Package 'capm'

August 29, 2016

Type Package

Title Companion Animal Population Management

Depends R (>= 3.2.2)

Imports deSolve, FME, survey, ggplot2, reshape2, shiny, grid, rgdal, maptools, sp

Description Quantitative analysis to support companion animal population management.

License GPL (>= 2)

LazyLoad yes

URL http://oswaldosantos.github.io/capm

Version 0.9.1

Date 2016-03-06

RoxygenNote 5.0.1

NeedsCompilation no

Author Oswaldo Santos Baquero [aut, cre],
Marcos Amaku [ctb],
Fernando Ferreira [ctb]

Maintainer Oswaldo Santos Baquero <oswaldosant@gmail.com>

Repository CRAN

Date/Publication 2016-03-07 01:01:41

R topics documented:

`capm-package` *The capm Package*

Description

Companion Animal Population Management. Provides functions for quantitative Companion Animal Population Management. Further information can be found in the URL given below.

Details

Package:	capm
Type:	Package
Version:	0.9.1
Date:	2016-03-06
Depends:	R (>= 3.2.2)
Imports:	deSolve, FME, survey, reshape2, ggplot2, shiny, grid, rgdal, maptools, sp
License:	GPL (>= 2)
LazyLoad:	yes
URL:	http://oswaldosantos.github.io/capm
Author:	Oswaldo Santos Baquero <oswaldosant@gmail.com>
Maintainer:	Oswaldo Santos Baquero <oswaldosant@gmail.com>
Contributors:	Marcos Amaku <amaku@vps.fmvz.usp.br>, Fernando Ferreira <fernando@vps.fmvz.usp.br>

`Calculate2StageSampleSize` *Two-stage cluster sampling size and composition*

Description

Calculates sample size and composition for a two-stage cluster sampling design to estimate a total.

Usage

```
Calculate2StageSampleSize(psu.ssu = NULL, psu.x = NULL, conf.level = 0.95,
  error = 0.1, cost = 4, minimum.ssu = 15)
```

Arguments

`psu.ssu`	`data.frame` with all primary sampling units (PSU). First column contains PSU unique identifiers. Second column contains `numeric` PSU sizes.
`psu.x`	`data.frame`. Each row corresponds to a secondary sampling unit (SSU) surveyed in a pilot study. First column contains the PSU identifiers to which the ssu belongs to. Second column contains the totals observed in the ssu and must be `numeric`.
`conf.level`	the confidence level required. It must be `numeric` between 0 and 1 inclusive.
`error`	the maximum relative difference between the estimate and the unknown population value. It must be `numeric` between 0 and 1 inclusive.
`cost`	the ratio of the cost of sampling a PSU to the cost of sampling a SSU.
`minimum.ssu`	integer to define the minimum number of SSU to be selected per PSU. If the calculated number of SSU to be selected is lesser than `minimum.ssu`, it is redefined as `minimum.ssu`. To avoid any lower threshold, define `minimum.ssu` as equal to 0.

Details

It is assumed that psu from the pilot are selected with probability proportional to size (PPS) and with replacement. ssu are assumed to be selected via simple (systematic) random sampling.

PSU must have the same identifiers in `psu.ssu` and in `psu.x`.

Value

Matrix with the sample size and composition and with variability estimates.

References

Levy P and Lemeshow S (2008). Sampling of populations: methods and applications, Fourth edition. John Wiley and Sons, Inc.

http://oswaldosantos.github.io/capm

Examples

```
# Load data with psu identifiers and sizes.
data(psu.ssu)

# Load data from a pilot sample.
data(pilot)

# Calculate sample size and composition.
(sample.sc <- Calculate2StageSampleSize(psu.ssu, pilot, conf.level = 0.95, error = 0.1, cost = 4))
```

`CalculateGlobalSens` *Global sensitivity analysis*

Description

Wraper for `sensRange` function, which calculates population size sensitivities, to parameters used in one of the following functions: `SolveIASA`, `SolveSI` or `SolveTC`.

Usage

```
CalculateGlobalSens(model.out = NULL, ranges = NULL, sensv = NULL,
  all = FALSE)
```

Arguments

`model.out`	an output from one of the previous function or a `list` with equivalent structure.
`ranges`	output from the `SetRanges` function, applied to the `pars` argument used in the function previously specified in `model.out`.
`sensv`	string with the name of the the output variables for which the sensitivity needs to be estimated.
`all`	logical. If `FALSE`, sensitivity ranges are calculated for each parameter. If `TRUE`, sensitivity ranges are calculated for the combination of all aparameters.

Details

When `all` is equal to TRUE, `dist` argument in `sensRange` is defined as "latin" and when equal to `FALSE`, as "grid". The `num` argument in `sensRange` is defined as 100.

Value

A `data.frame` (extended by `summary.sensRange` when `all == TRUE`) containing the parameter set and the corresponding values of the sensitivity output variables.

References

Soetaert K and Petzoldt T (2010). Inverse modelling, sensitivity and monte carlo analysis in R using package FME. Journal of Statistical Software, 33(3), pp. 1-28.

Reichert P and Kfinsch HR (2001). Practical identifiability analysis of large environmental simulation models. Water Resources Research, 37(4), pp.1015-1030.

http://oswaldosantos.github.io/capm

See Also

sensRange.

Examples

```
#####################
## SolveIASA model ##
#####################

## Parameters and intial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solve.iasa.pt <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:15, method = 'rk4')

## Set ranges 10 % greater and lesser than the
## point estimates.
rg.solve.iasa <- SetRanges(pars = pars.solve.iasa)

## Calculate golobal sensitivity of combined parameters.
## To calculate global sensitivity to each parameter, set
## all as FALSE.
glob.all.solve.iasa <- CalculateGlobalSens(
  model.out = solve.iasa.pt,
  ranges = rg.solve.iasa,
  sensv = 'n2', all = TRUE)
```

`CalculateLocalSens` *Local sensitivity analysis*

Description

Wraper for `sensFun` function, which estimates local effect of all model parameters on population size, applying the so-called sensitivity functions. The set of parameters used in any of the following functions can be assessed: `SolveIASA`, `SolveSI` or `SolveTC`.

