Package: cellKey (via r-universe)

August 21, 2024

Type Package

Date 2023-11-24

Title Consistent Perturbation of Statistical Frequency- And Magnitude Tables

Version 1.0.2

Description Data from statistical agencies and other institutions often need to be protected before they can be published. This package can be used to perturb statistical tables in a consistent way. The main idea is to add - at the micro data level - a record key for each unit. Based on these keys, for any cell in a statistical table a cell key is computed as a function on the record keys contributing to a specific cell. Values that are added to the cell in order to perturb it are derived from a lookup-table that maps values of cell keys to specific perturbation values. The theoretical basis for the methods implemented can be found in Thompson, Broadfoot and Elazar (2013) <https://unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2013/Topic_1_ABS.pdf> which was extended and enhanced by Giessing and Tent (2019) <https://unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2019/mtg1/SDC2019_S2_Germany_Giessing_Tent_AD.pdf>.

Depends R(>= 4.1), sdcHierarchies (>= 0.19.3), data.table

Imports rlang, methods, digest (>= 0.6.23), sdcTable (>= 0.32.2), ptable (>= 1.0.0), cli, utils, yaml, parallel

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Bernhard Meindl [aut, cre]

Maintainer Bernhard Meindl <bernhard.meindl@statistik.gv.at>

LazyData true

LazyDataCompression xz

BugReports https://github.com/sdcTools/userSupport/issues

URL https://github.com/sdcTools/cellKey

Date/Publication 2023-11-24 11:10:02 UTC

Repository https://bernhard-da.r-universe.dev

RemoteUrl https://github.com/cran/cellKey

RemoteRef HEAD

RemoteSha 7246630aadc2132c1a2fa1168393efd978b41aad

Contents

`ck_class` *R6 Class defining statistical tables that can be perturbed*

Description

This class allows to define statistical tables and perturb both count and numerical variables.

Usage

```
ck_setup(x, rkey, dims, w = NULL, countvars = NULL, numvars = NULL)
```

Arguments

x	an object coercible to a data.frame
rkey	either a column name within x referring to a variable containing record keys or a single integer(ish) number > 5 that referns to the number of digits for record keys that will be generated internally.
dims	a list containing slots for each variable that should be tabulated. Each slot consists should be created/modified using sdcHierarchies::hier_create(), sdcHierarchies::hier_add() and other functionality from package sdcHierarchies.
w	(character) a scalar character referring to a variable in x holding sampling weights. If w is NULL (the default), all weights are assumed to be 1
countvars	(character) an optional vector containing names of binary (0/1 coded) variables withing x that should be included in the problem instance. These variables can later be perturbed.
numvars	(character) an optional vector of numerical variables that can later be tabulated.

Details

Such objects are typically generated using ck_setup().

Value

A new cellkey_obj object. Such objects (internally) contain the fully computed statistical tables given input microdata (x), the hierarchical definitionals (dims) as well as the remaining inputs. Intermediate results are stored internally and can only be modified / accessed via the exported public methods described below.

Methods

Public methods:

- ck_class$new()
- ck_class$perturb()
- ck_class$freqtab()
- ck_class$numtab()
- ck_class$measures_cnts()
- ck_class$measures_nums()
- ck_class$allvars()
- ck_class$cntvars()
- ck_class$numvars()
- ck_class$hierarchy_info()
- ck_class$mod_cnts()
- ck_class$mod_nums()
- ck_class$supp_freq()
- ck_class$supp_val()
- ck_class$supp_cells()

- ck_class$supp_p()
- ck_class$supp_pq()
- ck_class$supp_nk()
- ck_class$params_cnts_get()
- ck_class$params_cnts_set()
- ck_class$reset_cntvars()
- ck_class$reset_numvars()
- ck_class$reset_allvars()
- ck_class$params_nums_get()
- ck_class$params_nums_set()
- ck_class$summary()
- ck_class$print()

Method `new()`**:** Create a new table instance

Usage:

```
ck_class$new(x, rkey, dims, w = NULL, countvars = NULL, numvars = NULL)
```

Arguments:

`x` an object coercible to a `data.frame`

`rkey` either a column name within `x` referring to a variable containing record keys or a single integer(ish) number > 5 that referns to the number of digits for record keys that will be generated internally.

