already be loaded (e.g. RSQLite for SQLite). The survey design object will contain only the design meta-data, and actual variables will be loaded from the database as needed. Use close to close the database connection and open to reopen the connection, e.g., after loading a saved object.

The database interface does not attempt to modify the underlying database and so can be used with read-only permissions on the database.

If data is an imputationList object (from the "mitools" package), svydesign will return a svyimputationList object containing a set of designs. Use with.svyimputationList to do analyses on these designs and MIcombine to combine the results.

Value

An object of class survey.design.

Author(s)

Thomas Lumley

See Also

as.svrepdesign for converting to replicate weight designs, subset.survey.design for domain estimates, update.survey.design to add variables.

mitools package for using multiple imputations

svyrecvar for details of variance estimation

election for examples of PPS sampling without replacement.

Examples

data(api)
# stratified sample
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
# one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
# two-stage cluster sample: weights computed from population sizes.
dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)

## multistage sampling has no effect when fpc is not given, so
## these are equivalent.
dclus2wr<-svydesign(id=~dnum+snum, weights=weights(dclus2), data=apiclus2)
dclus2wr2<-svydesign(id=~dnum, weights=weights(dclus2), data=apiclus2)
svyfactanal

Factor analysis in complex surveys (experimental).

Description

This function fits a factor analysis model or SEM, by maximum weighted likelihood.

Usage

svyfactanal(formula, design, factors,
            n = c("none", "sample", "degf", "effective", "min.effective"), ...)

Arguments

  formula       Model formula specifying the variables to use
  design        Survey design object
  factors       Number of factors to estimate
  n             Sample size to be used for testing: see below
  ...           Other arguments to pass to factanal.
Details

The population covariance matrix is estimated by \texttt{svyvar} and passed to \texttt{factanal}.

Although fitting these models requires only the estimated covariance matrix, inference requires a sample size. With \texttt{n="sample"}, the sample size is taken to be the number of observations; with \texttt{n="degf"}, the survey degrees of freedom as returned by \texttt{degf}. Using "sample" corresponds to standardizing weights to have mean 1, and is known to result in anti-conservative tests.

The other two methods estimate an effective sample size for each variable as the sample size where the standard error of a variance of a Normal distribution would match the design-based standard error estimated by \texttt{svyvar}. With \texttt{n="min.effective"} the minimum sample size across the variables is used; with \texttt{n="effective"} the harmonic mean is used. For \texttt{svyfactanal} the test of model adequacy is optional, and the default choice, \texttt{n="none"}, does not do the test.

Value

An object of class \texttt{factanal}

References

See Also

\texttt{factanal}

The \texttt{lavaan.survey} package fits structural equation models to complex samples using similar techniques.

Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

svyfactanal(~api99+api00+hsg+meals+ell+emer, design=dclus1, factors=2)

svyfactanal(~api99+api00+hsg+meals+ell+emer, design=dclus1, factors=2, n="effective")

# Population data for comparison
factanal(~api99+api00+hsg+meals+ell+emer, data=apirop, factors=2)
Usage

```r
## S3 method for class 'survey.design'
svyglm(formula, design, subset=NULL,
       family=stats::gaussian(), start=NULL, rescale=TRUE, ..., deff=FALSE,
       influence=FALSE)
## S3 method for class 'svyrep.design'
svyglm(formula, design, subset=NULL,
       family=stats::gaussian(), start=NULL, rescale=FALSE, ..., rho=NULL,
       return.replicates=FALSE, na.action, multicore=getOption("survey.multicore"))
## S3 method for class 'svyglm'
summary(object, correlation = FALSE, df.resid=NULL,
        ...)  
## S3 method for class 'svyglm'
predict(object, newdata=NULL, total=NULL,
       type=c("link","response","terms"),
       se.fit=(type != "terms"), vcov=FALSE,...)
## S3 method for class 'svrepglm'
predict(object, newdata=NULL, total=NULL,
       type=c("link","response","terms"),
       se.fit=(type != "terms"), vcov=FALSE,
       return.replicates=!is.null(object$replicates),...)
```

Arguments

- `formula`: Model formula
- `design`: Survey design from `svydesign` or `svrepdesign`. Must contain all variables in the formula
- `subset`: Expression to select a subpopulation
- `family`: Family object for `glm`
- `start`: Starting values for the coefficients (needed for some uncommon link/family combinations)
- `rescale`: Rescaling of weights, to improve numerical stability. The default rescales weights to sum to the sample size. Use `FALSE` to not rescale weights. For replicate-weight designs, use `TRUE` to rescale weights to sum to 1, as was the case before version 3.34.
- `...`: Other arguments passed to `glm` or `summary.glm`
- `rho`: For replicate BRR designs, to specify the parameter for Fay’s variance method, giving weights of `rho` and `2-rho`
- `return.replicates`: Return the replicates as the replicates component of the result? (for `predict`, only possible if they were computed in the `svyglm` fit)
- `deff`: Estimate the design effects
- `influence`: Return influence functions
- `object`: A `svyglm` object
correlation  Include the correlation matrix of parameters?
n.a.action  Handling of NAs
multicore  Use the multicore package to distribute replicates across processors?
df.resid  Optional denominator degrees of freedom for Wald tests
newdata  new data frame for prediction
total  population size when predicting population total
type  linear predictor (link) or response
se.fit  if TRUE, return variances of predictions
vcov  if TRUE and se=TRUE return full variance-covariance matrix of predictions

Details

For binomial and Poisson families use family=quasibinomial() and family=quasipoisson() to avoid a warning about non-integer numbers of successes. The ‘quasi’ versions of the family objects give the same point estimates and standard errors and do not give the warning.

If df.resid is not specified the df for the null model is computed by degf and the residual df computed by subtraction. This is recommended by Korn and Graubard and is correct for PSU-level covariates but is potentially very conservative for individual-level covariates. To get tests based on a Normal distribution use df.resid=Inf, and to use number of PSUs-number of strata, specify df.resid=degf(design).

Parallel processing with multicore=TRUE is helpful only for fairly large data sets and on computers with sufficient memory. It may be incompatible with GUIs, although the Mac Aqua GUI appears to be safe.

predict gives fitted values and sampling variability for specific new values of covariates. When newdata are the population mean it gives the regression estimator of the mean, and when newdata are the population totals and total is specified it gives the regression estimator of the population total. Regression estimators of mean and total can also be obtained with calibrate.

Value

svyglm returns an object of class svyglm. The predict method returns an object of class svystat

Note

svyglm always returns ’model-robust’ standard errors; the Horvitz-Thompson-type standard errors used everywhere in the survey package are a generalisation of the model-robust 'sandwich' estimators. In particular, a quasi-Poisson svyglm will return correct standard errors for relative risk regression models.

Note

This function does not return the same standard error estimates for the regression estimator of population mean and total as some textbooks, or SAS. However, it does give the same standard error estimator as estimating the mean or total with calibrated weights.

In particular, under simple random sampling with or without replacement there is a simple rescaling of the mean squared residual to estimate the mean squared error of the regression estimator.
The standard error estimate produced by `predict.svyglm` has very similar (asymptotically identical) expected value to the textbook estimate, and has the advantage of being applicable when the supplied `newdata` are not the population mean of the predictors. The difference is small when the sample size is large, but can be appreciable for small samples.

You can obtain the other standard error estimator by calling `predict.svyglm` with the covariates set to their estimated (rather than true) population mean values.

**Author(s)**

Thomas Lumley

**References**


**See Also**

glm, which is used to do most of the work.

regTermTest, for multiparameter tests

calibrate, for an alternative way to specify regression estimators of population totals or means

svyttest for one-sample and two-sample t-tests.

**Examples**

data(api)

dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
dclus2<-svydesign(id=~dnum+snum, weights=~pw, data=apiclus2)
rstrat<-as.svrepdesign(dstrat)
rclus2<-as.svrepdesign(dclus2)

summary(svyglm(api00~ell+meals+mobility, design=dstrat))
summary(svyglm(api00~ell+meals+mobility, design=dclus2))
summary(svyglm(api00~ell+meals+mobility, design=rstrat))
summary(svyglm(api00~ell+meals+mobility, design=rclus2))

## use quasibinomial, quasipoisson to avoid warning messages
summary(svyglm(sch.wide~ell+meals+mobility, design=dstrat, family=quasibinomial()))

## Compare regression and ratio estimation of totals
api.ratio <- svyratio(~api.stu,~enroll, design=dstrat)
pop<-data.frame(enroll=sum(apiipop$enroll, na.rm=TRUE))
npop <- nrow(apiipop)
predict(api.ratio, pop$enroll)
## regression estimator is less efficient

api.reg <- svyglm(api.stu~enroll, design=dstrat)
predict(api.reg, newdata=pop, total=npop)

## same as calibration estimator

svytotal(~api.stu, calibrate(dstrat, ~enroll, pop=c(npop, pop$enroll)))

## svyglm can also reproduce the ratio estimator

api.reg2 <- svyglm(api.stu~enroll-1, design=dstrat,
family=quasi(link="identity",var="mu"))
predict(api.reg2, newdata=pop, total=npop)

## higher efficiency by modelling variance better

api.reg3 <- svyglm(api.stu~enroll-1, design=dstrat,
family=quasi(link="identity",var="mu^3"))
predict(api.reg3, newdata=pop, total=npop)

## true value

sum(apipop$api.stu)

---

svygofchisq

### Test of fit to known probabilities

#### Description

A Rao-Scott-type version of the chi-squared test for goodness of fit to prespecified proportions. The test statistic is the chi-squared statistic applied to the estimated population table, and the reference distribution is a Satterthwaite approximation: the test statistic divided by the estimated scale is compared to a chi-squared distribution with the estimated df.

#### Usage

svygofchisq(formula, p, design, ...)

#### Arguments

- **formula**: Formula specifying a single factor variable
- **p**: Vector of probabilities for the categories of the factor, in the correct order (will be rescaled to sum to 1)
- **design**: Survey design object
- **...**: Other arguments to pass to `svytotal`, such as `na.rm`

#### Value

An object of class `htest`

#### See Also

[chisq.test, svychisq, pchisqsum]
Examples

```r
data(api)
dclus2 <- svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
true_p <- table(apipop$stype)
svygofchisq(~stype,dclus2,p=true_p)
svygofchisq(~stype,dclus2,p=c(1/3,1/3,1/3))
```

svyhist

Histograms and boxplots

Description

Histograms and boxplots weighted by the sampling weights.

Usage

```r
svyhist(formula, design, breaks = "Sturges",
include.lowest = TRUE, right = TRUE, xlab = NULL,
main = NULL, probability = TRUE, freq = !probability, ...)
svyboxplot(formula, design, all.outliers=FALSE,...)
```

Arguments

- **formula**: One-sided formula for `svyhist`, two-sided for `svyboxplot`
- **design**: A survey design object
- **xlab**: x-axis label
- **main**: Main title
- **probability**, **freq**: Y-axis is probability density or frequency
- **all.outliers**: Show all outliers in the boxplot, not just extremes
- **breaks, include.lowest, right**: As for `hist`
- **...**: Other arguments to `hist` or `bxp`

Details

The histogram breakpoints are computed as if the sample were a simple random sample of the same size.

The grouping variable in `svyboxplot`, if present, must be a factor.

The boxplot whiskers go to the maximum and minimum observations or to 1.5 interquartile ranges beyond the end of the box, whichever is closer. The maximum and minimum are plotted as outliers if they are beyond the ends of the whiskers, but other outlying points are not plotted unless `all.outliers=TRUE`. `svyboxplot` requires a two-sided formula; use `variable~1` for a single boxplot.
svyivreg

Two-stage least-squares for instrumental variable regression

Description

Estimates regressions with endogenous covariates using two-stage least squares. The function uses ivreg from the AER package for the main computations, and follows the syntax of that function.

Usage

svyivreg(formula, design, ...)

Arguments

- formula: formula specification(s) of the regression relationship and the instruments. See Details for details
- design: A survey design object
- ...: For future expansion
svykappa

Details

Regressors and instruments for svyivreg are specified in a formula with two parts on the right-hand side, e.g., $y \sim x_1 + x_2 | z_1 + z_2 + z_3$, where $x_1$ and $x_2$ are the regressors and $z_1$, $z_2$, and $z_3$ are the instruments. Note that exogenous regressors have to be included as instruments for themselves. For example, if there is one exogenous regressor $ex$ and one endogenous regressor $en$ with instrument $in$, the appropriate formula would be $y \sim ex + en | ex + in$. Equivalently, this can be specified as $y \sim ex + en | . - en + in$, i.e., by providing an update formula with a . in the second part of the formula.

Value

An object of class svyivreg

References


See Also

ivreg

svykappa

Cohen’s kappa for agreement

Description

Computes the unweighted kappa measure of agreement between two raters and the standard error. The measurements must both be factor variables in the survey design object.

Usage

svykappa(formula, design, ...)