Usage

```
CalculateLocalSens(model.out = NULL, sensv = "n")
```

Arguments

`model.out`	an output from one of the previous functions or a `list` with equivalent structure.
`sensv`	string with the name of the output variables for which sensitivity needs to be estimated.

Details

For further arguments of `sensFun`, defaults are used. See the help page of this function for details. Methods for class "sensFun" can be used.

Value

a `data.frame` of class `sensFun` containing the sensitivity functions. There is one row for each sensitivity variable at each independent time. The first column `x`, contains the time value; the second column `var`, the name of the observed variable; and remaining columns have the sensitivity parameters.

References

Soetaert K and Petzoldt T (2010). Inverse modelling, sensitivity and monte carlo analysis in R using package FME. Journal of Statistical Software, 33(3), pp. 1-28.

Reichert P and Kfinsch HR (2001). Practical identifiability analysis of large environmental simulation models. Water Resources Research, 37(4), pp.1015-1030.

http://oswaldosantos.github.io/capm

See Also

`sensRange`.

Examples

```
####################
## SolveIASA model ##
####################

## Parameters and intial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solve.iasa.pt <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:15, method = 'rk4')

## Calculate local sensitivities to all parameters.
local.solve.iasa2 <- CalculateLocalSens(
  model.out = solve.iasa.pt, sensv = 'n2')
local.solve.iasa1 <- CalculateLocalSens(
  model.out = solve.iasa.pt, sensv = 'n1')
```

`CalculatePopChange` *Population change.*

Description

Calculate the change in population size between two times. When only one time is specified, the population size at that time is returned.

Usage

```
CalculatePopChange(model.out = NULL, variable = NULL, t1 = NULL,
  t2 = NULL, ratio = TRUE)
```

Arguments

`model.out`	an output from one of the following functions or a `list` with equivalent structure: `SolveIASA`, `SolveSI` or `SolveTC`.

variable	string with the name of the the output variable for which the change needs to be calculated (see the variable argument for PlotModels.
t1	value specifying the first time.
t2	value specifying the second time.
ratio	logical. When TRUE, the calculated change is based on poulation size at t2 divided by population size at t1. When FALSE, the calculated change is based on poulation size at t2 minus population size at t1.

Value

Value representing the ratio (if ratio is TRUE) or the difference (if ratio is FALSE) between population size at time t2 and t1. If only one time is specified, the value is the population size at that time.

References

http://oswaldosantos.github.io/capm

Examples

```
###################
## SolveSI model ##
###################

# Parameters and initial conditions.
pars.solve.si = c(b = 0.245, d = 0.101,
                  k = 98050, s = 0.048)
init.solve.si = c(n = 89137, q = 0.198)

# Solve for a specific sterilization rate.
solve.si.pt = SolveSI(pars = pars.solve.si,
                              init = init.solve.si,
                              time = 0:15, dd = 'b',
                              im = 100, method = 'rk4')

# Calculate the population change (ratio) between times 0 and 15.
CalculatePopChange(solve.si.pt, variable = 'n', t2 = 15, t1 = 0)

# Calculate the population change (difference) between times 0 and 15.
CalculatePopChange(solve.si.pt, variable = 'n', t2 = 15,
                   t1 = 0, ratio = FALSE)

# Calculate the population zises at time 15.
CalculatePopChange(solve.si.pt, variable = 'n', t2 = 15)
```

`CalculateSimpleSampleSize` *Simple random sample size*

Description

Calculates sample size for a simple sampling design to estimate a total.

Usage

```
CalculateSimpleSampleSize(x = NULL, N = NULL, conf.level = 0.95,
  error = 0.1)
```

Arguments

`x`	`vector` pilot sample of the variable to be estimated. If x is a scalar, it is used as the relative variance of the variable to be estimated (`((N - 1) / N * sd(x)^2) / mean(x)^2`).
`N`	`numeric`. indicating the number of sampling units in the population.
`conf.level`	the confidence level required. It must be `numeric` between 0 and 1 inclusive.
`error`	the maximum relative difference between the estimate and the unknown population value. It must be `numeric` between 0 and 1 inclusive.

Value

numeric sample size rounded up to nearest integer.

References

Levy P and Lemeshow S (2008). Sampling of populations: methods and applications, Fourth edition. John Wiley and Sons, Inc.

http://oswaldosantos.github.io/capm

Examples

```
# Using a pilot sample from a population with 10000 sampling units.
pilot.sample <- rpois(50, 0.8)
CalculateSimpleSampleSize(x = pilot.sample, N = 10000,
                          conf.level = 0.95, error = 0.1)

# Using expected mean and standard deviation for a population
# with 10000 sampling units.
mean.x <- 0.98
sd.x <- 1.02
N <- 10000
V <- ((N - 1) / N * sd.x^2) / mean.x^2
CalculateSimpleSampleSize(x = V, N = 10000, conf.level = 0.95, error = 0.1)
```

`CalculateStratifiedSampleSize` *Stratified random sample size*

Description

Calculates sample size for a stratified random sampling design to estimate a total.

Usage

```
CalculateStratifiedSampleSize(strata = NULL, x = NULL, conf.level = 0.95,
  error = 0.1)
```

Arguments

`strata`	`vector`, `matrix` or `data.frame`. If vector, the length must be equal to the number of sampling units in the population and each element represent the strata memebership of each sampling unit. If matrix or data.frame, first column represent the size of each strata, second column represent the expected mean in each strata and third column represent the expected variance in each strata. Each row is a stra and must be named.
`x`	`data.frame` representing a pilot sample. First column has the variable to be estimated and second column has the strata membership of each observation.
`conf.level`	the confidence level required. It must be `numeric` between 0 and 1 inclusive.
`error`	the maximum relative difference between the estimate and the unknown population value. It must be `numeric` between 0 and 1 inclusive.

Value

numeric sample size rounded up to nearest integer.