`dims` a list containing slots for each variable that should be tabulated. Each slot consists should be created/modified using `sdcHierarchies::hier_create()`, `sdcHierarchies::hier_add()` and other functionality from package `sdcHierarchies`.

`w` (character) a scalar character referring to a variable in `x` holding sampling weights. If `w` is `NULL` (the default), all weights are assumed to be 1

`countvars` (character) an optional vector containing names of binary (0/1 coded) variables withing `x` that should be included in the problem instance. These variables can later be perturbed.

`numvars` (character) an optional vector of numerical variables that can later be tabulated.

Returns: A new `cellkey_obj` object. Such objects (internally) contain the fully computed statistical tables given input microdata (`x`), the hierarchical definitionals (`dims`) as well as the remaining inputs. Intermediate results are stored internally and can only be modified / accessed via the exported public methods described below.

Method `perturb()`**:** Perturb a count- or magnitude variable

Usage:

```
ck_class$perturb(v)
```

Arguments:

`v` name(s) of count- or magnitude variables that should be perturbed.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. Updated data can be accessed using other exported methods like `$freqtab()` or `$numtab()`.

Method `freqtab()`**:** Extract results from already perturbed count variables as a `data.table`

Usage:

```
ck_class$freqtab(v = NULL, path = NULL)
```

Arguments:

`v` a vector of variable names for count variables. If `NULL` (the default), the results are returned for all available count variables. For variables that have not yet perturbed, columns `puwc` and `pwc` are filled with `NA`.

`path` if not `NULL`, a scalar character defining a (relative or absolute) path to which the result table should be written. A `csv` file will be generated and, if specified, `path` must have ".csv" as file-ending

Returns: This method returns a `data.table` containing all combinations of the dimensional variables in the first n columns. Additionally, the following columns are shown:

- `vname`: name of the perturbed variable
- `uwc`: unweighted counts
- `wc`: weighted counts
- `puwc`: perturbed unweighted counts or `NA` if `vname` was not yet perturbed
- `pwc`: perturbed weighted counts or `NA` if `vname` was not yet perturbed

Method `numtab()`**:** Extract results from already perturbed continuous variables as a `data.table`.

Usage:

```
ck_class$numtab(v = NULL, mean_before_sum = FALSE, path = NULL)
```

Arguments:

`v` a vector of variable names of continuous variables. If `NULL` (the default), the results are returned for all available numeric variables.

`mean_before_sum` (logical); if `TRUE`, the perturbed values are adjusted by a factor `((n+p))/n` with

- `n`: the original weighted cell value
- `p`: the perturbed cell value

This makes sense if the the accuracy of the variable mean is considered to be more important than accuracy of sums of the variable. The default value is `FALSE` (no adjustment is done)

`path` if not `NULL`, a scalar character defining a (relative or absolute) path to which the result table should be written. A `csv` file will be generated and, if specified, `path` must have ".csv" as file-ending

Returns: This method returns a `data.table` containing all combinations of the dimensional variables in the first n columns. Additionally, the following columns are shown:

- `vname`: name of the perturbed variable
- `uws`: unweighted sum of the given variable
- `ws`: weighted cellsum
- `pws`: perturbed weighted sum of the given cell or `NA` if `vname` has not not perturbed

Method `measures_cnts()`**:** Utility measures for perturbed count variables

Usage:

```
ck_class$measures_cnts(v, exclude_zeros = TRUE)
```

Arguments:

`v` name of a count variable for which utility measures should be computed.

`exclude_zeros` should empty (zero) cells in the original values be excluded when computing distance measures

Returns: This method returns a `list` containing a set of utility measures based on some distance functions. For a detailed description of the computed measures, see `ck_cnt_measures()`

Method `measures_nums()`**:** Utility measures for continuous variables (not yet implemented)

Usage:

```
ck_class$measures_nums(v)
```

Arguments:

`v` name of a continuous variable for which utility measures should be computed.

Returns: for (now) an empty list; In future versions of the package, the Method will return utility measures for perturbed magnitude tables.