Arguments

formula one-sided formula giving two measurements
design survey design object
... passed to svymean internally (such as return.replicates or influence)

Value

Object of class svystat

See Also

svycontrast
Examples

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
svykappa(~comp.imp+sch.wide, dclus1)

dclus1<-update(dclus1, stypecopy=stype)
svykappa(~stype+stypecopy, dclus1)

(kappas<-svyby(~comp.imp+sch.wide,~stype,design=dclus1, svykappa, covmat=TRUE))
svycontrast(kappas, quote(E/H))

svykm

Estimate survival function.

Description
Estimates the survival function using a weighted Kaplan-Meier estimator.

Usage

svykm(formula, design, se=FALSE, ...)  
## S3 method for class 'svykm'
plot(x, xlab="time", ylab="Proportion surviving",  
     ylim=c(0,1), ci=NULL, lty=1,...)  
## S3 method for class 'svykm'
lines(x, xlab="time", type="s", ci=FALSE, lty=1,...)  
## S3 method for class 'svykmlist'
plot(x, pars=NULL, ci=FALSE,...)  
## S3 method for class 'svykm'
quantile(x, probs=c(0.75,0.5,0.25), ci=FALSE, level=0.95,...)  
## S3 method for class 'svykm'
confint(object, parm, level=0.95,...)

Arguments

formula Two-sided formula. The response variable should be a right-censored Surv object
design survey design object
se Compute standard errors? This is slow for moderate to large data sets
... in plot and lines methods, graphical parameters
x a svykm or svykmlist object
xlab,ylab,ylim,type as for plot
lty Line type, see par
ci | Plot (or return, for quantile) the confidence interval
pars | A list of vectors of graphical parameters for the separate curves in a svykmlist object
object | A svykmlist object
parm | vector of times to report confidence intervals
level | confidence level
probs | survival probabilities for computing survival quantiles (note that these are the complement of the usual quantile input, so 0.9 means 90% surviving, not 90% dead)

Details
When standard errors are computed, the survival curve is actually the Aalen (hazard-based) estimator rather than the Kaplan-Meier estimator.
The standard error computations use memory proportional to the sample size times the square of the number of events. This can be a lot.
In the case of equal-probability cluster sampling without replacement the computations are essentially the same as those of Williams (1995), and the same linearization strategy is used for other designs.
Confidence intervals are computed on the log(survival) scale, following the default in survival package, which was based on simulations by Link(1984).
Confidence intervals for quantiles use Woodruff’s method: the interval is the intersection of the horizontal line at the specified quantile with the pointwise confidence band around the survival curve.

Value
For svykmlist, an object of class svykmlist for a single curve or svykmlist for multiple curves.

References
Woodruff RS (1952) Confidence intervals for medians and other position measures. JASA 57, 622-627.

See Also
predict.svycoxph for survival curves from a Cox model
Examples

```r
data(pbc, package="survival")
pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
biasmodel<-glm(randomized~age*edema,data=pbc)
pbc$randprob<-fitted(biasmodel)

dpbc<-svydesign(id=~1, prob=~randprob, strata=~edema, data=subset(pbc,randomized))

s1<-svykm(Surv(time,status>0)~1, design=dpbc)
s2<-svykm(Surv(time,status>0)~I(bili>6), design=dpbc)

plot(s1)
plot(s2)
plot(s2, lwd=2, pars=list(lty=c(1,2),col=c("purple","forestgreen")))

quantile(s1, probs=c(0.9,0.75,0.5,0.25,0.1))

s3<-svykm(Surv(time,status>0)~I(bili>6), design=dpbc, se=TRUE)
plot(s3[[2]],col="purple")

confint(s3[[2]], parm=365*(1:5))
quantile(s3[[1]], ci=TRUE)
```

svyloglin

Loglinear models

Description

Fit and compare hierarchical loglinear models for complex survey data.

Usage

```r
svyloglin(formula, design, ...)
## S3 method for class 'svyloglin'
update(object, formula, ...)
## S3 method for class 'svyloglin'
anova(object, object1, ..., integrate=FALSE)
## S3 method for class 'anova.svyloglin'
print(x, pval=c("F","saddlepoint","lincom","chisq"), ...)
## S3 method for class 'svyloglin'
coef(object, ..., intercept=FALSE)
```

Arguments

- `formula` Model formula
- `design` survey design object
- `object, object1` loglinear model from svyloglin
pval p-value approximation: see Details
integrate Compute the exact asymptotic p-value (slow)?
... not used
intercept Report the intercept?
x anova object

Details
The loglinear model is fitted to a multiway table with probabilities estimated by svymean and with the sample size equal to the observed sample size, treating the resulting table as if it came from iid multinomial sampling, as described by Rao and Scott. The variance-covariance matrix does not include the intercept term, and so by default neither does the coef method. A Newton-Raphson algorithm is used, rather than iterative proportional fitting, so starting values are not needed.

The anova method computes the quantities that would be the score (Pearson) and likelihood ratio chi-squared statistics if the data were an iid sample. It computes four p-values for each of these, based on the exact asymptotic distribution (see pchisqsum), a saddlepoint approximation to this distribution, a scaled chi-squared distribution, and a scaled F-distribution. When testing the two-way interaction model against the main-effects model in a two-way table the score statistic and p-values match the Rao-Scott tests computed by svychisq.

The anova method can only compare two models if they are for exactly the same multiway table (same variables and same order). The update method will help with this. It is also much faster to use update than svyloglin for a large data set: its time complexity depends only on the size of the model, not on the size of the data set.

It is not possible to fit a model using a variable created inline, eg I(x<10), since the multiway table is based on all variables used in the formula.

Value
Object of class "svyloglin"

References

See Also
svychisq, svyglm, pchisqsum

Examples
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
a<-svyloglin(~stype+comp.imp,dclus1)
b<-update(a,~.^2)
an<-anova(a,b)
an
print(an, pval="saddlepoint")
svylogrank

## Wald test
regTermTest(b, ~stype:comp.imp)

## linear-by-linear association
d<-update(a,~.+as.numeric(stype):as.numeric(comp.imp))
an1<-anova(a,d)
an1

svylogrank  

### Description

Computes a weighted version of the logrank test for comparing two or more survival distributions. The generalization to complex samples is based on the characterization of the logrank test as the score test in a Cox model. Under simple random sampling with replacement, this function with rho=0 and gamma=0 is almost identical to the robust score test in the survival package. The rho=0 and gamma=0 version was proposed by Rader (2014).

#### Usage

svylogrank(formula, design, rho=0, gamma=0, method=c("small", "large", "score"), ...)

#### Arguments

- **formula**: Model formula with a single predictor. The predictor must be a factor if it has more than two levels.
- **design**: A survey design object
- **rho, gamma**: Coefficients for the Harrington/Fleming G-rho-gamma tests. The default is the logrank test, rho=1 gives a generalised Wilcoxon test
- **method**: "small" works faster when a matrix with dimension number of events by number of people fits easily in memory; "large" works faster for large data sets; "score" works by brute-force construction of an expanded data set, and is for debugging
- **...**: for future expansion.

#### Value

A vector containing the z-statistic for comparing each level of the variable to the lowest, the chisquared statistic for the logrank test, and the p-value.

#### References

**svymle**

*Maximum pseudolikelihood estimation in complex surveys*

**See Also**

svykm, svycoxph.

**Examples**

```r
library("survival")
data(nwtco)
## stratified on case status
dcchs<-twophase(id=list(~seqno,-seqno), strata=list(NULL,-rel),
               subset=~I(in.subcohort | rel), data=nwtco, method="simple")
svylogrank(Surv(edrel,rel)~factor(stage),design=dcchs)

data(pbc, package="survival")
pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
biasmodel<-glm(randomized~age*edema,data=pbc)
pbc$randprob<-fitted(biasmodel)
dpbc<-svydesign(id=~1, prob=~randprob, strata=~edema, data=subset(pbc,randomized))
svylogrank(Surv(time,status==2)-trt,design=dpbc)
svylogrank(Surv(time,status==2)-trt,design=dpbc,rho=1)
rpbc<-as.svrepdesign(dpbc)
svylogrank(Surv(time,status==2)-trt,design=rpbc)
```

**Description**

Maximises a user-specified likelihood parametrised by multiple linear predictors to data from a complex sample survey and computes the sandwich variance estimator of the coefficients. Note that this function maximises an estimated population likelihood, it is not the sample MLE.

**Usage**

```r
svymle(loglike, gradient = NULL, design, formulas, start = NULL, control
       = list(), na.action="na.fail", method=NULL, lower=NULL, upper=NULL,influence=FALSE,...)
```

**Arguments**

- `loglike` : vectorised loglikelihood function
- `gradient` : Derivative of loglike. Required for variance computation and helpful for fitting
- `design` : a survey.design object
formulas A list of formulas specifying the variable and linear predictors: see Details below
start Starting values for parameters
control control options for the optimiser: see the help page for the optimiser you are using.
lower,upper Parameter bounds for bobyqa
influence Return the influence functions (primarily for svyby)
na.action Handling of NAs
method "nlm" to use nlm, "uobyqa" or "bobyqa" to use those optimisers from the minqa package; otherwise passed to optim
... Arguments to loglike and gradient that are not to be optimised over.
object svymle object
stderr Choice of standard error estimator. The default is a standard sandwich estimator. See Details below.

Details
Optimization is done by nlm by default or if method="nlm". Otherwise optim is used and method specifies the method and control specifies control parameters.

The design object contains all the data and design information from the survey, so all the formulas refer to variables in this object. The formulas argument needs to specify the response variable and a linear predictor for each freely varying argument of loglike.

Consider for example the dnorm function, with arguments x, mean, sd and log, and suppose we want to estimate the mean of y as a linear function of a variable z, and to estimate a constant standard deviation. The log argument must be fixed at FALSE to get the loglikelihood. A formulas argument would be list(~y, mean=~z, sd=~1). Note that the data variable y must be the first argument to dnorm and the first formula and that all the other formulas are labelled. It is also permitted to have the data variable as the left-hand side of one of the formulas: eg list( mean=y~z, sd=~1).

The two optimisers from the minqa package do not use any derivatives to be specified for optimisation, but they do assume that the function is smooth enough for a quadratic approximation, ie, that two derivatives exist.

The usual variance estimator for MLEs in a survey sample is a ‘sandwich’ variance that requires the score vector and the information matrix. It requires only sampling assumptions to be valid (though some model assumptions are required for it to be useful). This is the stderr="robust" option, which is available only when the gradient argument was specified.

If the model is correctly specified and the sampling is at random conditional on variables in the model then standard errors based on just the information matrix will be approximately valid. In particular, for independent sampling where weights and strata depend on variables in the model the stderr="model" should work fairly well.

Value
An object of class svymle
Author(s)

Thomas Lumley

See Also

svydesign, svyglm

Examples

data(api)

dstrat<-svydesign(id=~1, strata=~stype, weight=~pw, fpc=~fpc, data=apistrat)

## fit with glm
m0 <- svyglm(api00~api99+ell,family="gaussian",design=dstrat)
## fit as mle (without gradient)
m1 <- svymle(loglike=dnorm,gradient=NULL, design=dstrat,
  formulas=list(mean=api00~api99+ell, sd=~1),
  start=list(c(80,1,0),c(20)), log=TRUE)
## with gradient
gr<- function(x,mean,sd,log){
  dm<-2*(x - mean)/(2*sd^2)
  ds<-(x-mean)^2*(2*(2 * sd)^2)/(2*sd^2)^2 - sqrt(2*pi)/(sd*sqrt(2*pi))
  cbind(dm,ds)
}
m2 <- svymle(loglike=dnorm,gradient=gr, design=dstrat,
  formulas=list(mean=api00~api99+ell, sd=~1),
  start=list(c(80,1,0),c(20)), log=TRUE, method="BFGS")

summary(m0)
summary(m1,stderr="model")
summary(m2)

## Using offsets
m3 <- svymle(loglike=dnorm,gradient=gr, design=dstrat,
  formulas=list(mean=api00~api99+offset(ell)+ell, sd=~1),
  start=list(c(80,1,0),c(20)), log=TRUE, method="BFGS")

## demonstrating multiple linear predictors
m3 <- svymle(loglike=dnorm,gradient=gr, design=dstrat,
  formulas=list(mean=api00~api99+offset(ell)+ell, sd=stype),
  start=list(c(80,1,0),c(20,0,0)), log=TRUE, method="BFGS")

## More complicated censored lognormal data example
## showing that the response variable can be multivariate

data(pbc, package="survival")
```r
# Example code for svymle

pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
biasmodel <- glm(randomized~age*edema, data=pbc)
pbc$randprob <- fitted(biasmodel)
dpbc <- svydesign(id=~1, prob=~randprob, strata=~edema, data=subset(pbc, randomized))

## Censored lognormal likelihood
lcens <- function(x, mean, sd){
  ifelse(x[,2]==1,
    dnorm(log(x[,1]), mean, sd, log=TRUE),
    pnorm(log(x[,1]), mean, sd, log=TRUE, lower.tail=FALSE)
  )
}

gcens <- function(x, mean, sd){
  dz <- -dnorm(log(x[,1]), mean, sd) / pnorm(log(x[,1]), mean, sd, lower.tail=FALSE)
  dm <- ifelse(x[,2]==1,
    2*(log(x[,1]) - mean)/(2*sd^2),
    dz*-1/sd
  )
  ds <- ifelse(x[,2]==1,
    (log(x[,1])-mean)^2*(2*(2*sd^2)) / (2*sd^2)^2 - sqrt(2*pi)/(sd*sqrt(2*pi)),
    dz*-(log(x[,1])-mean)/(sd*sd)
  )
  cbind(dm, ds)
}

m <- svymle(loglike=lcens, gradient=gcens, design=dpbc, method="newuoa",
            formulas=list(mean=I(cbind(time,status>0))~bili+protime+albumin,
                          sd=~1),
            start=list(c(10,0,0,0),c(1)))

summary(m)

## The same model, but now specifying the lower bound of zero on the
## log standard deviation

mbox <- svymle(loglike=lcens, gradient=gcens, design=dpbc, method="bobyqa",
               formulas=list(mean=I(cbind(time,status>0))~bili+protime+albumin,
                             sd=~1),
               lower=list(c(-Inf,-Inf,-Inf,-Inf),0), upper=Inf,
               start=list(c(10,0,0,0),c(1)))

## The censored lognormal model is now available in svysurvreg()

summary(svysurvreg(Surv(time,status>0)~bili+protime+albumin,
                    design=dpbc, dist="lognormal"))

## Compare svymle scale value after log transformation
svycontrast(m, quote(log(`sd.((Intercept)`))))
```
svynls  

Probability-weighted nonlinear least squares

Description

Fits a nonlinear model by probability-weighted least squares. Uses nls to do the fitting, but estimates design-based standard errors with either linearisation or replicate weights. See nls for documentation of model specification and fitting.