References

Levy P and Lemeshow S (2008). Sampling of populations: methods and applications, Fourth edition. John Wiley and Sons, Inc.

http://oswaldosantos.github.io/capm

Examples

```
# Using a pilot sample from a population with 10000 sampling units.
strata <- rep(c('Rural', 'Urban'), c(100, 9900))
pilot.sample <- data.frame(c(rpois(5, 1.3), rpois(45, 0.8)),
                           rep(c('Rural', 'Urban'), c(5, 45)))
CalculateStratifiedSampleSize(strata, pilot.sample)

# Using expected mean and variance for a population with
# 10000 sampling units.
```

```
str.n <- c(Rural = 100, Urban = 9900)
str.mean <- c(Rural = 1.4, Urban = 0.98)
str.var <- c(Rural = 1.48, Urban = 1.02)
CalculateStratifiedSampleSize(cbind(str.n, str.mean, str.var))
```

`DesignSurvey` *Survey design*

Description

A wraper for `svydesign` function from the survey package, to define one of the following survey designs: two-stage cluster, simple (systematic) or stratified. In the first case, weights are calculated considering a probability proportional to size sampling with replacement for the first stage and a simple random sampling for the second stage. Finite population correction is specified as the population size for each level of sampling.

Usage

```
DesignSurvey(sample = NULL, psu.ssu = NULL, psu.col = NULL,
  ssu.col = NULL, psu.2cd = NULL, N = NULL, strata = NULL, ...)
```

Arguments

`sample`	`data.frame` with sample observations. for two-stage cluster designs, one of the columns must contain unique identifiers for PSU and another column must contain unique identifiers for Secondary Sampling Units (SSU).
`psu.ssu`	`data.frame` with all Primary Sampling Units (PSU). First column contains PSU unique identifiers. Second column contains `numeric` PSU sizes. It is only used for two-stage cluster designs.
`psu.col`	the column of `sample` containing the psu identifiers (for two-stage cluster designs). It is only used for two-stage cluster designs.
`ssu.col`	the column of `sample` containing the ssu identifiers (for two-stage cluster designs). It is only used for two-stage cluster designs.
`psu.2cd`	value indicating that the survey is a two-stage cluster design and the number of psu included (for psu included more than once, each must be counted).
`N`	for simple designs, a `numeric` value representing the total of sampling units in the population. for a stratified design, it is a column of `sample` indicating, for each observation, the total of sampling units in its respective strata. `N` is ignored in two-stage cluster designs.
`strata`	for stratified designs, a column of `sample` indicating the strata memebership of each observation.
`...`	further arguments passed to `svydesign` function.

Details

For two-stage cluster designs, a PSU appearing in both `psu.ssu` and in `sample` must have the same identifier. SSU identifiers must be unique but can appear more than once if there is more than one observation per SSU. `sample` argument must have just the varibles to be estimated plus the variables required to define the design (two-stage cluster or stratified).

Value

An object of class survey.design.

References

Lumley, T. (2011). Complex surveys: A guide to analysis using R (Vol. 565). Wiley.

http://oswaldosantos.github.io/capm

Examples

```
# Load data with PSU identifiers and sizes.
data(psu.ssu)

# Load data with sample data.
data(survey.data)

## Specify a two-stage cluster design that included 20 PSU.
DesignSurvey(sample = survey.data, psu.ssu = psu.ssu,
             psu.col = 2, ssu.col = 1, psu.2cd = 20)

## Assuming that survey.sampling is a simple design.
DesignSurvey(sample = survey.data, N = 144600)

## Assuming that survey.sampling is a stratified design.
# Hypothetical strata
strat <- survey.data
strat$strat <- 'Urban'
strat$strat[round(runif(5, 1, nrow(strat)))] <- 'Rural'
strat$strat.size <- 144000
strat$strat.size[strat$strat == 'Rural'] <- 600
DesignSurvey(strat, N = 'strat.size', strata = 'strat')
```

`GraphicInterface` *Graphic interface to use some capm functions*

Description

Graphic interface to use some capm functions

Usage

```
GraphicInterface(set.func)
```

Arguments

set.func	string to select a graphic interface for a set of functions to achieve a specific task. SelectSamplingUnits creates a graphic interface to select sampling units for pilot or final survey designs. CalculateSampleSize creates a graphic interface to calculate sample size and composition. The graphic interface created by SurveyAnalysis support functionality to analyise survey data. SolveIASA creates an interface to simulate population dynamics and to assess population-based interventions.

Details

The graphic interfaces are created with the shiny package and thus will open in a browser.

Value

a graphic interface in a browser.

References

http://oswaldosantos.github.io/capm

Examples

```
# Uncomment the following line to open the graphic interface in a browser:
# GraphicInterface(set.func = 'SelectSamplingUnits')
```

MapkmlPSU *Creates *.kml files of a subset of polygons from a polygon shapefile*

Description

Subset polygons acording to the matches between a vector and a specified column from a SpatialPolygonsDataFrame.

Usage

```
MapkmlPSU(shape = NULL, psu = NULL, id = NULL, path = ".")
```

Arguments

shape	string with the name of a polygon shapefile or an object of class SpatialPolygonsDataFrame (see examples).
psu	the values to be matched.
id	column of the *.dbf file with the values to be matched against.
path	string indicating the path to the folder containing the shapfile. If the shapefile is in the working directory or if shape argument is a shapefile, path can be ignored.

Details

If there are *.kml files in the working directory, the new created files will overwrite it in case of name matching.

shape must receive a shapefile with appropriate coordinate reference system, otherwise, MapkmlPSU report an error.

Value

*.kml files of the subsetted polygons.

References

http://oswaldosantos.github.io/capm

See Also

readShapeSpatial

Examples

```
# Load data with the polygon identifiers.
data(psu.ssu)

# Take a sample of 10 PSU with probability
# proportional to size with replacement.
(selected.psu <- SamplePPS(psu.ssu, 10, write = FALSE))

## Define shape from shapefile.
shp.path <- system.file('extdata', package="capm")
# The code above used a shapefile avaliable in the
# capm package.
# You might want to write a code like:
# shp.path  <- 'path_to_the_folder_with_the_shapefile'

# Create *kml files of 10 polygons.
# Uncomment the following line to create kml files:
# MapkmlPSU('santos', selected.psu[, 1], 1, shp.path)

## Define the shape argument as an object x of class SpatialPolygonsDataFrame.
```

```
# MapkmlPSU(x, selected.psu[, 1], 1)
```

`pilot` *Pilot study to calculate sample size and composition*

Description

This data set has 5 secondary sampling units (SSU) selected with the `SampleSystematic` function, within 10 primary sampling units (PSU) from a sampling frame of Santos city, selected with the `SamplePPS` function. For each ssu, there is a number of dogs, simulated from a poisson ditribution, with lambda equal to the weighted mean (0.8) of number od fogs per ssu observed in field studies. The pilot is intended to calculate sample size and composition to estimate owned dog population size.