Method `allvars()`**:** Names of variables that can be perturbed / tabulated

Usage:

```
ck_class$allvars()
```

Returns: returns a `list` with the following two elements:

- `cntvars`: character vector with names of available count variables for perturbation
- `numvars`: character vector with names of available numerical variables for perturbation

Method `cntvars()`**:** Names of count variables that can be perturbed

Usage:

```
ck_class$cntvars()
```

Returns: a character vector containing variable names

Method `numvars()`**:** Names of continuous variables that can be perturbed

Usage:

```
ck_class$numvars()
```

Returns: a character vector containing variable names

Method `hierarchy_info()`**:** Information about hierarchies

Usage:

```
ck_class$hierarchy_info()
```

Returns: a `list` (for each dimensional variable) with information on the hierarchies. This may be used to restrict output tables to specific levels or codes. Each list element is a `data.table` containing the following variables:

- `code`: the name of a code within the hierarchy
- `level`: number defining the level of the code; the higher the number, the lower the hierarchy with `1` being the overall total
- `is_leaf`: if `TRUE`, this code is a leaf node which means no other codes contribute to it
- `parent`: name of the parent code

Method `mod_cnts()`**:** Modifications applied to count variables

Usage:

```
ck_class$mod_cnts()
```

Returns: a `data.table` containing modifications applied to count variables

Method `mod_nums()`**:** Modifications applied to numerical variables

Usage:

```
ck_class$mod_nums()
```

Returns: a `data.table` containing modifications applied to numerical variables

Method `supp_freq()`**:** Identify sensitive cells based on minimum frequency rule

Usage:

```
ck_class$supp_freq(v, n, weighted = TRUE)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`n` a number defining the threshold. All cells `<= n` are considered as unsafe.

`weighted` if `TRUE`, the weighted number of contributors to a cell are compared to the threshold specified in `n` (default); else the unweighted number of contributors is used.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `supp_val()`**:** Identify sensitive cells based on weighted or unweighted cell value

Usage:

```
ck_class$supp_val(v, n, weighted = TRUE)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`n` a number defining the threshold. All cells `<= n` are considered as unsafe.

`weighted` if `TRUE`, the weighted cell value of variable `v` is compared to the threshold specified in `n` (default); else the unweighted number is used.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `supp_cells()`**:** Identify sensitive cells based on their `names`

Usage:

```
ck_class$supp_cells(v, inp)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`inp` a `data.frame` where each colum represents a dimensional variable. Each row of this input is then used to compute the relevant cells to be identified as sensitive where `NA`-values are possible and used to match any characteristics of the dimensional variable.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `supp_p()`**:** Identify sensitive cells based on the p%-rule rule. Please note that this rule can only be applied to positive-only variables.

Usage:

```
ck_class$supp_p(v, p)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`p` a number defining a percentage between `1` and `99`.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `supp_pq()`**:** Identify sensitive cells based on the pq-rule. Please note that this rule can only be applied to positive-only variables.

Usage:

```
ck_class$supp_pq(v, p, q)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`p` a number defining a percentage between `1` and `99`.

`q` a number defining a percentage between `1` and `99`. This value must be larger than `p`.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `supp_nk()`**:** Identify sensitive cells based on the nk-dominance rule. Please note that this rule can only be applied to positive-only variables.

Usage:

```
ck_class$supp_nk(v, n, k)
```

Arguments:

`v` a single variable name of a continuous variable (see method `numvars()`)

`n` an integerish number `>= 2`

`k` a number defining a percentage between `1` and `99`. All cells to which the top `n` contributers contribute more than `k%` is considered unsafe

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `params_cnts_get()`**:** Return perturbation parameters of count variables

Usage:

```
ck_class$params_cnts_get()
```

Returns: a named list in which each list-element contains the active perturbation parameters for the specific count variable defined by the list-name.

Method `params_cnts_set()`**:** Set perturbation parameters for count variables

Usage:

```
ck_class$params_cnts_set(val, v = NULL)
```

Arguments:

`val` a perturbation object created with `ck_params_cnts()`

`v` a character vector (or `NULL`). If `NULL` (the default), the perturbation parameters provided in `val` are set for all count variables; otherwise one may specify the names of the count variables for which the parameters should be set.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `reset_cntvars()`**:** reset results and parameters for already perturbed count variables

Usage:

```
ck_class$reset_cntvars(v = NULL)
```

Arguments:

`v` if `v` equals `NULL` (the default), the results are reset for all perturbed count variables; otherwise it is possible to specify the names of already perturbed count variables.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()` or `$freqtab()`).

Method `reset_numvars()`**:** reset results and parameters for already perturbed numerical variables

Usage:

```
ck_class$reset_numvars(v = NULL)
```

Arguments:

`v` if `v` equals `NULL` (the default), the results are reset for all perturbed numerical variables; otherwise it is possible to specify the names of already perturbed continuous variables.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()` or `$numtab()`).

Method `reset_allvars()`**:** reset results and parameters for all already perturbed variables.

Usage:

```
ck_class$reset_allvars()
```

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`, `$freqtab()` or `$numtab()`).

Method `params_nums_get()`**:** Return perturbation parameters of continuous variables

Usage:

```
ck_class$params_nums_get()
```

Returns: a named list in which each list-element contains the active perturbation parameters for the specific continuous variable defined by the list-name.

Method `params_nums_set()`**:** set perturbation parameters for continuous variables.

Usage:

```
ck_class$params_nums_set(val, v = NULL)
```

Arguments:

`val` a perturbation object created with `ck_params_nums()`

`v` a character vector (or `NULL`); if `NULL` (the default), the perturbation parameters provided in `val` are set for all continuous variables; otherwise one may specify the names of the numeric variables for which the parameters should be set.

Returns: A modified `cellkey_obj` object in which private slots were updated for side-effects. These updated values are used by other methods (e.g `$perturb()`).

Method `summary()`**:** some aggregated summary statistics about perturbed variables

Usage:

```
ck_class$summary()
```

Returns: invisible `NULL`

Method `print()`**:** prints information about the current table

Usage:

```
ck_class$print()
```

Returns: invisible `NULL`

Examples

```
x <- ck_create_testdata()

# create some 0/1 variables that should be perturbed later
x[, cnt_females := ifelse(sex == "male", 0, 1)]
x[, cnt_males := ifelse(sex == "male", 1, 0)]
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
# a variable with positive and negative contributions
x[, mixed := sample(-10:10, nrow(x), replace = TRUE)]

# create record keys
x$rkey <- ck_generate_rkeys(dat = x)

# define required inputs

# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_sex <- hier_add(d_sex, root = "female", "f")
d_sex <- hier_add(d_sex, root = "male", "m")

d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
d_age <- hier_add(d_age, root = "age_group1", "ag1a")
d_age <- hier_add(d_age, root = "age_group2", "ag2a")

# define the cell key object
countvars <- c("cnt_females", "cnt_males", "cnt_highincome")
numvars <- c("expend", "income", "savings", "mixed")
tab <- ck_setup(
  x = x,
  rkey = "rkey",
  dims = list(sex = d_sex, age = d_age),
  w = "sampling_weight",
```

```
  countvars = countvars,
  numvars = numvars)

# show some information about this table instance
tab$print() # identical with print(tab)

# information about the hierarchies
tab$hierarchy_info()

# which variables have been defined?
tab$allvars()

# count variables
tab$cntvars()

# continuous variables
tab$numvars()

# create perturbation parameters for "total" variable and
# write to yaml-file

# create a ptable using functionality from the ptable-pkg
f_yaml <- tempfile(fileext = ".yaml")
p_cnts1 <- ck_params_cnts(
  ptab = ptable::pt_ex_cnts(),
  path = f_yaml)

# read parameters from yaml-file and set them for variable `"total"`
p_cnts1 <- ck_read_yaml(path = f_yaml)

tab$params_cnts_set(val = p_cnts1, v = "total")

# create alternative perturbation parameters by specifying parameters
para2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, create = FALSE)

p_cnts2 <- ck_params_cnts(ptab = para2)