Usage

svynls(formula, design, start, weights=NULL, ...)

Arguments

- **formula**: Nonlinear model specified as a formula; see nls
- **design**: Survey design object
- **start**: starting values, passed to nls
- **weights**: Non-sampling weights, eg precision weights to give more efficient estimation in the presence of heteroscedasticity.
- **...**: Other arguments to nls (especially, start). Also supports return.replicates for replicate-weight designs and influence for other designs.

Value

Object of class svynls. The fitted nls object is included as the fit element.

See Also

svymle for maximum likelihood with linear predictors on one or more parameters

Examples

```r
set.seed(2020-4-3)
x<-rep(seq(0,50,1),10)
y<-((runif(1,10,20)*x)/(runif(1,0,10)+x))+rnorm(510,0,1)
pop_model<-nls(y~a*x/(b+x), start=c(a=15,b=5))
df<-data.frame(x=x,y=y)
df$p<-ifelse((y-fitted(pop_model))*(x-mean(x))>0, .4,.1)
df$strata<-ifelse(df$p==.4,"a","b")
in_sample<-stratsample(df$strata, round(table(df$strat)*c(0.4,0.1)))
```
svyolr

```
sdf<-df[in_sample,]
des<-svydesign(id=~1, strata=~strata, prob=~p, data=sdf)
pop_model
(biased_sample<-nls(y~a*x/(b+x),data=sdf, start=c(a=15,b=5)))
(corrected <- svynls(y~a*x/(b+x), design=des, start=c(a=15,b=5)))
```

---

**svyolr**

Proportional odds and related models

**Description**

Fits cumulative link models: proportional odds, probit, complementary log-log, and cauchit.

**Usage**

```
svyolr(formula, design, ...)
## S3 method for class 'survey.design2'
svyolr(formula, design, start, subset=NULL, ...,
   na.action = na.omit, method = c("logistic", "probit", "cloglog", "cauchit"))
## S3 method for class 'svyrep.design'
svyolr(formula, design, subset=NULL, ...,
   return.replicates=FALSE,
   multicore=getOption("survey.multicore"))
## S3 method for class 'svyolr'
predict(object, newdata, type = c("class", "probs"), ...)
```

**Arguments**

- **formula**: Formula: the response must be a factor with at least three levels
- **design**: survey design object
- **subset**: subset of the design to use; NULL for all of it
- **...**: dots
- **start**: Optional starting values for optimization
- **na.action**: handling of missing values
- **multicore**: Use multicore package to distribute computation of replicates across multiple processors?
- **method**: Link function
- **return.replicates**: return the individual replicate-weight estimates
- **object**: object of class svyolr
- **newdata**: new data for predictions
- **type**: return vector of most likely class or matrix of probabilities

**Value**

An object of class svyolr
svyplot

Plots for survey data

Description

Because observations in survey samples may represent very different numbers of units in the population ordinary plots can be misleading. The svyplot function produces scatterplots adjusted in various ways for sampling weights.

Usage

svyplot(formula, design,...)

## Default S3 method:

svyplot(formula, design, style = c("bubble", "hex", "grayhex","subsample","transparent"),
sample.size = 500, subset = NULL, legend = 1, inches = 0.05,
amount=NULL, basecol="black",
alpha=c(0, 0.8),xbins=30,...)

Arguments

formula A model formula
design A survey object (svydesign or svrepdesign)
style See Details below
sample.size For style="subsample"
svyplot

subset expression using variables in the design object
legend For style="hex" or "grayhex"
inches Scale for bubble plots
amount list with x and y components for amount of jittering to use in subsample plots, or NULL for the default amount
basecol base color for transparent plots, or a function to compute the color (see below), or color for bubble plots
alpha minimum and maximum opacity for transparent plots
xbins Number of (x-axis) bins for hexagonal binning
... Passed to plot methods

Details

Bubble plots are scatterplots with circles whose area is proportional to the sampling weight. The two "hex" styles produce hexagonal binning scatterplots, and require the hexbin package from Bioconductor. The "transparent" style plots points with opacity proportional to sampling weight.

The subsample method uses the sampling weights to create a sample from approximately the population distribution and passes this to plot

Bubble plots are suited to small surveys, hexagonal binning and transparency to large surveys where plotting all the points would result in too much overlap.

basecol can be a function taking one data frame argument, which will be passed the data frame of variables from the survey object. This could be memory-intensive for large data sets.

Value

None

References


See Also

symbols for other options (such as colour) for bubble plots.
svyttable for plots of discrete data.

Examples

data(api)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)

svyplot(api00~api99, design=dstrat, style="bubble")
svyplot(api00~api99, design=dstrat, style="transparent",pch=19)
svyprcomp

### svyprcomp

SVYPRCOMP is a function from the R package "survey".

#### Description

Computes principal components using the sampling weights.

#### Usage

```r
svyprcomp(formula, design, center = TRUE, scale. = FALSE, tol = NULL, scores = FALSE, ...)
```

#### Arguments

- `formula`: model formula describing variables to be used.
- `design`: survey design object.
- `center`: Center data before analysis?

---

svyprcomp

Sampling-weighted principal component analysis

### Description

Computes principal components using the sampling weights.

### Usage

```r
svyprcomp(formula, design, center = TRUE, scale. = FALSE, tol = NULL, scores = FALSE, ...)
```

### Arguments

- `formula`: model formula describing variables to be used.
- `design`: survey design object.
- `center`: Center data before analysis?
svyprcomp

scale. Scale to unit variance before analysis?
tol Tolerance for omitting components from the results; a proportion of the standard
deviation of the first component. The default is to keep all components.
scores Return scores on each component? These are needed for biplot.
x A svyprcomp object
cols Base colors for observations and variables respectively
x1abs Formula, or character vector, giving labels for each observation
weight How to display the sampling weights: "scaled" changes the size of the point
label, "transparent" uses opacity proportional to sampling weight, "none"
changes neither.
max.alpha Opacity for the largest sampling weight, or for all points if weight!="transparent"
max.cex Character size (as a multiple of par("cex")) for the largest sampling weight, or
for all points if weight!="scaled"
xlim,ylim,xlab,ylab Graphical parameters
expand,arrow.len See biplot
pc.biplot See link{biplot.prcomp}
... Other arguments to prcomp, or graphical parameters for biplot

Value

svyprcomp returns an object of class svyprcomp, similar to class prcomp but including design
information

See Also

prcomp, biplot.prcomp

Examples

data(api)
dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)

pc <- svyprcomp(~api99+api00+ell+hsg+meals+emer, design=dclus2, scale=TRUE,scores=TRUE)

pc

biplot(pc, x1abs=~dnum, weight="none")

biplot(pc, x1abs=~dnum,max.alpha=1)

biplot(pc, weight="scaled",max.cex=1.5, x1abs=~dnum)
svypredmeans  

Predictive marginal means

Description
Predictive marginal means for a generalised linear model, using the method of Korn and Graubard (1999) and matching the results of SUDAAN. The predictive marginal mean for one level of a factor is the probability-weighted average of the fitted values for the model on new data where all the observations are set to that level of the factor but have whatever values of adjustment variables they really have.

Usage
svypredmeans(adjustmodel, groupfactor, predictat=NULL)

Arguments
- **adjustmodel**: A generalised linear model fit by svyglm with the adjustment variable but without the factor for which predictive means are wanted
- **groupfactor**: A one-sided formula specifying the factor for which predictive means are wanted. Can use, eg, ~interaction(race, sex) for combining variables. This does not have to be a factor, but it will be modelled linearly if it isn’t
- **predictat**: A vector of the values of groupfactor where you want predictions. If groupfactor is a factor, these must be values in the data, but if it is numeric you can interpolate/extrapolate

Value
An object of class svystat with the predictive marginal means and their covariance matrix.

Note
It is possible to supply an adjustment model with only an intercept, but the results are then the same as svymean

It makes no sense to have a variable in the adjustment model that is part of the grouping factor, and will give an error message or NA.

References
svyqqplot

Quantile-quantile plots for survey data

Description

Quantile-quantile plots either against a specified distribution function or comparing two variables from the same or different designs.

Usage

svyqqplot(formula, design, designx = NULL, na.rm = TRUE, qrule = "hf8", xlab = NULL, ylab = NULL, ...)  
svyqqmath(x, design, null=qnorm, na.rm=TRUE, xlab="Expected",ylab="Observed",...)

Arguments

x, formula A one-sided formula for svyqqmath or a two-sided formula for svyqqplot

design Survey design object to look up variables

designx Survey design object to look up the RHS variable in svyqqplot, if different from the LHS variable

null Quantile function to compare the data quantiles to

na.rm Remove missing values

qrule How to define quantiles for svyqqplot – see svyquantile for possible values

See Also

svyglm

Worked example using National Health Interview Survey data: https://gist.github.com/tslumley/2e74cd0ac12a671d2724

Examples

data(nhanes)
nhanes_design <- svydesign(id=~SDMVPSU, strata=~SDMVSTRA, weights=~WTMEC2YR, nest=TRUE,data=nhanes)  
agesexmodel<-svyglm(HI_CHOL~agecat+RIAGENDR, design=nhanes_design,family=quasibinomial)  
## high cholesterol by race/ethnicity, adjusted for demographic differences
means<-svypredmeans(agesexmodel, -factor(race))
means
## relative risks compared to non-Hispanic white
svycontrast(means,quote(`1'/`2'))
svycontrast(means,quote(`3'/`2'))
data(api)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
demog_model <- svyglm(api00~mobility+ell+hsg+meals, design=dstrat)
svypredmeans(demog_model,~enroll, predictat=c(100,300,1000,3000))
svyranktest

Design-based rank tests

Description

Design-based versions of k-sample rank tests. The built-in tests are all for location hypotheses, but
the user could specify others.

Usage

svyranktest(formula, design,
    test = c("wilcoxon", "vanderWaerden", "median","KruskalWallis"), ...)
svyranktest

Arguments

- **formula**: Model formula `y~g` for outcome variable `y` and group `g`
- **design**: A survey design object
- **test**: Which rank test to use: Wilcoxon, van der Waerden’s normal-scores test, Mood’s test for the median, or a function `f(r,N)` where `r` is the rank and `N` the estimated population size. "KruskalWallis" is a synonym for "wilcoxon" for more than two groups.

Details

These tests are for the null hypothesis that the population or superpopulation distributions of the response variable are different between groups, targeted at population or superpopulation alternatives. The 'ranks' are defined as quantiles of the pooled distribution of the variable, so they do not just go from 1 to N; the null hypothesis does not depend on the weights, but the ranks do.

The tests reduce to the usual Normal approximations to the usual rank tests under iid sampling. Unlike the traditional rank tests, they are not exact in small samples.

Value

Object of class `htest`

Note that with more than two groups the statistic element of the return value holds the numerator degrees of freedom and the parameter element holds the test statistic.