Usage

```
data(pilot)
```

Format

A data frame with 50 observations on the following 2 variables.

`psu` PSU to which ssu belongs to.

`dogs` Number of dogs per SSU.

Examples

```
data(pilot)
str(pilot)
head(pilot)
```

`PlotGlobalSens` *Plot results of GlobalSens function*

Description

Plot results of of `CalculateGlobalSens` function.

Usage

```
PlotGlobalSens(global.out = NULL, x.label = "Time",
  y.label = "Population", legend.label = "Sensitivity range",
  mm.label = "min - max", sd.label = "mean +- sd   ")
```

Arguments

global.out	output from CalculateGlobalSens function.
x.label	string with the name of x axis.
y.label	string with the name of y axis.
legend.label	string with name for the legend.
mm.label	string with the name of the envelope calculated using the minimum and maximum ranges.
sd.label	string with the name of the envelope calculated using the mean +- standard deviation ranges.

Details

Font size of saved plots is usually different to the font size seen in graphic browsers. Before changing font sizes, see the final result in saved (or preview) plots.

Other details of the plot can be modifyed using appropriate functions from ggplot2 package.

References

http://oswaldosantos.github.io/capm

See Also

plot.deSolve.

Examples

```
#####################
## SolveIASA model ##
#####################

## Parameters and intial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solve.iasa.pt <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:15, method = 'rk4')
```

```
## Set ranges 10 % greater and lesser than the
## point estimates.
rg.solve.iasa <- SetRanges(pars = pars.solve.iasa)

## Calculate golobal sensitivity of combined parameters.
## To calculate global sensitivity to each parameter, set
## all as FALSE.
glob.all.solve.iasa <- CalculateGlobalSens(
  model.out = solve.iasa.pt,
  ranges = rg.solve.iasa,
  sensv = 'n2', all = TRUE)

### Plot the sensitivities of combined parameters.
PlotGlobalSens(glob.all.solve.iasa)
```

PlotLocalSens *Plot results of CalculateLocalSens function*

Description

Plot results of the CalculateLocalSens function.

Usage

```
PlotLocalSens(local.out = NULL, x.sens = "Time", y.sens = "Sensitivity",
  y.ind = c("L1", "L2", "Mean", "Min", "Max"), label.size = 10,
  x.axis.angle = 90, type = 1)
```

Arguments

local.out	output from CalculateLocalSens function.
x.sens	string with the name of x axis for sensitivity functions.
y.sens	string with the name of y axis for sensitivity functions.
y.ind	string with the name of y axis for the parameter importance indices.
label.size	a number to specify the size of axes labels and text.
x.axis.angle	a number with angle of rotation for x axis text. Passed to angle argument of element_text.
type	a number to define the type of graphical output. 1: importance index L1; 2: importance index L2; 3: mean of sensitivity functions; 5: minimum of sensitivity functions; and 5: maximum of sensitivity functions; 6: sensitivity functions and all importance indices are ploted.

Details

Font size of saved plots is usually different to the font size seen in graphic browsers. Before changing font sizes, see the final result in saved (or preview) plots.

References

Chang W (2012). R Graphics Cookbook. O'Reilly Media, Inc.

Soetaert K, Cash J and Mazzia F (2012). Solving differential equations in R. Springer.

http://oswaldosantos.github.io/capm

See Also

plot.sensFun.

Examples

```
#####################
## SolveIASA model ##
#####################

## Parameters and intial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solve.iasa.pt <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:15, method = 'rk4')

## Calculate local sensitivities to all parameters.
local.solve.iasa2 <- CalculateLocalSens(
  model.out = solve.iasa.pt, sensv = 'n2')

## Plot local sensitivities
# Uncomment the following line:
# PlotLocalSens(local.solve.iasa2)
```

`PlotModels` *Plot results of capm model functions*

Description

Plot results of one of the following functions: SolveIASA, SolveSI or SolveTC.

Usage

```
PlotModels(model.out = NULL, variable = NULL, col = "red",
  col1 = c("cadetblue1", "yellow", "red"), col2 = c("blue", "darkgreen",
  "darkred"), x.label = "Years", y.label = NULL,
  scenarios.label = "v = (__ * owned carrying capacity)",
  legend.label = NULL, pop = NULL)
```

Arguments

model.out	output of one of the function previously mentioned.
variable	string to specify the variable to be ploted. For SolveSI function: "n" (population size). "q" (proportion of sterilized animals). For SolveIASA function using only point estimates: "f1" (owned intact females). "fs1" (owned sterilized females). "m1" (owned intact males). "ms1" (owned sterilized males). "f2" (stray intact females). "fs2" (stray sterilized females). "m2" (stray intact males). "ms2" (stray sterilized males). "n1" (owned intact animals). "ns1" (owned sterilized animals). "n2" (stray intact animals). "ns2" (stray sterilized animals). "N1" (owned animals). "N2" (stray animals). "N" (total population). For SolveIASA function using *.range arguments: "f" (intact females). "fs" (sterilized females). "m" (intact males). "ms" (sterilized males). "n" (intact animals). "ns" (sterilized animals). "N" (Total population stratified by reproductive status). For SolveTC function: "n" (fertile animals).

"g" (sterilized animals).

"u" (cumulative of sterilized animals)

`col` string indicating the color of ploted line, when `s.range` is `NULL`.

`col1` `character vector` indicating the color of lowest (highest) population sizes (proportion of sterilized animals), when `s.range` is not `NULL`.