# use these ptable it for the remaining variables
tab$params_cnts_set(val = p_cnts2, v = countvars)

# perturb a variable
tab$perturb(v = "total")

# multiple variables can be perturbed as well
tab$perturb(v = c("cnt_males", "cnt_highincome"))

# return weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# numerical variables (positive variables using flex-function)
# we also write the config to a yaml file
f_yaml <- tempfile(fileext = ".yaml")
```

```
# create a ptable using functionality from the ptable-pkg
# a single ptable for all cells
ptab1 <- ptable::pt_ex_nums(parity = TRUE, separation = FALSE)

# a single ptab for all cells except for very small ones
ptab2 <- ptable::pt_ex_nums(parity = TRUE, separation = TRUE)

# different ptables for cells with even/odd number of contributors
# and very small cells
ptab3 <- ptable::pt_ex_nums(parity = FALSE, separation = TRUE)

p_nums1 <- ck_params_nums(
  ptab = ptab1,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.30, 0.03),
    epsilon = c(1, 0.5, 0.2),
    q = 3),
  mu_c = 2,
  same_key = FALSE,
  use_zero_rkeys = FALSE,
  path = f_yaml)

# we read the parameters from the yaml-file
p_nums1 <- ck_read_yaml(path = f_yaml)

# for variables with positive and negative values
p_nums2 <- ck_params_nums(
  ptab = ptab2,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.15, 0.02),
    epsilon = c(1, 0.4, 0.15),
    q = 3),
  mu_c = 2,
  same_key = FALSE)

# simple perturbation parameters (not using the flex-function approach)
p_nums3 <- ck_params_nums(
  ptab = ptab3,
  type = "mean",
  mult_params = ck_simpleparams(p = 0.25),
  mu_c = 2,
  same_key = FALSE)

# use `p_nums1` for all variables
tab$params_nums_set(p_nums1, c("savings", "income", "expend"))
```

```
# use different parameters for variable `mixed`
tab$params_nums_set(p_nums2, v = "mixed")

# identify sensitive cells to which extra protection (`mu_c`) is added.
tab$supp_p(v = "income", p = 85)
tab$supp_pq(v = "income", p = 85, q = 90)
tab$supp_nk(v = "income", n = 2, k = 90)
tab$supp_freq(v = "income", n = 14, weighted = FALSE)
tab$supp_val(v = "income", n = 10000, weighted = TRUE)
tab$supp_cells(
  v = "income",
  inp = data.frame(
    sex = c("female", "female"),
    "age" = c("age_group1", "age_group3")
  )
)

# perturb variables
tab$perturb(v = c("income", "savings"))

# extract results
tab$numtab("income", mean_before_sum = TRUE)
tab$numtab("income", mean_before_sum = FALSE)
tab$numtab("savings")

# results can be resetted, too
tab$reset_cntvars(v = "cnt_males")

# we can then set other parameters and perturb again
tab$params_cnts_set(val = p_cnts1, v = "cnt_males")

tab$perturb(v = "cnt_males")

# write results to a .csv file
tab$freqtab(
  v = c("total", "cnt_males"),
  path = file.path(tempdir(), "outtab.csv")
)

# show results containing weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# utility measures for a count variable
tab$measures_cnts(v = "total", exclude_zeros = TRUE)

# modifications for perturbed count variables
tab$mod_cnts()

# display a summary about utility measures
tab$summary()
```

ck_cnt_measures *Utility measures for perturbed counts*

Description

This function computes utility/information loss measures based on two numeric vectors (original and perturbed)

Usage