References


See Also

- `svytest`
- `svylogrank`

Examples

```r
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, fpc=~fpc, data=apiclus1)

svyranktest(ell~comp.imp, dclus1)
svyranktest(ell~comp.imp, dclus1, test="median")

svyranktest(ell~stype, dclus1)
svyranktest(ell~stype, dclus1, test="median")

str(svyranktest(ell~stype, dclus1))
```

## upper quartile
svyratio(ell~comp.imp, dclus1, test=function(r,N) as.numeric(r>0.75*N))

quantiletest<-function(p){
  rval<-function(r,N) as.numeric(r>(N*p))
  attr(rval,"name")<-paste(p,"quantile")
  rval
}
svyranktest(ell~comp.imp, dclus1, test=quantiletest(0.5))
svyranktest(ell~comp.imp, dclus1, test=quantiletest(0.75))

## replicate weights
rclus1<-as.svrepdesign(dclus1)
svyranktest(ell~stype, rclus1)

svyratio

### Description

Ratio estimation and estimates of totals based on ratios for complex survey samples. Estimating domain (subpopulation) means can be done more easily with svymean.

### Usage

- svyratio(numerator=formula, denominator, design, separate=FALSE, na.rm=FALSE, formula, covmat=FALSE, deff=FALSE, influence=FALSE, ...)
- svyratio(numerator=formula, denominator, design, na.rm=FALSE, formula, covmat=FALSE, return.replicates=FALSE, deff=FALSE, ...)
- svyratio(numerator=formula, denominator, design, separate=FALSE, na.rm=FALSE, formula, ...)
- predict(object, total, se=TRUE, ...)
- predict(object, total, se=TRUE, ...)
- predict(object, total, se=TRUE, ...)
- predict(object, total, se=TRUE, ...)
- SE(object,...,drop=TRUE)
- coef(object,...,drop=TRUE)
- confint(object, parm, level = 0.95, df = Inf,...)
svyratio

Arguments

- **numerator, formula**
  - formula, expression, or data frame giving numerator variable(s)
- **denominator**
  - formula, expression, or data frame giving denominator variable(s)
- **design**
  - survey design object
- **object**
  - result of svyratio
- **total**
  - vector of population totals for the denominator variables in object, or list of vectors of population stratum totals if separate=TRUE
- **se**
  - Return standard errors?
- **separate**
  - Estimate ratio separately for strata
- **na.rm**
  - Remove missing values?
- **covmat**
  - Compute the full variance-covariance matrix of the ratios
- **deff**
  - Compute design effects
- **return.replicates**
  - Return replicate estimates of ratios
- **influence**
  - Return influence functions
- **drop**
  - Return a vector rather than a matrix
- **parm**
  - a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
- **level**
  - the confidence level required.
- **df**
  - degrees of freedom for t-distribution in confidence interval, use degf(design)
  - for number of PSUs minus number of strata
- **...**
  - Other unused arguments for other methods

Details

The separate ratio estimate of a total is the sum of ratio estimates in each stratum. If the stratum totals supplied in the total argument and the strata in the design object both have names these names will be matched. If they do not have names it is important that the sample totals are supplied in the correct order, the same order as shown in the output of summary(design).

When design is a two-phase design, stratification will be on the second phase.

Value

svyratio returns an object of class svyratio. The predict method returns a matrix of population totals and optionally a matrix of standard errors.

Author(s)

Thomas Lumley

References

See Also

svydesign
svymean for estimating proportions and domain means
calibrate for estimators related to the separate ratio estimator.

Examples

data(scd)

## survey design objects
scddes<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA, nest=TRUE, fpc=rep(5,6))
scdnofpc<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA, nest=TRUE)

# convert to BRR replicate weights
scd2brr <- as.svrepdesign(scdnofpc, type="BRR")

# use BRR replicate weights from Levy and Lemeshow
repweights<-2*cbind(c(1,0,1,0,1,0), c(1,0,0,1,0,1), c(0,1,1,0,0,1),
c(0,1,0,1,1,0))
scdrep<-svrepdesign(data=scd, type="BRR", repweights=repweights)

# ratio estimates
svyratio(~alive, ~arrests, design=scddes)
svyratio(~alive, ~arrests, design=scdnofpc)
svyratio(~alive, ~arrests, design=scd2brr)
svyratio(~alive, ~arrests, design=scdrep)

data(api)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)

## domain means are ratio estimates, but available directly
svyratio(~api.stu*(comp.imp=='Yes')), ~as.numeric(comp.imp=='Yes'), dstrat)
svymean(~api.stu, subset(dstrat, comp.imp=='Yes'))

## separate and combined ratio estimates of total
(sep<-svyratio(~api.stu, ~enroll, dstrat, separate=TRUE))
(com<-svyratio(~api.stu, ~enroll, dstrat))

stratum.totals<-list(E=1877350, H=1013824, M=920298)
predict(sep, total=stratum.totals)
predict(com, total=sum(unlist(stratum.totals)))

SE(com)
coef(com)
coef(com, drop=FALSE)
confint(com)
svyrecvar

Variance estimation for multistage surveys

Description

Compute the variance of a total under multistage sampling, using a recursive descent algorithm.

Usage

svyrecvar(x, clusters, stratas,fpcs, postStrata = NULL, lonely.psu = getOption("survey.lonely.psu"), one.stage=getOption("survey.ultimate.cluster"))

Arguments

x Matrix of data or estimating functions
clusters Data frame or matrix with cluster ids for each stage
stratas Strata for each stage
fpcs Information on population and sample size for each stage, created by as.fpc
postStrata post-stratification information as created by postStratify or calibrate
lonely.psu How to handle strata with a single PSU
one.stage If TRUE, compute a one-stage (ultimate-cluster) estimator

Details

The main use of this function is to compute the variance of the sum of a set of estimating functions under multistage sampling. The sampling is assumed to be simple or stratified random sampling within clusters at each stage except perhaps the last stage. The variance of a statistic is computed from the variance of estimating functions as described by Binder (1983).

Use one.stage=FALSE for compatibility with other software that does not perform multi-stage calculations, and set options(survey.ultimate.cluster=TRUE) to make this the default.

The idea of a recursive algorithm is due to Bellhouse (1985). Texts such as Cochran (1977) and Sarndal et al (1991) describe the decomposition of the variance into a single-stage between-cluster estimator and a within-cluster estimator, and this is applied recursively.

If one.stage is a positive integer it specifies the number of stages of sampling to use in the recursive estimator.

If pps="brewer", standard errors are estimated using Brewer's approximation for PPS without replacement, option 2 of those described by Berger (2004). The fpc argument must then be specified in terms of sampling fractions, not population sizes (or omitted, but then the pps argument would have no effect and the with-replacement standard errors would be correct).

Value

A covariance matrix
Note

A simple set of finite population corrections will only be exactly correct when each successive stage uses simple or stratified random sampling without replacement. A correction under general unequal probability sampling (eg PPS) would require joint inclusion probabilities (or, at least, sampling probabilities for units not included in the sample), information not generally available.

The quality of Brewer’s approximation is excellent in Berger’s simulations, but the accuracy may vary depending on the sampling algorithm used.

References

Brewer KRW (2002) Combined Survey Sampling Inference (Weighing Basu’s Elephants) [Chapter 9]

See Also

svrVar for replicate weight designs
svyCprod for a description of how variances are estimated at each stage

Examples

data(mu284)
dmu284<-svydesign(id=~id1+id2,fpc=~n1+n2, data=mu284)
svytotal(~y1, dmu284)

data(api)
# two-stage cluster sample
dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
surveysummary(dclus2)
svymean(~api00, dclus2)
svytotal(~enroll, dclus2, na.rm=TRUE)

# bootstrap for multistage sample
mrclus2<-as.svrepdesign(dclus2, type="mrb", replicates=100)
svytotal(~enroll, mrclus2, na.rm=TRUE)

# two-stage 'with replacement'
dclus2wr<-svydesign(id=~dnum+snum, weights=~pw, data=apiclus2)
surveysummary(dclus2wr)
svymean(~api00, dclus2wr)
svyscoretest

svytotal(~enroll, dclus2wr, na.rm=TRUE)

svyscoretest

Score tests in survey regression models

Description

Performs two versions of the efficient score test. These are the same for a single parameter. In
the working score test, different parameters are weighted according to the inverse of the estimated
population Fisher information. In the pseudoscore test, parameters are weighted according to the
inverse of their estimated covariance matrix.

Usage

svyscoretest(model, drop.terms=NULL, add.terms=NULL,
method=c("working","pseudoscore","individual"), ddf=NULL,
lrt.approximation = "satterthwaite", ...)
## S3 method for class 'svyglm'
svyscoretest(model, drop.terms=NULL, add.terms=NULL,
method=c("working","pseudoscore","individual"), ddf=NULL,
lrt.approximation = "satterthwaite", fullrank=TRUE, ...)

Arguments

model A model of a class having a svyscoretest method (currently just svyglm)
drop.terms Model formula giving terms to remove from model
add.terms Model formula giving terms to add to model
method The type of score test to use. For a single parameter they are equivalent. To
report tests for each column separately use individual
ddf denominator degrees of freedom for an F or linear combination of F distribu-
tions. Use Inf to get chi-squared distributions. NULL asks for the model residual
degrees of freedom, which is conservative.
lrt.approximation For the working score, the method for computing/approximating the null distri-
bution: see pchisqsum
fullrank If FALSE and method="individual", keep even linearly dependent columns of
the efficient score
... for future expansion
Details

The working score test will be asymptotically equivalent to the Rao-Scott likelihood ratio test computed by `regTermTest` and `anova.svyglm`. The paper by Rao, Scott and Skinner calls this a "naive" score test. The null distribution is a linear combination of chi-squared (or F) variables.

The pseudoscore test will be asymptotically equivalent to the Wald test computed by `regTermTest`; it has a chi-squared (or F) null distribution.

If `ddf` is negative or zero, which can happen with large numbers of predictors and small numbers of PSUs, it will be changed to 1 with a warning.

Value

For "pseudoscore" and "working" score methods, a named vector with the test statistic, degrees of freedom, and p-value. For "individual" an object of class "svystat"

References


See Also

`regTermTest`, `anova.svyglm`

Examples

data(myco)
dmyco<-svydesign(id=~1, strata=~interaction(Age,leprosy),weights=~wt,data=myco)
m_full<-svyglm(leprosy~I((Age+7.5)^-2)+Scar, family=quasibinomial, design=dmyco)
svyscoretest(m_full, ~Scar)

svyscoretest(m_full,add.terms= ~I((Age+7.5)^-2):Scar)
svyscoretest(m_full,add.terms= ~factor(Age), method="pseudo")
svyscoretest(m_full,add.terms= ~factor(Age),method="individual",fullrank=FALSE)

svyscoretest(m_full,add.terms= ~factor(Age),method="individual")
svysmooth

Usage

svysmooth(formula, design, ...)  
## Default S3 method:  
svysmooth(formula, design, method = c("locpoly", "quantreg"),  
  bandwidth = NULL, quantile, df = 4, ...)  
## S3 method for class 'svysmooth'  
plot(x, which=NULL, type="l", xlabs=NULL, ylab=NULL,...)  
## S3 method for class 'svysmooth'  
lines(x, which=NULL,...)  
make.panel.svysmooth(design,bandwidth=NULL)

Arguments

formula One-sided formula for density estimation, two-sided for smoothing  
design Survey design object  
method local polynomial smoothing for the mean or regression splines for quantiles  
bandwidth Smoothing bandwidth for "locpoly" or NULL for automatic choice  
quantile quantile to be estimated for "quantreg"  
df Degrees of freedom for "quantreg"  
which Which plots to show (default is all)  
type as for plot  
xlabs Optional vector of x-axis labels  
ylab Optional y-axis label  
... More arguments  
x Object of class svysmooth

Details

svysmooth does one-dimensional smoothing. If formula has multiple predictor variables a separate one-dimensional smooth is performed for each one.  
For method="locpoly" the extra arguments are passed to locpoly from the KernSmooth package, for method="quantreg" they are passed to rq from the quantreg package. The automatic choice of bandwidth for method="locpoly" uses the default settings for dp1k and dpill in the KernSmooth package.  
make.panel.svysmooth() makes a function that plots points and draws a weighted smooth curve through them, a weighted replacement for panel.smooth that can be passed to functions such as termplot or plot.lm. The resulting function has a span argument that will set the bandwidth; if this is not specified the automatic choice will be used.

Value

An object of class svysmooth, a list of lists, each with x and y components.

See Also

svyhist for histograms
svystandardize

Direct standardization within domains

Description

In health surveys it is often of interest to standardize domains to have the same distribution of, eg, age as in a target population. The operation is similar to post-stratification, except that the totals for the domains are fixed at the current estimates, not at known population values. This function matches the estimates produced by the (US) National Center for Health Statistics.

Usage

svystandardize(design, by, over, population, excluding.missing = NULL)

Arguments

design survey design object

by A one-sided formula specifying the variables whose distribution will be standardised

over A one-sided formula specifying the domains within which the standardisation will occur, or ~1 to use the whole population.

population Desired population totals or proportions for the levels of combinations of variables in by

excluding.missing Optionally, a one-sided formula specifying variables whose missing values should be dropped before calculating the domain totals.
svystandardize  

Value  
A new survey design object of the same type as the input.