`col2` `character vector` indicating the color of highest (lowest) population sizes (proportion of sterilized animals), when `s.range` is not `NULL`.

`x.label` string with the name of x axis.

`y.label` string with the name of y axis.

`scenarios.label`
string with the name of scenarios of `SolveIASA` output, determined by the immigartion rates. Within the string, use the expression __ in the location where you want to appear the value of the immigartion rate. For line breaking, use `\n` (see examples).

`legend.label` string with the name of legend, for plots of `SolveIASA` output.

`pop` value indicating the output of `SolveIASA` to be ploted. When `NULL` (default), plots for owned and stray populations under scenarios created by immigartion rate are created. If `1`, the plots of owned population for the minimum immigartion rate are ploted. When `2`, the plots of stray population for the minimum immigartion rate are ploted. If `3`, the plots of owned population for the maximum immigartion rate are ploted. When `4`, the plots of owned population for the maximum immigartion rate are ploted.

Details

Font size of saved plots is usually different to the font size seen in graphic browsers. Before changing font sizes, see the final result in saved (or preview) plots.

Other details of the plot can be modifyed using appropriate functions from `ggplot2` package.

References

Chang W (2012). R Graphics Cookbook. O'Reilly Media, Inc.

http://oswaldosantos.github.io/capm

See Also

plot.deSolve.

Examples

```
####################
## SolveIASA model ##
####################

## Parameters and intial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
```

```
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solveiasa.pt <- SolveIASA(pars = pars.solve.iasa,
                          init = init.solve.iasa,
                          time = 0:10, method = 'rk4')

# Solve for parameter ranges.
solveiasa.rg <- SolveIASA(pars = pars.solve.iasa,
                          init = init.solve.iasa,
                          time = 0:10,
                          s.range = seq(0, .4, l = 15),
                          a.range = c(0, .2),
                          alpha.range = c(0, .2),
                          v.range = c(0, .1),
                          method = 'rk4')

## Plot stray population sizes using point estimates
# Uncomment the following line:
# PlotModels(solveiasa.pt, variable = "ns2")

## Plot all scenarios and change the label for the scenarios.
# Uncomment the following line:
# PlotModels(solveiasa.rg, variable = 'ns')
```

`PlotPopPyramid` *Population PlotPopPyramid*

Description

Display two opposed horizontal barplots (pyramid).

Usage

```
PlotPopPyramid(dat = NULL, age.col = NULL, sex.col = NULL,
  str.col = NULL, x.label = "Total", stage.label = "Years",
  legend.label = "Sterilized", inner.color = "Gold2",
  outer.color = "DarkOliveGreen", label.size = 13)
```

Arguments

dat	data.frame.
age.col	the number or name of dat column which have a numeric vector representing ages or stage categories.
sex.col	the number or name of dat column which have a factor with two levels representing the sex of individuals (see Details).
str.col	the number of dat column which have a factor representing the reproductive status of individuals (see Details).
x.label	string to be used as a label for x-axis. If non defined, x.label is equal to "Total" (see Details).
stage.label	a string to be used as a label for ages or stages categories. If non defined, stage.label is equal to "Years" (see Details).
legend.label	a string to be used as a label for the legend. If not defined, legend.label is equal to "Sterilized".
inner.color	any valid way to specify colors. When str.col is NULL, inner.color is the color of bars. When str.col is not NULL, innercolor is the inner color of bars. If non defined, inner.color is equal to "Gold2".
outer.color	any valid way to specify colors. When str.col is NULL, outer.color is ignored. When str.col is not NULL, outer.color is the outer color of bars. If non defined, outercolor is equal to "DarkOliveGreen".
label.size	string to define the font size for labels.

Details

PlotPopPyramid is mainly intended for companion animals population pyramids, although it can display other types of opposed bar charts with suitable modification of the arguments.

The bars to the left of 0 on the x axis correspond to the minimum value of as.numeric(dat[, sex.col]). If str.col is not NULL, bars will be stacked, with the minimum value of as.numeric(dat[, str.col]) as their base.

On the top of the plot, it is displayed the total number of observations of each level of dat[, sex.col]. The levels of sex.col are used as labels.

The legend labels are equal to the levels of dat[, str.col].

Font size of saved plots is usually different to the font size seen in graphic browsers. Before changing font sizes, see the final result in saved (or preview) plots.

Other details of the plot can be modifyed using appropriate functions from ggplot2 package (see examples).

Value

Two opposed horizontal barplots.

Note

In companion animals population surveys, some age categories might be empty. One difference between PlotPopPyramid and pryramid.plot is that the first does not drop empty age categories.

References

http://oswaldosantos.github.io/capm

Examples

```
## Load data with information about age, sex and reproductive status of individuals.
data(survey.data)
# Uncomment the following lines:
# PlotPopPyramid(survey.data, age.col = 5, sex.col = 4, str.col = 6)
# PlotPopPyramid(survey.data, age.col = 5, sex.col = 4)
```

---

psu.ssu *Primary and secondary sampling frames from Santos, Brazil.*

Description

This data set list the primary (enumeration units/setores censitarios) and secondary (household/domicilios) sampling units from urban area of Santos city, according to the census of 2010, made by the Instituto Brasileiro de Geografia e Estatistica - IBGE.

Usage

```
data(psu.ssu)
```

Format

A data frame with 649 observations on the following 2 variables.

`psu` Primary sampling units

`ssu` Total number of secondary sampling units per primary sampling unit.

Source

www.ibge.gov.br

Examples

```
data(psu.ssu)
str(psu.ssu)
head(psu.ssu)
```

SamplePPS *Probability proportional to size sampling with replacement*

Description

Select Primary Sampling Units (PSU) with probability proportional to size with replacement.

Usage

```
SamplePPS(psu.ssu = NULL, psu = NULL, write = FALSE, ...)
```

Arguments

psu.ssu	data.frame with all PSU. First column contains PSU unique identifiers. Second column contains numeric PSU sizes.
psu	the number of PSU to be selected.
write	logical. If TRUE, a *.csv file containing the PSU and their Secondary Sampling Units (SSU) is writed in the current working directory.
...	further arguments passed to write.table function.