```
ck_cnt_measures(orig, pert, exclude_zeros = TRUE)
```

Arguments

orig	a numeric vector holding original values
pert	a numeric vector holding perturbed values
exclude_zeros	a scalar logical value; if TRUE (the default), all only cells with counts > 0 are used when computing distances d1, d2 and d3. If this argument is FALSE, the complete vector is used.

Value

a list containing the following elements:

- overview: a data.table with the following three columns:
  - noise: amount of noise computed as orig - pert
  - cnt: number of cells perturbed with the value given in column noise
  - pct: percentage of cells perturbed with the value given in column noise
- measures: a data.table containing measures of the distribution of three different distances between original and perturbed values of the unweighted counts. Column what specifies the computed measure. The three distances considered are:
  - d1: absolute distance between original and masked values
  - d2: relative absolute distance between original and masked values
  - d3: absolute distance between square-roots of original and perturbed values
- cumdistr_d1, cumdistr_d2 and cumdistr_d3: for each distance d1, d2 and d3, a data.table with the following three columns:
  - cat: a specific value (for d1) or interval (for distances d2 and d3)
  - cnt: number of records smaller or equal the value in column cat for the given distance
  - pct: proportion of records smaller or equal the value in column cat for the selected distance
- false_zero: number of cells that were perturbed to zero
- false_nonzero: number of cells that were initially zero but have been perturbed to a number different from zero
- exclude_zeros: were empty cells exluded from computation or not

Examples

```
orig <- c(1:10, 0, 0)
pert <- orig; pert[c(1, 5, 7)] <- c(0, 6, 9)

# ignore empty cells when computing measures `d1`, `d2`, `d3`
ck_cnt_measures(orig = orig, pert = pert, exclude_zeros = TRUE)

# use all cells
ck_cnt_measures(orig = orig, pert = pert, exclude_zeros = FALSE)

# for an application on a perturbed object, see ?cellkey_pkg
```

---

ck_create_testdata *ck_create_testdata*

Description

this function generates some test-data

Usage

```
ck_create_testdata()
```

Value

a `data.frame`

Examples

```
dat <- ck_create_testdata(); print(str(dat))
```

---

ck_dat_hc92 *A real-world data set on persons*

Description

820000 obervations in 5 Variables without sampling weights.

Format

ck_dat_hc92: a data frame with 820000 observations on the following 6 variables.

- `id`: a numeric identifier
- `geo_m`: a character vector defining regions
- `sex a`: character vector defining gender
- `age_m`: a character vector containing age groups
- `yae_h`: a character vector
- `rkey`: a numeric vector holding record keys

References

https://ec.europa.eu/eurostat/cros/content/3-random-noise-cell-key-method_en

Examples

```
data(ck_dat_hc92)
head(ck_dat_hc92)
```

`ck_flexparams` *Set parameters required to perturb numeric variables using a flex function*

Description

`ck_flexparams()` allows to define a flex function that is used to lookup perturbation magnitudes (percentages) used when perturbing continuous variables.

Usage

```
ck_flexparams(fp, p = c(0.25, 0.05), epsilon = 1, q = 3)
```

Arguments

`fp`	(numeric scalar); at which point should the noise coefficient function reaches its desired maximum (defined by the first element of `p`)
`p`	a numeric vector of length `2` where both elements specify a percentage. The first value refers to the desired maximum perturbation percentage for small cells (depending on `fp`) while the second element refers to the desired maximum perturbation percentage for large cells. Both values must be between `0` and `1` and need to be in descending order.
`epsilon`	a numeric vector in descending order with all values `>= 0` and `<= 1` with the first element forced to equal 1. The length of this vector must correspond with the number `top_k` specified in `ck_params_nums()` when creating parameters for `type == "top_contr"` which is checked at runtime. This setting allows to use different flex-functions for the largest `top_k` contributors.
`q`	(numeric scalar); Parameter of the function; `q` needs to be `>= 1`

Details

details about the flex function can be found in Deliverable D4.2, Part I in SGA *"Open Source tools for perturbative confidentiality methods"*

Value

an object suitable as input for `ck_params_nums()`.

See Also

ck_simpleparams(), ck_params_nums()

Examples

```
x <- ck_create_testdata()

# create some 0/1 variables that should be perturbed later
x[, cnt_females := ifelse(sex == "male", 0, 1)]
x[, cnt_males := ifelse(sex == "male", 1, 0)]
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
# a variable with positive and negative contributions
x[, mixed := sample(-10:10, nrow(x), replace = TRUE)]

# create record keys
x$rkey <- ck_generate_rkeys(dat = x)

# define required inputs

# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_sex <- hier_add(d_sex, root = "female", "f")
d_sex <- hier_add(d_sex, root = "male", "m")

d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
d_age <- hier_add(d_age, root = "age_group1", "ag1a")
d_age <- hier_add(d_age, root = "age_group2", "ag2a")