Note  
The standard error estimates do not exactly match the NCHS estimates

References  
National Center for Health Statistics https://www.cdc.gov/nchs/tutorials/NHANES/NHANESAnalyses/agestandardization/age_standardization_intro.htm

See Also  
postStratify, svyby

Examples  
## matches http://www.cdc.gov/nchs/data/databriefs/db92_fig1.png
```r
data(nhanes)
popage <- c( 55901 , 77670 , 72816 , 45364 )
design<-svydesign(id=~SDMVPSU, strata=~SDMVSTRA, weights=~WTMEC2YR, data=nhanes, nest=TRUE)
stdes<-svystandardize(design, by=~agecat, over=~race+RIAGENDR,
  population=popage, excluding.missing=~HI_CHOL)
svyby(~HI_CHOL, ~race+RIAGENDR, svymean, design=subset(stdes,
agecat!="(0,19]"))
```

```r
data(nhanes)
nhanes_design <- svydesign(ids = ~ SDMVPSU, strata = ~ SDMVSTRA,
    weights = ~ WTMEC2YR, nest = TRUE, data = nhanes)
## These are the same
nhanes_adj <- svystandardize(update(nhanes_design, all_adults = "1"),
    by = ~ agecat, over = ~ all_adults,
    population = c(55901, 77670, 72816, 45364),
    excluding.missing = ~ HI_CHOL)
svymean(~I(HI_CHOL == 1), nhanes_adj, na.rm = TRUE)
```

```r
nhanes_adj <- svystandardize(nhanes_design,
    by = ~ agecat, over = ~ 1,
    population = c(55901, 77670, 72816, 45364),
    excluding.missing = ~ HI_CHOL)
svymean(~I(HI_CHOL == 1), nhanes_adj, na.rm = TRUE)
```
svysurvreg

Fit accelerated failure models to survey data

Description

This function calls survreg from the 'survival' package to fit accelerated failure (accelerated life) models to complex survey data, and then computes correct standard errors by linearisation. It has the same arguments as survreg, except that the second argument is design rather than data.

Usage

### S3 method for class 'survey.design'

svysurvreg(formula, design, weights=NULL, subset=NULL, ...)

Arguments

- formula: Model formula
- design: Survey design object, including two-phase designs
- weights: Additional weights to multiply by the sampling weights. No, I don’t know why you’d want to do that.
- subset: subset to use in fitting (if needed)
- ...: Other arguments of survreg

Value

Object of class svysurvreg, with the same structure as a survreg object but with NA for the log-likelihood.

Note

The residuals method is identical to that for survreg objects except the weighted option defaults to TRUE.

Examples

data(pbc, package="survival")
pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
biasmodel<-glm(randomized~age*edema, data=pbc)
pbc$randprob<-fitted(biasmodel)
dpbc<svydesign(id=~1, prob=-randprob, strata=~edema, 
data=subset(pbc, randomized))

model <- svysurvreg(Surv(time, status>0)~bili+protime+albumin, design=dpbc, dist="weibull")
summary(model)
svytable

Contingency tables for survey data

Description

Contingency tables and chisquared tests of association for survey data.

Usage

## S3 method for class 'survey.design'
svytable(formula, design, Ntotal = NULL, round = FALSE,...)
## S3 method for class 'svyrep.design'
svytable(formula, design,
Ntotal = sum(weights(design, "sampling")), round = FALSE,...)
## S3 method for class 'survey.design'
svychisq(formula, design,
statistic = c("F", "Chisq","Wald","adjWald","lincom",
"saddlepoint","wls-score"),na.rm=TRUE,...)
## S3 method for class 'svyrep.design'
svychisq(formula, design,
statistic = c("F", "Chisq","Wald","adjWald","lincom",
"saddlepoint","wls-score"),na.rm=TRUE,...)
## S3 method for class 'svytable'
summary(object,
statistic = c("F","Chisq","Wald","adjWald","lincom","saddlepoint"),...)
degf(design, ...)  
## S3 method for class 'survey.design2'
degf(design, ...)  
## S3 method for class 'svyrep.design'
degf(design, tol=1e-5,...)

Arguments

formula Model formula specifying margins for the table (using + only)
design survey object
statistic See Details below
Ntotal A population total or set of population stratum totals to normalise to.
round Should the table entries be rounded to the nearest integer?
na.rm Remove missing values
object Output from svytable
... For svytable these are passed to xtabs. Use exclude=NULL, na.action=na.pass to include NAs in the table
tol Tolerance for qr in computing the matrix rank
svytable

Details

The svytable function computes a weighted crosstabulation. This is especially useful for producing graphics. It is sometimes easier to use svytotal or svymeans, which also produce standard errors, design effects, etc.

The frequencies in the table can be normalised to some convenient total such as 100 or 1.0 by specifying the Ntotal argument. If the formula has a left-hand side the mean or sum of this variable rather than the frequency is tabulated.

The Ntotal argument can be either a single number or a data frame whose first column gives the (first-stage) sampling strata and second column the population size in each stratum. In this second case the svytable command performs 'post-stratification': tabulating and scaling to the population within strata and then adding up the strata.

As with other xtabs objects, the output of svytable can be processed by ftable for more attractive display. The summary method for svytable objects calls svychisq for a test of independence.

svychisq computes first and second-order Rao-Scott corrections to the Pearson chisquared test, and two Wald-type tests.

The default (statistic="F") is the Rao-Scott second-order correction. The p-values are computed with a Satterthwaite approximation to the distribution and with denominator degrees of freedom as recommended by Thomas and Rao (1990). The alternative statistic="Chisq" adjusts the Pearson chisquare statistic by a design effect estimate and then compares it to the chisquare distribution it would have under simple random sampling.

The statistic="Wald" test is that proposed by Koch et al (1975) and used by the SUDAAN software package. It is a Wald test based on the differences between the observed cells counts and those expected under independence. The adjustment given by statistic="adjWald" reduces the statistic when the number of PSUs is small compared to the number of degrees of freedom of the test. Thomas and Rao (1987) compare these tests and find the adjustment benefical.

statistic="lincom" replaces the numerator of the Rao-Scott F with the exact asymptotic distribution, which is a linear combination of chi-squared variables (see pchisqsum, and statistic="saddlepoint" uses a saddlepoint approximation to this distribution. The CompQuadForm package is needed for statistic="lincom" but not for statistic="saddlepoint". The saddlepoint approximation is especially useful when the p-value is very small (as in large-scale multiple testing problems).

statistic="wls-score" is an experimental implementation of the weighted least squares score test of Lipsitz et al (2015). It is not identical to that paper, for example, I think the denominator degrees of freedom need to be reduced by JK for a JxK table, not (J-1)(K-1). And it's very close to the "adjWald" test.

For designs using replicate weights the code is essentially the same as for designs with sampling structure, since the necessary variance computations are done by the appropriate methods of svytotal and svymeans. The exception is that the degrees of freedom is computed as one less than the rank of the matrix of replicate weights (by degf).

At the moment, svychisq works only for 2-dimensional tables.

Value

The table commands return an xtabs object, svychisq returns a htest object.
Note
Rao and Scott (1984) leave open one computational issue. In computing ‘generalised design effects’ for these tests, should the variance under simple random sampling be estimated using the observed proportions or the the predicted proportions under the null hypothesis? svychisq uses the observed proportions, following simulations by Sribney (1998), and the choices made in Stata.

References
Koch, GG, Freeman, DH, Freeman, JL (1975) "Strategies in the multivariate analysis of data from complex surveys" International Statistical Review 43: 59-78

See Also
svytotal and svymean report totals and proportions by category for factor variables.
See svyby and ftable.svystat to construct more complex tables of summary statistics.
See svyloglin for loglinear models.
See regTermTest for Rao-Scott tests in regression models.
See https://notstatschat.rbind.io/2019/06/08/design-degrees-of-freedom-brief-note/ for an explanation of the design degrees of freedom with replicate weights.

Examples
data(api)
xtabs(~sch.wide+stype, data=apipop)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
summary(dclus1)
(tbl <- svytable(~sch.wide+stype, dclus1))
plot(tbl)
fourfoldplot(svytable(~sch.wide+comp.imp+stype,design=dclus1,round=TRUE), conf.level=0)
svychisq(~sch.wide+stype, dclus1)
summary(tbl, statistic="Chisq")
svychisq(~sch.wide+stype, dclus1, statistic="adjWald")

rclus1 <- as.svrepdesign(dclus1)
summary(svytable(~sch.wide+stype, rclus1))
svychisq(~sch.wide+stype, rclus1, statistic="adjWald")

svyttest

### Design-based t-test

#### Description

One-sample or two-sample t-test. This function is a wrapper for `svymean` in the one-sample case and for `svyglm` in the two-sample case. Degrees of freedom are degf(design)-1 for the one-sample test and degf(design)-2 for the two-sample case.

#### Usage

```r
svyttest(formula, design, ...)
```

#### Arguments

- `formula`: Formula, `outcome~group` for two-sample, `outcome~0` or `outcome~1` for one-sample. The `group` variable must be a factor or character with two levels, or be coded 0/1 or 1/2
- `design`: survey design object
- `...`: for methods

#### Value

Object of class `htest`

#### See Also

`t.test`

#### Examples

```r
data(api)
dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
tt<-svyttest(enroll~comp.imp, dclus2)
tt
confint(tt, level=0.9)

svyttest(enroll~I(stype=="E"),dclus2)
svyttest(I(api00-api99)~0, dclus2)
```
trimWeights

Trim sampling weights

Description
Trims very high or very low sampling weights to reduce the influence of outlying observations. In a replicate-weight design object, the replicate weights are also trimmed. The total amount trimmed is divided among the observations that were not trimmed, so that the total weight remains the same.

Usage
trimWeights(design, upper = Inf, lower = -Inf, ...)
## S3 method for class 'survey.design2'
trimWeights(design, upper = Inf, lower = -Inf, strict=FALSE,...)
## S3 method for class 'svyrep.design'
trimWeights(design, upper = Inf, lower = -Inf, strict=FALSE, compress=FALSE,...)

Arguments
design         A survey design object
upper          Upper bound for weights
lower          Lower bound for weights
strict         The reapportionment of the 'trimmings' from the weights can push other weights over the limits. If trim=TRUE the function repeats the trimming iteratively to prevent this. For replicate-weight designs strict applies only to the trimming of the sampling weights.
compress       Compress the replicate weights after trimming.
...            Other arguments for future expansion

Value
A new survey design object with trimmed weights.

See Also
calibrate has a trim option for trimming the calibration adjustments.

Examples
data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)

pop.totals<-c(`(Intercept)`=6194, stypeH=755, stypeM=1018, api99=3914869)
dclus1g<-calibrate(dclus1, ~stype+api99, pop.totals)
summary(weights(dclus1g))
dclus1t<-trimWeights(dclus1g,lower=20, upper=45)
summary(weights(dclus1t))
dclus1tt<-trimWeights(dclus1g, lower=20, upper=45, strict=TRUE)
summary(weights(dclus1tt))

svymean(~api99+api00+stype, dclus1g)
svymean(~api99+api00+stype, dclus1t)
svymean(~api99+api00+stype, dclus1tt)

twophase  Two-phase designs

Description

In a two-phase design a sample is taken from a population and a subsample taken from the sample, typically stratified by variables not known for the whole population. The second phase can use any design supported for single-phase sampling. The first phase must currently be one-stage element or cluster sampling.

Usage

twophase(id, strata = NULL, probs = NULL, weights = NULL, fpc = NULL, subset, data, method=c("full","approx","simple"), pps=NULL)
twophasevar(x, design)
twophase2var(x, design)

Arguments

id  list of two formulas for sampling unit identifiers
strata  list of two formulas (or NULLs) for stratum identifies
probs  list of two formulas (or NULLs) for sampling probabilities
weights  Only for method="approx", list of two formulas (or NULLs) for sampling weights
fpc  list of two formulas (or NULLs) for finite population corrections
subset  formula specifying which observations are selected in phase 2
data  Data frame will all data for phase 1 and 2
method  "full" requires (much) more memory, but gives unbiased variance estimates for general multistage designs at both phases. "simple" or "approx" uses the standard error calculation from version 3.14 and earlier, which uses much less memory and is correct for designs with simple random sampling at phase one and stratified random sampling at phase two.
pps  With method="full", an optional list of two PPS specifications for svydesign. At the moment, the phase-one element must be NULL
x  probability-weighted estimating functions
design  two-phase design
Details

The population for the second phase is the first-phase sample. If the second phase sample uses stratified (multistage cluster) sampling without replacement and all the stratum and sampling unit identifier variables are available for the whole first-phase sample it is possible to estimate the sampling probabilities/weights and the finite population correction. These would then be specified as NULL.

Two-phase case-control and case-cohort studies in biostatistics will typically have simple random sampling with replacement as the first stage. Variances given here may differ slightly from those in the biostatistics literature where a model-based estimator of the first-stage variance would typically be used.