Value

data.frame. First column contains the selected psu identifiers, coerced by as.character, to avoid scientific notation in case the identifiers be large numbers of class numeric. Second column contain PSU sizes, a variable needed for second stage sampling with SampleSystematic.

References

Levy P and Lemeshow S (2008). Sampling of populations: methods and applications, Fourth edition. John Wiley and Sons, Inc.

http://oswaldosantos.github.io/capm

See Also

SampleSystematic.

Examples

```
# Load data with PSU identifiers and sizes.
data(psu.ssu)

# Take a sample of 10 PSU with probability proportional to size with replacement.
(selected.psu <- SamplePPS(psu.ssu, 10, write = FALSE))
```

SampleSystematic *Simple and stratified systematic sampling*

Description

Select sampling units using simple or stratified systematic samplin. In the context of two-stage cluster sampling, select Secondary Sampling Units (SSU) in one or more Primary Sampling Units (PSU), using systematic sampling.

Usage

```
SampleSystematic(psu.ssu = NULL, su = NULL, N = NULL, write = FALSE,
  ...)
```

Arguments

psu.ssu	data.frame with all PSU. First column contains PSU unique identifiers. Second column contains numeric PSU sizes. Only used for the second stage of a two-stage cluster design (see details).
su	numeric indicating the number of sampling units to be selected. If su has more than one element, stratified sampling is applied and psu.ssu is ignored (see details).
N	numeric indicating the number of sampling units in the population. It is intended for simple or stratified sampling designs and when used, psu.ssu is ignored (see details).
write	logical. If TRUE, a *.csv file containing the PSU and their SSU is writed in the current working directory.
...	further arguments passed to write.table function.

Details

When N is defined, psu.ssu is ignored (not need to be defined). If N has one element, su must too and the result is a simple systematic selection. If N has more than one element, su must have the same number of elements and each oredered pair represent an strata. Thus, when N has more than one element, the result is a stratified sampling with systematic selection within each strata (see examples).

Value

A matrix. For the second stage in a two-stage cluster sampling, the names of columns are the identifiers of selected psu, coerced by as.character to avoid scientific notation in case the identifiers be of class numeric. The rows correspond to the selected SSU within each psu. For simple systematic sampling, the rows correspond to the selected sampling units. For stratified sampling, each column represent an strata and the rows correspond to the selected sampling units in each strata.

References

Levy P and Lemeshow S (2008). Sampling of populations: methods and applications, Fourth edition. John Wiley and Sons, Inc.

http://oswaldosantos.github.io/capm

See Also

SamplePPS.

Examples

```
# Load data with PSU identifiers and sizes.
data(psu.ssu)

# Take a sample of 10 PSU, with probability
# proportional to size and with replacement.
selected.psu <- SamplePPS(psu.ssu, 10, write = FALSE)

# Take a systematic sampling of 5 SSU within each
# PSU of selected.psu.
SampleSystematic(selected.psu, 5, write = FALSE)

# Simple systematic sampling
SampleSystematic(su = 5, N = 100)

# Stratified systematic sampling
SampleSystematic(su = c('Urban' = 50, 'Rural' = 10),
                 N = c('Urban' = 4000, 'Rural' = 150))
```

SetRanges *Parameter ranges for global sensitivity analysis*

Description

Define the minimum and maximum values for parameters whose global sensitivities are to be assesses with CalculateGlobalSens or sensRange functions.

Usage

```
SetRanges(pars = NULL, range = 0.1)
```

Arguments

pars	the same pars vector used in one of the following functions: SolveSI or SolveIASA.
range	scale factor to define the minimum and maximum for each parameter. The default is 0.1, which set the minimum and maximum as 10 percent lesser and greater than the pars values.

Value

data.frame with the complete set of parameter ranges.

References

Soetaert K and Petzoldt T (2010). Inverse modelling, sensitivity and monte carlo analysis in R using package FME. Journal of Statistical Software, 33(3), pp. 1-28.

Reichert P and Kfinsch HR (2001). Practical identifiability analysis of large environmental simulation models. Water Resources Research, 37(4), pp. 1015-1030.

http://oswaldosantos.github.io/capm

See Also

sensRange and SolveSI.

Examples

```
data(psu.ssu)
data(survey.data)

#####################
## SolveIASA model ##
#####################

# Parameters and initial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

# Set ranges 10 % greater and lesser than the
# point estimates.
rg.solve.iasa <- SetRanges(pars.solve.iasa)
```

SolveIASA *Modelling of immigration, abandonment, sterilization and adoption of companion animals*

Description

System of ordinary differential equations to simulate the effect of immigration of owned dogs, abandonment, sterilization of owned and stray dogs and adoption, on population dynamics.

Usage

```
SolveIASA(pars = NULL, init = NULL, time = NULL, s.range = NULL,
  a.range = NULL, alpha.range = NULL, v.range = NULL, s.fm = TRUE, ...)
```

Arguments

pars	a named vector of length 21, with point estimates of model parameters (see details).
init	a named vector of length 8, with point estimates of model parameters (see details).
time	time sequence for which output is wanted; the first value of times must be the initial time.
s.range	optional sequence (between 0 and 1) of the sterilization rates to be simulated.
a.range	optional vector of length 2, with range (ie, confidence interval) of abandonment rates to be assessed. If given, the rates evaluated are those specified by the argument plus the point estimate given in pars.
alpha.range	optional vector of length 2, with range (ie, confidence interval) of adoption rates to be assessed. If given, the rates evaluated are those specified by the argument plus the point estimate given in pars.
v.range	optional vector of length 2, with range of values of immigration rates to be assessed. This must be expressed as a percentage of owned animals carrying capacity.
s.fm	logical. If TRUE, s.range is used for females and males and if FALSE, it is only used for females (for males, the point estimate given in pars is used.)
...	further arguments passed to ode function.

Details

The pars argument must contain named values, using the following conventions: 1: owned animals; 2: stray animals; f: females; m: males. Then:

b1 and b2: number of births.

df1, dm1, df2 and dm2: death rate.

sf1, sm1, sf2 and sm2: sterilization rate.

k1 and k2: carrying capacity.

h1 and h2: mean harem size.

a: abandonment rate.

alpha: adoption rate.

v: immigration rate.

z: proportion of sterilized immigrants.