Variance computations are based on the conditioning argument in Section 9.3 of Sarndal et al. Method "full" corresponds exactly to the formulas in that reference. Method "simple" or "approx" (the two are the same) uses less time and memory but is exact only for some special cases. The most important special case is the two-phase epidemiologic designs where phase 1 is simple random sampling from an infinite population and phase 2 is stratified random sampling. See the tests directory for a worked example. The only disadvantage of method="simple" in these cases is that standardization of margins (marginpred) is not available.

For method="full", sampling probabilities must be available for each stage of sampling, within each phase. For multistage sampling this requires specifying either fpc or probs as a formula with a term for each stage of sampling. If no fpc or probs are specified at phase 1 it is treated as simple random sampling from an infinite population, and population totals will not be correctly estimated, but means, quantiles, and regression models will be correct.

The pps argument allows for PPS sampling at phase two (or eventually at phase one), and also for Poisson sampling at phase two as a model for non-response.

Value

twophase returns an object of class twophase2 (for method="full") or twophase. The structure of twophase2 objects may change as unnecessary components are removed.

twophase2var and twophasevar return a variance matrix with an attribute containing the separate phase 1 and phase 2 contributions to the variance.

References


See Also

svydesign, svyrecvar for multi*stage* sampling
calibrate for calibration (GREG) estimators.
estWeights for two-phase designs for missing data.
The "epi" and "phase1" vignettes for examples and technical details.

Examples

```r
## two-phase simple random sampling.
data(pbc, package="survival")
pbc$randomized<-with(pbc, !is.na(trt) & trt>0)
pbc$id<-1:nrow(pbc)
d2pbc<-twophase(id=list(~id,~id), data=pbc, subset=~randomized)
svymean(~bili, d2pbc)

## two-stage sampling as two-phase
data(mu284)
ii<-with(mu284, c(1:15, rep(1:5,n2[1:5]-3)))
mu284.1$id<-1:nrow(mu284.1)
mu284.1$sub<-rep(c(TRUE,FALSE),c(15,34-15))
dmu284<-svydesign(id=~id1+id2,fpc=~n1+n2, data=mu284)
## first phase cluster sample, second phase stratified within cluster
d2mu284<-twophase(id=list(~id1,-id),strata=list(NULL,~id),
                   fpc=list(~n1,NULL),data=mu284.1,subset=~sub)
svymean(~y1,dmu284)
svymean(~y1,d2mu284)

## case-cohort design: this example requires R 2.2.0 or later
library("survival")
data(nwtco)

## stratified on case status
dcchs<-twophase(id=list(~seqno,~seqno), strata=list(NULL,~rel),
                 subset=~(in.subcohort | rel), data=nwtco)
svycoxph(Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12), design=dcchs)

## Using survival::cch
subcoh <- nwtco$in.subcohort
selccoh <- with(nwtco, rel==1|subcoh==1)
ccoh.data <- nwtco[selccoh,]
ccoh.data$subcohort <- subcoh[selccoh]
cch(Surv(edrel, rel) ~ factor(stage) + factor(histol) + I(age/12), data =ccoh.data,
    subcoh = ~subcohort, id=~seqno, cohort.size=4028, method="LinYing")

## two-phase case-control
## Similar to Breslow & Chatterjee, Applied Statistics (1999) but with
## a slightly different version of the data set
nwtco$incc2<-as.logical(with(nwtco, ifelse(rel | instit==2,1,rbinom(nrow(nwtco),1,1))))
dccs2<-twophase(id=list(~seqno,~seqno),strata=list(NULL,~interaction(rel,instit)),
```

data=nwtco, subset=~incc2)
dccs8<-twophase(id=list(~seqno,~seqno),strata=list(NULL,~interaction(rel,stage,instit)),
data=nwtco, subset=~incc2)
summary(glm(rel~factor(stage)*factor(histol),data=nwtco,family=binomial()))
summary(svyglm(rel~factor(stage)*factor(histol),design=dccs2,family=quasibinomial()))
summary(svyglm(rel~factor(stage)*factor(histol),design=dccs8,family=quasibinomial()))

## Stratification on stage is really post-stratification, so we should use calibrate()
gccs8<-calibrate(dccs2, phase=2, formula=~interaction(rel,stage,instit))
summary(svyglm(rel~factor(stage)*factor(histol),design=gccs8,family=quasibinomial()))

## For this saturated model calibration is equivalent to estimating weights.
pccs8<-calibrate(dccs2, phase=2,formula=~interaction(rel,stage,instit), calfun="rrz")
summary(svyglm(rel~factor(stage)*factor(histol),design=pccs8,family=quasibinomial()))

## Since sampling is SRS at phase 1 and stratified RS at phase 2, we
## can use method="simple" to save memory.
dccs8_simple<-twophase(id=list(~seqno,~seqno),strata=list(NULL,~interaction(rel,stage,instit)),
data=nwtco, subset=~incc2,method="simple")
summary(svyglm(rel~factor(stage)*factor(histol),design=dccs8_simple,family=quasibinomial()))

update.survey.design  Add variables to a survey design

Description
Update the data variables in a survey design, either with a formula for a new set of variables or with
an expression for variables to be added.

Usage
## S3 method for class 'survey.design'
update(object, ...)
## S3 method for class 'twophase'
update(object, ...)
## S3 method for class 'svyrep.design'
update(object, ...)
## S3 method for class 'DBIsvydesign'
update(object, ...)

Arguments
object  a survey design object
...  Arguments tag=expr add a new variable tag computed by evaluating expr in
      the survey data.
Details

Database-backed objects may not have write access to the database and so `update` does not attempt to modify the database. The expressions are stored and are evaluated when the data is loaded.

If a set of new variables will be used extensively it may be more efficient to modify the database, either with SQL queries from the R interface or separately. One useful intermediate approach is to create a table with the new variables and a view that joins this table to the table of existing variables.

There is now a base-R function `transform` for adding new variables to a data frame, so I have added `transform` as a synonym for `update` for survey objects.

Value

A survey design object

See Also

`svydesign`, `svrepdesign`, `twophase`

Examples

data(api)
dstrat<-svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
dstrat<-update(dstrat, apidiff=api00-api99)
svymean(~api99+api00+apidiff, dstrat)

weights.survey.design

Survey design weights

Description

Extract weights from a survey design object.

Usage

```r
## S3 method for class 'survey.design'
weights(object, ...)

## S3 method for class 'svyrep.design'
weights(object, 
type=c("replication","sampling","analysis"), ...)

## S3 method for class 'survey_fpc'
weights(object,final=TRUE,...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Survey design object</td>
</tr>
<tr>
<td>type</td>
<td>Type of weights: &quot;analysis&quot; combines sampling and replication weights.</td>
</tr>
<tr>
<td>final</td>
<td>If FALSE return a data frame with sampling weights at each stage of sampling.</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments ignored</td>
</tr>
</tbody>
</table>
with.svyimputationList

Value

vector or matrix of weights

See Also

svydesign, svrepdesign, as.fpc

Examples

data(scd)

scddes<-svydesign(data=scd, prob=~1, id=~ambulance, strata=~ESA,
   nest=TRUE, fpc=rep(5,6))
repweights<-2*cbind(c(1,0,1,0,1,0), c(1,0,0,1,0,1), c(0,1,1,0,0,1), c(0,1,0,1,1,0))
scdrep<-svrepdesign(data=scd, type="BRR", repweights=repweights)
weights(scdrep)
weights(scdrep, type="sampling")
weights(scdrep, type="analysis")
weights(scddes)

with.svyimputationList

Analyse multiple imputations

Description

Performs a survey analysis on each of the designs in a svyimputationList objects and returns a list of results suitable for MIcombine. The analysis may be specified as an expression or as a function.

Usage

## S3 method for class 'svyimputationList'
with(data, expr, fun, ..., multicore=getOption("survey.multicore"))
## S3 method for class 'svyimputationList'
subset(x, subset,...)

Arguments

data,x A svyimputationList object
expr An expression giving a survey analysis
fun A function taking a survey design object as its argument
... for future expansion
multicore Use multicore package to distribute imputed data sets over multiple processors?
subset An logical expression specifying the subset
Value

A list of the results from applying the analysis to each design object.

See Also

MIcombine, in the mitools package

Examples

library(mitools)
data.dir<-system.file("dta",package="mitools")
files.men<-list.files(data.dir,pattern="m.\.dta\$",full=TRUE)
men<-imputationList(lapply(files.men, foreign::read.dta,
warn.missing.labels=FALSE))
files.women<-list.files(data.dir,pattern="f.\.dta\$",full=TRUE)
women<-imputationList(lapply(files.women, foreign::read.dta,
warn.missing.labels=FALSE))
men<-update(men, sex=1)
women<-update(women, sex=0)
all<-rbind(men,women)
designs<-svydesign(id=~id, strata=~sex, data=all)
designs

results<-with(designs, svymean(~drkfre))
MIcombine(results)
summary(MIcombine(results))

repsdesigns<-as.svrepdesign(designs, type="boot", replicates=50)
MIcombine(with(repsdesigns, svymean(~drkfre)))
Arguments

- **mapping**: A formula or list of formulas describing each variable in the analysis that has plausible values. The left-hand side of the formula is the name to use in the analysis; the right-hand side gives the names in the dataset.

- **data**: A survey design object, as created by `svydesign` or `svrepdesign`.

- **action**: With `rewrite=TRUE`, a function taking a survey design object as its only argument, or a quoted expression. With `rewrite=TRUE` a function taking a survey design object as its only argument, or a quoted expression with `.DESIGN` referring to the survey design object to be used.

- **rewrite**: Rewrite action before evaluating it (versus constructing new data sets)

Value

A list of the results returned by each evaluation of action, with the call as an attribute.

See Also

- `with.svyimputationList`

Examples

```r
if(require(mitools)){
  data(pisamaths, package="mitools")
  des<-svydesign(id=~SCHOOLID+STIDSTD, strata=~STRATUM, nest=TRUE,
                 weights=~W_FSCHWT+condwt, data=pisamaths)

  oo<-options(survey.lonely.psu="remove")

  results<-withPV(list(maths~PV1MATH+PV2MATH+PV3MATH+PV4MATH+PV5MATH),
                   data=des,
                   action=quote(svyglm(maths~ST04Q01*(PCGIRLS+SMRATIO)+MATHEFF+OPENPS, design=des)),
                   rewrite=TRUE)

  summary(MIcombine(results))
  options(oo)
}
```

Description

Given a function or expression computing a statistic based on sampling weights, `withReplicates` evaluates the statistic and produces a replicate-based estimate of variance. `vcov.svrep.design` produces the variance estimate from a set of replicates and the design object.
withReplicates

Usage

withReplicates(design, theta,..., return.replicates=FALSE)
## S3 method for class 'svyrep.design'
withReplicates(design, theta, rho = NULL, ..., 
  scale.weights=FALSE, return.replicates=FALSE)
## S3 method for class 'svrepvar'
withReplicates(design, theta, ..., return.replicates=FALSE)
## S3 method for class 'svrepstat'
withReplicates(design, theta, ..., return.replicates=FALSE)
## S3 method for class 'svyimputationList'
withReplicates(design, theta, ..., return.replicates=FALSE)
## S3 method for class 'svyrep.design'
vcov(object, replicates, centre,...)

Arguments

design A survey design with replicate weights (eg from svrepdesign) or a suitable object with replicate parameter estimates
theta A function or expression: see Details below
rho If design uses BRR weights, rho optionally specifies the parameter for Fay’s variance estimator.
... Other arguments to theta
scale.weights Divide the probability weights by their sum (can help with overflow problems)
return.replicates Return the replicate estimates as well as the variance?
object The replicate-weights design object used to create the replicates
replicates A set of replicates
centre The centering value for variance calculation. If object$mse is TRUE this is the result of estimation using the sampling weights, and must be supplied. If object$mse is FALSE the mean of the replicates is used and this argument is silently ignored.

Details

The method for svyrep.design objects evaluates a function or expression using the sampling weights and then each set of replicate weights. The method for svrepvar objects evaluates the function or expression on an estimated population covariance matrix and its replicates, to simplify multivariate statistics such as structural equation models.

For the svyrep.design method, if theta is a function its first argument will be a vector of weights and the second argument will be a data frame containing the variables from the design object. If it is an expression, the sampling weights will be available as the variable .weights. Variables in the design object will also be in scope. It is possible to use global variables in the expression, but unwise, as they may be masked by local variables inside withReplicates.

For the svrepvar method a function will get the covariance matrix as its first argument, and an expression will be evaluated with .replicate set to the variance matrix.
For the `svrepstat` method a function will get the point estimate, and an expression will be evaluated with `.replicate` set to each replicate. The method can only be used when the `svrepstat` object includes replicates.

The `svyimputationList` method runs `withReplicates` on each imputed design (which must be replicate-weight designs).

**Value**

If `return.replicates=FALSE`, the weighted statistic, with the variance matrix as the "var" attribute. If `return.replicates=TRUE`, a list with elements `theta` for the usual return value and `replicates` for the replicates.

**See Also**

`svrepsdesign`, `as.svrepsdesign`, `svrVar`

**Examples**