The init argument must contain named values for the inital number of animals, using the following conventions: 1: owned animals; 2: stray animals; f: females; m: males; and s: sterilized. Then, the names are:

f1, fs1, m1, ms1, f2, fs2, m2 and ms2.

If any range is specified (e.g s.range), the remaining ranges must be specified too (a.range, alpha.range and v.range). The function is a wrapper around the defaults of ode function, whose help page must be consulted for details. An exception is the method argument, which here has "rk4" as a default.

Value

list. The first element, name, is a string with the name of the function, the second element, model, is the model function. The third, fourth and fifth elements are vectors (pars, init, time, respectively) containing the pars, init and time arguments of the function. The sisxthth element results is a data.frame with up to as many rows as elements in time. The first column contain the time and subsequent columns contain the size of specific subpopulations, named according to conventions above. The group column differentiate between owned and strays. When *.range arguments are given, the last fourth columsn specify their instances.

Note

Logistic growth models are not intended for scenarios in which population size is greater than carrying capacity and growth rate is negative.

References

http://oswaldosantos.github.io/capm

See Also

ode.

Examples

```
# Parameters and initial conditions.
pars.solve.iasa = c(
   b1 = 21871, b2 = 4374,
   df1 = 0.104, dm1 = 0.098, df2 = 0.125, dm2 = 0.118,
   sf1 = 0.069, sf2 = 0.05, sm1 = 0.028, sm2 = 0.05,
   k1 = 98050, k2 = 8055, h1 = 1, h2 = 0.5,
   a = 0.054, alpha = 0.1, v = 0.2, z = 0.1)

init.solve.iasa = c(
   f1 = 33425, fs1 = 10865,
   m1 = 38039, ms1 = 6808,
   f2 = 3343, fs2 = 109,
   m2 = 3804, ms2 = 68)


# Solve for point estimates.
solve.iasa.pt <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:8, method = 'rk4')
```

```
# Solve for parameter ranges.
solve.iasa.rg <- SolveIASA(pars = pars.solve.iasa,
                           init = init.solve.iasa,
                           time = 0:8,
                           s.range = seq(0, .4, l = 15),
                           a.range = c(0, .2),
                           alpha.range = c(0, .2),
                           v.range = c(0, .1),
                           method = 'rk4')
```

SolveSI *Modelling of sterilization and immigration of comapnion animals.*

Description

System of ordinary differential equations to simulate the effect of sterilization and immigration on population dynamics.

Usage

```
SolveSI(pars = NULL, init = NULL, time = NULL, dd = "b", im = 0,
  s.range = NULL, ...)
```

Arguments

pars	vector of length 4. The values are point estimates of birth rate, death rate, carrying capacity and sterilization rate. The names of this values must be "b", "d", "k" and "s", respectively.
init	vector of length 2. The values are initial population size and initial proportion of sterilized animals. The names of this values must be "n" and "q", respectively.
time	time sequence for which output is wanted; the first value of times must be the initial time.
dd	string equal to b or d to define if density-dependece act on birth or death rartes respectively.
im	a number representing the total of immigrants per time unit.
s.range	optional sequence (between 0 and 1) of the sterilization rates to be simulated.
...	further arguments passed to ode function.

Details

The implemented model is described by Amaku, et. al., 2009 and the function is a wrapper around the defaults of ode function, whose help page must be consulted for details.

Value

`list`. The first element, `name`, is a string with the name of the function, the second element, `model`, is the model function. The third, fourth and fifth elements are vectors (`pars`, `init`, `time`, respectively) containing the `pars`, `init` and `time` arguments of the function. The sisxth element `results` is a `data.frame` with up to as many rows as elements in time. First column contains the time, second column the population size and third column the proportion of sterilized animals. If `s.range` is specified, fourth column contains its specific instances.

Note

Logistic growth models are not intended for scenarios in which population size is greater than carrying capacity and growth rate is negative.

References

Amaku M, Dias R and Ferreira F (2009). Dinamica populacional canina: potenciais efeitos de campanhas de esterilizacao. Revista Panamericana de Salud Publica, 25(4), pp. 300-304.

Soetaert K, Cash J and Mazzia F (2012). Solving differential equations in R. Springer.

http://oswaldosantos.github.io/capm

See Also

ode.

Examples

```
# Parameters and initial conditions from estimates
# obtained in examples section from svysumm function but
# estimating a proportion insted of a total for births.
pars.solve.si = c(b = 0.245, d = 0.101,
                  k = 98050, s = 0.048)
init.solve.si = c(n = 89137, q = 0.198)

# Solve for a specific sterilization rate.
solvesi.pt = SolveSI(pars = pars.solve.si,
                     init = init.solve.si,
                     time = 0:15, dd = 'b',
                     im = 100, method = 'rk4')

# Solve for a range of sterilization rates.
solvesi.rg = SolveSI(pars = pars.solve.si,
                     init = init.solve.si,
                     time = 0:15, dd = 'b', im = 100,
                     s.range = seq(0, .4, l = 50),
                     method = 'rk4')
```

SolveTC *Modelling of reversible contraception for companion animals*

Description

System of ordinary differential equations to simulate the effect of reversible contraception in a population at equilibrium, where deaths are compensated by births and net immigration.

Usage

```
SolveTC(pars = NULL, init = NULL, time = NULL, f.range = NULL,
  s.range = NULL, z.range = NULL, ...)
```

Arguments

pars	a named vector of length 5. The values are point estimates of the death rate (d), the fertility recovery rate (f), the sterilization rate (s), the proportion of infertile immigrants (z) and the proportion of the death rate compensated by immigration (r). Abreviations in parentheses indicate the names that must be given to the values.
init	a named vector of length 2, with the total number of fertil (n) and infertil (g) animals.
time	time sequence for which output is wanted; the first value of times must be the initial time.
f.range	optional sequence (between 0 and 1) with the fertility recovery rates to be simulated.
s.range	optional vector of length 2, with a range of sterilization rates to be assessed. If given, the rates evaluated are those specified by the argument plus the point estimate given in pars.
z.range	optional vector of length 2, with a range of the proportion of infertile immigrants. If given, the rates evaluated are those specified by the argument plus the point estimate given in pars.
...	further arguments passed to ode function.