```r
data(scd)
repweights<-2*cbind(c(1,0,1,0,1,0), c(1,0,0,1,0,1), c(0,1,1,0,0,1),
c(0,1,0,1,1,0))
scdrep<-svrepsdesign(data=scd, type="BRR", repweights=repweights)

a<-svyratio(~alive, ~arrests, design=scdrep)
print(a$ratio)
print(a$var)
withReplicates(scdrep, quote(sum(.weights*alive)/sum(.weights*arrests)))
withReplicates(scdrep, function(w,data)
  sum(w*data$alive)/sum(w*data$arrests))

data(api)
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
rclus1<-as.svrepsdesign(dclus1)
varmat<-svyvar(~api00+api99+ell+meals+hsg+mobility,rclus1,return.replicates=TRUE)
withReplicates(varmat, quote( factanal(covmat=.replicate, factors=2)$unique) )

data(nhanes)
nhanesdesign <- svydesign(id=~SDMVPSU, strata=~SDMVSTRA, weights=~WTMEC2YR, nest=TRUE,data=nhanes)
logistic <- svyglm(HI_CHOL~race+agecat+RIAGENDR, design=as.svrepsdesign(nhanesdesign), family=quasibinomial, return.replicates=TRUE)
fitted<-predict(logistic, return.replicates=TRUE, type="response")
sensitivity<-function(pred,actual) mean(pred>0.1 & actual)/mean(actual)
withReplicates(fitted, sensitivity, actual=logistic$y)
```

```r
## Not run:
library(quantreg)
data(api)
## one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
## convert to bootstrap
bclus1<-as.svrepsdesign(dclus1,type="bootstrap", replicates=100)
```
## median regression

withReplicates(bclus1, quote(coef(rq(api00~api99, tau=0.5, weights=.weights))))

## End(Not run)

## pearson correlation

dstrat <- svydesign(id=~1, strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
bstrat<- as.svrepdesign(dstrat,type="subbootstrap")

v <- svyvar(~api00+api99, bstrat, return.replicates=TRUE)
vcor<--cov2cor(as.matrix(v))[2,1]
vreps<-v$replicates
correps<-apply(vreps,1, function(v) v[2]/sqrt(v[1]*v[4]))

vcov(bstrat,correps, centre=vcor)

---

### xdesign

*Crossed effects and other sparse correlations*

**Description**

Defines a design object with multiple dimensions of correlation: observations that share any of the `id` variables are correlated, or you can supply an adjacency matrix or Matrix to specify which are correlated. Supports crossed designs (eg multiple raters of multiple objects) and non-nested observational correlation (eg observations sharing primary school or secondary school). Has methods for `svymean`, `svytotal`, `svyglm` (so far).

**Usage**

```r
xdesign(id = NULL, strata = NULL, weights = NULL, data, fpc = NULL, adjacency = NULL, overlap = c("unbiased", "positive"), allow.non.binary = FALSE)
```

**Arguments**

- `id` list of formulas specifying cluster identifiers for each clustering dimension (or NULL)
- `strata` Not implemented
- `weights` model formula specifying (sampling) weights
- `data` data frame containing all the variables
- `fpc` Not implemented
- `adjacency` Adjacency matrix or Matrix indicating which pairs of observations are correlated
- `overlap` See details below
- `allow.non.binary` If FALSE check that adjacency is a binary 0/1 or TRUE/FALSE matrix or Matrix.
Details

Subsetting for these objects actually drops observations; it is not equivalent to just setting weights to zero as for survey designs. So, for example, a subset of a balanced design will not be a balanced design.

The overlap option controls double-counting of some variance terms. Suppose there are two clustering dimensions, ~a and ~b. If we compute variance matrices clustered on a and clustered on b and add them, observations that share both a and b will be counted twice, giving a positively biased estimator. We can subtract off a variance matrix clustered on combinations of a and b to give an unbiased variance estimator. However, the unbiased estimator is not guaranteed to be positive definite. In the references, Miglioretti and Heagerty use the overlap="positive" estimator and Cameron et al use the overlap="unbiased" estimator.

Value

An object of class xdesign

References


See Also

salamander

Examples

```r
## With one clustering dimension, is close to the with-replacement
## survey estimator, but not identical unless clusters are equal size
data(api)
dclus1r<-svydesign(id=~dnum, weights=~pw, data=apiclus1)
xclus1<-xdesign(id=list(~dnum), weights=~pw, data=apiclus1)
xclus1

svymean(~enroll,dclus1r)
svymean(~enroll,xclus1)