Value

list. The first element, name, is a string with the name of the function, the second element, model, is the model function. The third, fourth and fifth elements are vectors (pars, init, time, respectively) containing the pars, init and time arguments of the function. The sisxthth element results is a data.frame with up to as many rows as elements in time. The first fourth columns contain the time and the variables: n, g and u. When *.range arguments are given, additional columns contain the variables f, s and z.

References

http://oswaldosantos.github.io/capm

See Also

ode.

Examples

```
# Parameters and initial conditions.
pars.solvetc <- c(d = 1 / 6, f = 0.5, s = 0.2,
                  z = 0.2, r = 0.8)

init.solvetc <- c(n = 950, g = 50)

# Solve for point estimates.
solvetc.pt <- SolveTC(pars = pars.solvetc,
                      init = init.solvetc,
                      time = 0:10, method = 'rk4')

# Solve for parameter ranges.
solvetc.rg <- SolveTC(pars = pars.solvetc,
                      init = init.solvetc,
                      time = 0:15,
                      f.range = seq(0, 1, 0.1),
                      s.range = c(0.05, 0.4),
                      z.range = c(0.05, 0.4),
                      method = 'rk4')
```

SummarySurvey *Summary statistics for sample surveys*

Description

Wraps functions for summary statistics from survey package.

Usage

```
SummarySurvey(design = NULL, variables = NULL, conf.level = 0.95,
  rnd = 3)
```

Arguments

design	an output form DesignSurvey function.
variables	character vector with the type of estimate for each variable contained in design (see details).
conf.level	the confidence level required.
rnd	the number of decimal places (round) or significant digits (signif) to be used. If NA, scientific notation is used.

Details

The length of `variables` must be equal to the length of `names(design$variables)` (see examples).

Value

Matrix with survey summaries.

References

Lumley, T. (2011). Complex surveys: A guide to analysis using R (Vol. 565). Wiley.

http://oswaldosantos.github.io/capm

Examples

```
# Load data.
data(psu.ssu)
data(survey.data)

########################################
## Example 1 (two-stage cluster design) ##
## General estimates                   ##
########################################

# Specify the two-stage cluster design.
design <- DesignSurvey(sample = survey.data, psu.ssu = psu.ssu,
                       psu.col = 2, ssu.col = 1, psu.2cd = 20)

# Look at the variables contained in the survey design
names(design$variables)

# Specify the type of estimate for each variable
variables <- c("total", "prop", "mean", rep("prop", 2),
               "total", rep("prop", 8))

# Make sure you specify the correct type of estimate for each variable
cbind(names(design$variables), variables)

# Calculate the summary statistics for the survey.
# Uncomment the following two lines (will take some seconds).
# estimates <- SummarySurvey(design, variables = variables, rnd = 3)

########################################
## Example 2 (two-stage cluster design) ##
## Sex-specific estimates              ##
########################################

# Make a copy of the dataset and select some
# variables of interest.
sample1 <- survey.data[, c(1:4, 6:7, 11)]
```

```
# Transform to numeric the "sterilized" variable in order
# to estimate its total.
sample1[, 5] <- as.character(sample1[, 5])
sample1[which(sample1$sterilized == "yes"), 5] <- 1
sample1[which(sample1[, 5] == "no"), 5] <- 0
sample1[, 5] <- as.numeric(sample1[, 5])

# Define a survey design for each sex.
design.sex <- DesignSurvey(sample = sample1, psu.ssu = psu.ssu,
                           psu.col = 2, ssu.col = 1, psu.2cd = 20)
design.f <- subset(design.sex, sex == 'Female')
design.m <- subset(design.sex, sex == 'Male')

# Look at the variables contained in the survey design
names(design.sex$variables)

# Specify the type of estimate for each variable
variables.sex <- c("total", "", "total", "prop", "prop")

# Make sure you specify the correct type of
# estimate for each variable
cbind(names(design.sex$variables), variables.sex)

# Calculate the summary statistics for the surveys.
# Uncomment the following two lines (will take some seconds).
# estimates.f <- SummarySurvey(design.f, variables.sex, rnd = 3)
# estimates.m <- SummarySurvey(design.m, variables.sex, rnd = 3)
```

`survey.data` *Estimates of dog demographic variables*

Description

This data set list dog demographic variables obtained from simulations based on field data.

Usage

```
data(survey.data)
```

Format

A data frame with 469 observations on the following 16 variables.

`interview_id` Interview Id.

`psu` Primary sampling unit Id.

`dogs` There is a dog in the secondary sampling unit (SSU). For each interviewed SSU with more than one dog, there is as many rows as dogs in it.

`sex` Sex of the dog.

`age` Age of the dog

`sterilized` The dog is sterilized (spayed/neutered).

`sterilized.ly` The dog was sterilized during the last year.

`births` Number of puppies during the last year.

`present` The dog was present in the SSU at the moment of the interview.

`fate` Fate of dogs: died, given, in_home (remains in home), lost or sold.

`acquired` Mode in which the dog was acquired: adopted, born_in_home, bought, gift (given by someone).

`outside` The dog was acquired in another city.

`acquired.ly` The dog was acquired during the last year

`immigrant` "acquired"" equals to "bought" or "outside"" equales to "yes". Immigration is understood as the movements from outside populations (populations from other cities and populations which maintain the pet market of the city.). Movements from stray populations are parameterized under category "adopted" of the variable "acquired".

`immigrant.ly` The dog immigrated during the last year.

`immigrant.sterilized.ly` The dog who immigrated during the last year was already sterilized.

Details

Some model parameters used in other capm function are estimates of this variables, obtained with `SummarySurvey` function. Variables refering to events occurred during the year previous to the interview are used as annual rates.

The data were simulated to approximate rates obtained in field studies to be published.

Examples

```
data(survey.data)
str(survey.data)
head(survey.data)
```

Index