data(salamander)
xsalamander<-xdesign(id=list(~Male, ~Female), data=salamander,
                     overlap="unbiased")
xsalamander
degf(xsalamander)
```
Description
Design information from the Youth Risk Behaviors Survey (YRBS), together with the single variable ‘Never/Rarely wore bike helmet’. Used as an analysis example by CDC.

Usage
data("yrbs")

Format
A data frame with 15624 observations on the following 4 variables.

weight sampling weights
stratum sampling strata
psu primary sampling units
qn8 1=Yes, 2=No

Source

References
Centers for Disease Control and Prevention (2016) Software for Analysis of YRBS Data. [CRAN doesn’t believe the URL is valid]

Examples
data(yrbs)

yrbs_design <- svydesign(id=~psu, weight=~weight, strata=~stratum, data=yrbs)
yrbs_design <- update(yrbs_design, qn8yes=2-qn8)

ci <- svyciprop(~qn8yes, yrbs_design, na.rm=TRUE, method="xlogit")

## to print more digits: matches SUDAAN and SPSS exactly, per table 3 of reference
coef(ci)
SE(ci)
attr(ci,"ci")
Index

* algebra
  paley, 50
* category
  svytable, 137
* datasets
  api, 6
  crowd, 26
  election, 28
  fpc, 31
  hospital, 35
  mu284, 39
  myco, 40
  nhanes, 44
  salamander, 61
  scd, 62
  yrbs, 154
* distribution
  pchisqsum, 52
* generalized linear mixed model
  salamander, 61
* hplot
  barplot.svystat, 13
  svycdf, 83
  svycoplot, 89
  svyhist, 104
  svyplot, 118
  svyprcomp, 120
  svysmooth, 132
* htest
  svyranktest, 124
  svytable, 137
  svytest, 140
* manip
  as.fpc, 9
  as.svysurvey, 12
  calibrate, 18
  compressWeights, 24
  dimnames.DBIsvydesign, 27
  estweights, 29
  ftable.svystat, 32
  nonresponse, 45
  postStratify, 54
  rake, 57
  subset.survey.design, 69
  svyby, 80
  svydesign, 95
  update.survey.design, 145
* models
  se, 63
  svymle, 112
* multivariate
  svyfactanal, 98
  svyprcomp, 120
* optimize
  svymle, 112
* regression
  anova.svylm, 3
  psrsq, 56
  regTermTest, 59
  svy.varcoef, 79
  svycoxph, 90
  svyglm, 99
  svypredmeans, 122
* survey
  anova.svylm, 3
  as.fpc, 9
  as.svrepsdesign, 10
  as.svydesign2, 12
  barplot.svystat, 13
  bootweights, 14
  brrweights, 15
  calibrate, 18
  compressWeights, 24
  confint.svyclm, 25
  dimnames.DBIsvydesign, 27
  election, 28
  estweights, 29
  ftable.svystat, 32
hadamard, 33
HR, 36
make.calfun, 37
marginpred, 38
newsyquantile, 41
nonresponse, 45
oldsvyquantile, 46
open.DBIsvydesign, 49
paley, 50
pchisqsum, 52
postStratify, 54
psrsq, 56
rake, 57
stratsample, 68
subset.survey.design, 69
surveyoptions, 70
surveysummary, 71
svrepdesign, 75
svrVar, 78
svy.varcoef, 79
svyby, 80
svycdf, 83
svyciprop, 85
svycontrast, 87
svycoplot, 89
svyCprod, 90
svyCprod, 92
svycralpha, 94
svydesign, 95
svyfactanal, 98
svyglm, 99
svyhist, 104
svyivreg, 105
svykappa, 106
svykm, 107
svyloglin, 109
svylogrank, 111
svymle, 112
svynls, 116
svyolr, 117
svyplot, 118
svyprcomp, 120
svypredmeans, 122
svyqplot, 123
svyranktest, 124
svyratio, 126
svyrecreg, 129
svysmooth, 132
svystandardize, 134
svysurvreg, 136
svytable, 137
svyttest, 140
trimWeights, 141
twophase, 142
update.survey.design, 145
weights.survey.design, 146
with.svyimputationList, 147
withPV.survey.design, 148
withReplicates, 149
* survival
  svycoxph, 90
  svykm, 107
  svylogrank, 111
  svysurvreg, 136
* univar
  newsyquantile, 41
  oldsvyquantile, 46
  surveysummary, 71
  svydesign, 95
* utilities
  svyCprod, 92
  .svycheck(as.svydesign2), 12
  [.nonresponse(nonresponse), 45
  [.repweights_compressed
    (compressWeights), 24
  [.survey.design(subset.survey.design), 69
  [.svyrep.design(svrepdesign), 75
  [.twophase(twophase), 142
AIC.svycoxph(svycoxph), 90
AIC.svyglm, 57, 91
AIC.svyglm(anova.svyglm), 3
anova, 3, 59, 60
anova.svycoxph(anova.svyglm), 3
anova.svyglm, 3, 132
anova.svyloglin(svyloglin), 109
api, 6
apiclass1(api), 6
apiclass2(api), 6
apipop(api), 6
apisrs(api), 6
apistrat(api), 6
approxfun, 47
as.fpc, 9, 129, 147
as.matrix.repweights(compressWeights), 24
INDEX

as.matrix.repweights_compressed (compressWeights), 24
as.svrepdesign, 10, 13, 17, 24, 73, 77, 79, 97, 151
as.svystat, 157
as.vector.repweights_compressed (compressWeights), 24
barplot, 13
barplot.svrepstat (barplot.svystat), 13
barplot.svby (barplot.svystat), 13
barplot.svystat, 13
BIC.svyglm (anova.svyglm), 3
binom.test, 85
biplot, 121
biplot.prcomp, 121
biplot.svyrprcomp (svyprcomp), 120
bootstratum (bootweights), 14
bootweights, 11, 14, 77
brrweights, 11, 15, 34, 77, 79
bxp, 104
cal.linear (make.calfun), 37
cal.logit (make.calfun), 37
cal.raking (make.calfun), 37
cal.sinh (make.calfun), 37
cal_names (calibrate), 18
calibrate, 18, 30, 37, 38, 55, 58, 101, 102, 128, 129, 141, 144
chisq.test, 103
close, 77, 97
close.DBIsvydesign (open.DBIsvydesign), 49
test, 59
test (surveysummary), 71
test (svyby), 80
test (svyglm (svyglm), 99
test (svyloglin (svyloglin), 109
test (svymle (svymle), 112
test (svyratio (svyratio), 126
test (svystat (surveysummary), 71
test (compressWeights), 24, 55, 58
test, 25
test (surveysummary), 71
test (svyby (svyby), 80
test (svyglm, 25
test (svykm (svykm), 107
test (svyratio (svyratio), 126
test (svystat, 85

confint.svrepstat (surveysummary), 71
confint.svystat (svytest), 140
confint.svystat (svytest), 140
contrasts, 60
coxph, 91
crowd, 26
cv (surveysummary), 71
deff (surveysummary), 71
deff.svby (svyby), 80
degf, 47, 73, 77, 99, 101
degf (svytable), 137
deriv, 87
dim.DBIsvydesign
 (dimnames.DBIsvydesign), 27
dim.repweights_compressed
 (compressWeights), 24
dim.survey.design
 (dimnames.DBIsvydesign), 27
dim.svyimputationList
 (dimnames.DBIsvydesign), 27
dim.svyrep.design
 (dimnames.DBIsvydesign), 27
dim.twophase (dimnames.DBIsvydesign), 27
dimnames.DBIsvydesign, 27
dimnames.repweights_compressed
 (compressWeights), 24
dimnames.survey.design
 (dimnames.DBIsvydesign), 27
dimnames.svyimputationList
 (dimnames.DBIsvydesign), 27
dimnames.svyrep.design
 (dimnames.DBIsvydesign), 27
dimnames.twophase
 (dimnames.DBIsvydesign), 27
dnorm, 113
dotchart (barplot.svystat), 13
election, 28, 36, 97
election_insamp (election), 28
election_jointHR (election), 28
election_jointprob (election), 28
election_pps (election), 28
estWeights, 20, 144
estWeights (estweights), 29
estweights, 29
extractAIC.svrepglm (anova.svyglm), 3
extractAIC.svycoxph (svycoxph), 90
extractAIC.svyglm (anova.svyglm), 3
election, 28, 36, 97
election_insamp (election), 28
election_jointHR (election), 28
election_jointprob (election), 28
election_pps (election), 28
estWeights, 20, 144
estWeights (estweights), 29
estweights, 29
extractAIC.svrepglm (anova.svyglm), 3
extractAIC.svycoxph (svycoxph), 90
extractAIC.svyglm (anova.svyglm), 3
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>factanal</td>
<td>98, 99</td>
</tr>
<tr>
<td>fpc</td>
<td>31</td>
</tr>
<tr>
<td>ftable</td>
<td>33</td>
</tr>
<tr>
<td>ftable.svrepsstat</td>
<td>32</td>
</tr>
<tr>
<td>ftable.svby</td>
<td>82</td>
</tr>
<tr>
<td>ftable.svby.svrepsstat</td>
<td>32</td>
</tr>
<tr>
<td>glm</td>
<td>79, 102</td>
</tr>
<tr>
<td>grake</td>
<td>18</td>
</tr>
<tr>
<td>hadamard</td>
<td>16, 17, 33, 50, 51</td>
</tr>
<tr>
<td>hist</td>
<td>104</td>
</tr>
<tr>
<td>hospital</td>
<td>35</td>
</tr>
<tr>
<td>HR</td>
<td>36, 95</td>
</tr>
<tr>
<td>image</td>
<td>76</td>
</tr>
<tr>
<td>image.svreps.design</td>
<td>75</td>
</tr>
<tr>
<td>interaction</td>
<td>72</td>
</tr>
<tr>
<td>is.hadamard</td>
<td>50</td>
</tr>
<tr>
<td>ivreg</td>
<td>106</td>
</tr>
<tr>
<td>jk1weights</td>
<td>77, 79</td>
</tr>
<tr>
<td>jk1weights.brrweights</td>
<td>15</td>
</tr>
<tr>
<td>jknweights</td>
<td>24, 77, 79</td>
</tr>
<tr>
<td>jknweights.brrweights</td>
<td>15</td>
</tr>
<tr>
<td>joinCells</td>
<td>(nonresponse)</td>
</tr>
<tr>
<td>lines.svykm</td>
<td>107</td>
</tr>
<tr>
<td>lines.svysmooth</td>
<td>132</td>
</tr>
<tr>
<td>make.calfun</td>
<td>19, 21, 37</td>
</tr>
<tr>
<td>make.formula</td>
<td>(surveysummary)</td>
</tr>
<tr>
<td>make.panel.svysmooth</td>
<td>132</td>
</tr>
<tr>
<td>marginpred</td>
<td>38, 143</td>
</tr>
<tr>
<td>model.frame.svreps.design</td>
<td>75</td>
</tr>
<tr>
<td>model.frame.twophase</td>
<td>(twophase) 142</td>
</tr>
<tr>
<td>mrbweights</td>
<td>11</td>
</tr>
<tr>
<td>mrbweights.bootweights</td>
<td>14</td>
</tr>
<tr>
<td>mu284</td>
<td>39</td>
</tr>
<tr>
<td>multistage.svyrecvar</td>
<td>129</td>
</tr>
<tr>
<td>multistage.phase1.twophase</td>
<td>142</td>
</tr>
<tr>
<td>multistage.rcpp.svyrecvar</td>
<td>129</td>
</tr>
<tr>
<td>myco</td>
<td>40</td>
</tr>
<tr>
<td>na.exclude.survey.design</td>
<td>(svydesign) 95</td>
</tr>
<tr>
<td>na.exclude.twophase</td>
<td>(twophase) 142</td>
</tr>
<tr>
<td>na.fail.survey.design</td>
<td>(svydesign) 95</td>
</tr>
<tr>
<td>na.fail.twophase</td>
<td>(twophase) 142</td>
</tr>
<tr>
<td>na.omit.survey.design</td>
<td>(svydesign) 95</td>
</tr>
<tr>
<td>na.omit.twophase</td>
<td>(twophase) 142</td>
</tr>
<tr>
<td>neighbours</td>
<td>(nonresponse) 45</td>
</tr>
<tr>
<td>newsvyquantile</td>
<td>41</td>
</tr>
<tr>
<td>nhanes</td>
<td>44</td>
</tr>
<tr>
<td>nlm</td>
<td>113</td>
</tr>
<tr>
<td>nls</td>
<td>116</td>
</tr>
<tr>
<td>nonresponse</td>
<td>45</td>
</tr>
<tr>
<td>oldsvyquantile</td>
<td>43, 46</td>
</tr>
<tr>
<td>onestage.svyCprod</td>
<td>92</td>
</tr>
<tr>
<td>onestage.phase1.twophase</td>
<td>142</td>
</tr>
<tr>
<td>onestat</td>
<td>(svyCprod) 92</td>
</tr>
<tr>
<td>onestat.phase1.twophase</td>
<td>142</td>
</tr>
<tr>
<td>open</td>
<td>77, 97</td>
</tr>
<tr>
<td>open DBIsvydesign</td>
<td>49</td>
</tr>
<tr>
<td>optim</td>
<td>113</td>
</tr>
<tr>
<td>paley</td>
<td>34, 50</td>
</tr>
<tr>
<td>panel.smooth</td>
<td>133</td>
</tr>
<tr>
<td>par</td>
<td>107</td>
</tr>
<tr>
<td>pchisq</td>
<td>53</td>
</tr>
<tr>
<td>pchisqsum</td>
<td>5, 52, 60, 103, 110, 131, 138</td>
</tr>
<tr>
<td>pFSum</td>
<td>(pchisqsum) 52</td>
</tr>
<tr>
<td>plot</td>
<td>119</td>
</tr>
<tr>
<td>plot.lm</td>
<td>133</td>
</tr>
<tr>
<td>plot.stepfun</td>
<td>84</td>
</tr>
<tr>
<td>plot.svycdf</td>
<td>83</td>
</tr>
<tr>
<td>plot.svykm</td>
<td>(svykm) 107</td>
</tr>
<tr>
<td>plot.svykm.svykm</td>
<td>(svykm) 107</td>
</tr>
<tr>
<td>plot.svykm.svykm.svykm</td>
<td>132</td>
</tr>
<tr>
<td>poisson_sampling</td>
<td>53</td>
</tr>
<tr>
<td>postStratify</td>
<td>21, 30, 54, 58, 93, 94, 129, 135</td>
</tr>
<tr>
<td>ppscov</td>
<td>53, 54</td>
</tr>
<tr>
<td>ppscov (HR)</td>
<td>36</td>
</tr>
<tr>
<td>ppsmat</td>
<td>95, 96</td>
</tr>
<tr>
<td>ppsmat (HR)</td>
<td>36</td>
</tr>
<tr>
<td>prcomp</td>
<td>121</td>
</tr>
<tr>
<td>predict.coxph</td>
<td>91</td>
</tr>
<tr>
<td>predict.svrepglm</td>
<td>(svyglm) 99</td>
</tr>
<tr>
<td>predict.svycoxph</td>
<td>38, 108</td>
</tr>
<tr>
<td>predict.svycoxph.svyglm</td>
<td>90</td>
</tr>
<tr>
<td>predict.svyglm</td>
<td>(svyglm) 99</td>
</tr>
<tr>
<td>predict.svyolr</td>
<td>(svyolr) 117</td>
</tr>
<tr>
<td>predict.svyratio</td>
<td>(svyratio) 126</td>
</tr>
<tr>
<td>predict.svyratio.separate</td>
<td>(svyratio) 126</td>
</tr>
<tr>
<td>print.anova.svyloglin</td>
<td>(svyloglin) 109</td>
</tr>
<tr>
<td>print.nonresponse</td>
<td>(nonresponse) 45</td>
</tr>
</tbody>
</table>
print.nonresponseSubset (nonresponse), 45
print.regTermTest (regTermTest), 59
print.summary.svrep.design (svrepdesign), 75
print.summary.svytable (svytable), 137
print.summary.twophase (twophase), 142
print.svycdf (svycdf), 83
print.svymle (svymle), 112
print.svyquantile (oldsvyquantile), 46
print.svyratio (svyratio), 126
print.svyratio_separate (svyratio), 126
print.svrep.design (svrepdesign), 75
print.svysmooth (svysmooth), 132
print.twophase (twophase), 142
psrsq, 56

qnorm, 124
qqplot, 124
qr, 137
quantile, 42, 43, 108, 124
quantile.svykm (svykm), 107

rake, 19, 21, 54, 55, 57
regTermTest, 4, 5, 59, 87, 91, 102, 118, 132, 139
residuals.svrepglm (svyglm), 99
residuals.svyglm (svyglm), 99

salamander, 61, 153
sample, 68
scd, 62
SE, 63, 73
SE.svyby (svyby), 80
SE.svyratio (svyratio), 126
smoothArea, 64
smoothUnit, 66
sparseCells (nonresponse), 45
stepfun, 84
stratsample, 68
subbootweights, 11
subbootweights (bootweights), 14
subset.survey.design, 69, 96, 97
subset.svymputationList (with.svymputationList), 147
subset.svrep.design (subset.survey.design), 69
subset.twophase (twophase), 142
summary.svrepglm (svyglm), 99

summary.svreptable (svytable), 137
summary.svyglm (svyglm), 99
summary.svymlm (svymlm), 112
summary.svrep.design (svrepdesign), 75
summary.svytable (svytable), 137
summary.twophase (twophase), 142
survey.adjust.domain.lonely (surveyoptions), 70
survey.drop.replicates (surveyoptions), 70
survey.lonely.psu (surveyoptions), 70
survey.multicore (surveyoptions), 70
survey.replicates.mse (surveyoptions), 70
survey.ultimate.cluster (surveyoptions), 70
survey.use_rcpp (surveyoptions), 70
survey.want.obsolete (surveyoptions), 70
surveyoptions, 17, 70, 94
surveysummary, 71
svrepdesign, 11, 73, 75, 79, 100, 146, 147, 150, 151
svytable (svytable), 137
svYVar, 17, 78, 130, 151
svy.varcoef, 79
svyboxplot (svyhist), 104
svyby, 47, 72, 80, 135, 139
svybyx (svyby), 80
svycdf, 83
svychisq, 103, 110
svychisq (svytable), 137
svyciprop, 42, 48, 73, 85
svycontrast, 73, 81, 87, 106
svycoplot, 89
svycoxph, 3, 4, 90, 112
svyCprod, 79, 92, 97, 130
svyralpha, 94
svydesign, 10–12, 36, 50, 54, 69, 73, 77, 79, 94, 95, 100, 114, 128, 142, 143, 146, 147
svyfactanal, 98
svyglm, 3, 4, 25, 56, 72, 79, 99, 110, 114, 118, 122, 123, 140
svylogchisq, 103
svyhist, 84, 104, 133
svyivreg, 105
svykappa, 106
svyk, 48, 91, 107, 112
svyloglin, 109, 139
svylogrank, 111, 125
svymean, 86, 110, 122, 126, 128, 138–140
svymean (surveysummary), 71
svymle, 112, 116
svynls, 116
svyolr, 117
svyplot, 89, 105, 118
svyprcomp, 120
svypredmeans, 38, 122
svyqqmath (svyqqplot), 123
svyqqplot, 123
svyquantile, 46, 73, 84, 123
svyquantile (newsvyquantile), 41
svyranktest, 124
svyratio, 72, 126
svyrecvar, 10, 12, 70, 92, 94, 97, 129, 143
svyrecvar.phase1 (twophase), 142
svyscoretest, 131
svysmooth, 132
svysmoothArea (smoothArea), 64
svysmoothUnit (smoothUnit), 66
svystandardize, 134
svysurveysreg, 136
svytable, 13, 82, 119, 137
svytotal, 103, 138, 139
svytotal (surveysummary), 71
svyttest, 73, 102, 125, 140
svyvar, 99
svyvar (surveysummary), 71
symbols, 119
t.test, 140
table, 55
termplot, 133
transform, 146
trimWeights, 20, 21, 141
twophase, 21, 30, 142, 146
twophase2var (twophase), 142
twophasevar (twophase), 142
unwtd.count (svyby), 80
update.DBIsvydesign, 27
update.DBIsvydesign (update.survey.design), 145
update.survey.design, 82, 97, 145
update.svyloglin (svyloglin), 109
update.svyrep.design (update.survey.design), 145
update.twophase (update.survey.design), 145
vcov, 59, 60, 63, 64
vcov.svreseqstat (surveysummary), 71
vcov.svyglm (svyglm), 99
vcov.svymle (svymle), 112
vcov.svyrep.design (withReplicates), 149
vcov.svyrepstat (surveysummary), 71
weights.nonresponse (nonresponse), 45
weights.survey.design, 146
weights.survey_fpc (weights.survey.design), 146
weights.svyrep.design (weights.survey.design), 146
weights.svyrep.stat (weights.survey.design), 146
with.svyimputationList, 27, 97, 147, 149
withPV.survey.design, 148
withReplicates, 97, 149
xdesign, 152
xtabs, 55
xyplot, 89
yrbs, 86, 154