# define the cell key object
countvars <- c("cnt_females", "cnt_males", "cnt_highincome")
numvars <- c("expend", "income", "savings", "mixed")
tab <- ck_setup(
  x = x,
  rkey = "rkey",
  dims = list(sex = d_sex, age = d_age),
  w = "sampling_weight",
  countvars = countvars,
  numvars = numvars)

# show some information about this table instance
tab$print() # identical with print(tab)

# information about the hierarchies
tab$hierarchy_info()

# which variables have been defined?
tab$allvars()

# count variables
tab$cntvars()

# continuous variables
```

```
tab$numvars()

# create perturbation parameters for "total" variable and
# write to yaml-file

# create a ptable using functionality from the ptable-pkg
f_yaml <- tempfile(fileext = ".yaml")
p_cnts1 <- ck_params_cnts(
  ptab = ptable::pt_ex_cnts(),
  path = f_yaml)

# read parameters from yaml-file and set them for variable `"total"`
p_cnts1 <- ck_read_yaml(path = f_yaml)

tab$params_cnts_set(val = p_cnts1, v = "total")

# create alternative perturbation parameters by specifying parameters
para2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, create = FALSE)

p_cnts2 <- ck_params_cnts(ptab = para2)

# use these ptable it for the remaining variables
tab$params_cnts_set(val = p_cnts2, v = countvars)

# perturb a variable
tab$perturb(v = "total")

# multiple variables can be perturbed as well
tab$perturb(v = c("cnt_males", "cnt_highincome"))

# return weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# numerical variables (positive variables using flex-function)
# we also write the config to a yaml file
f_yaml <- tempfile(fileext = ".yaml")

# create a ptable using functionality from the ptable-pkg
# a single ptable for all cells
ptab1 <- ptable::pt_ex_nums(parity = TRUE, separation = FALSE)

# a single ptab for all cells except for very small ones
ptab2 <- ptable::pt_ex_nums(parity = TRUE, separation = TRUE)

# different ptables for cells with even/odd number of contributors
# and very small cells
ptab3 <- ptable::pt_ex_nums(parity = FALSE, separation = TRUE)

p_nums1 <- ck_params_nums(
  ptab = ptab1,
  type = "top_contr",
  top_k = 3,
```

```
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.30, 0.03),
    epsilon = c(1, 0.5, 0.2),
    q = 3),
  mu_c = 2,
  same_key = FALSE,
  use_zero_rkeys = FALSE,
  path = f_yaml)

# we read the parameters from the yaml-file
p_nums1 <- ck_read_yaml(path = f_yaml)

# for variables with positive and negative values
p_nums2 <- ck_params_nums(
  ptab = ptab2,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.15, 0.02),
    epsilon = c(1, 0.4, 0.15),
    q = 3),
  mu_c = 2,
  same_key = FALSE)

# simple perturbation parameters (not using the flex-function approach)
p_nums3 <- ck_params_nums(
  ptab = ptab3,
  type = "mean",
  mult_params = ck_simpleparams(p = 0.25),
  mu_c = 2,
  same_key = FALSE)

# use `p_nums1` for all variables
tab$params_nums_set(p_nums1, c("savings", "income", "expend"))

# use different parameters for variable `mixed`
tab$params_nums_set(p_nums2, v = "mixed")

# identify sensitive cells to which extra protection (`mu_c`) is added.
tab$supp_p(v = "income", p = 85)
tab$supp_pq(v = "income", p = 85, q = 90)
tab$supp_nk(v = "income", n = 2, k = 90)
tab$supp_freq(v = "income", n = 14, weighted = FALSE)
tab$supp_val(v = "income", n = 10000, weighted = TRUE)
tab$supp_cells(
  v = "income",
  inp = data.frame(
    sex = c("female", "female"),
    "age" = c("age_group1", "age_group3")
  )
)
```

```
# perturb variables
tab$perturb(v = c("income", "savings"))

# extract results
tab$numtab("income", mean_before_sum = TRUE)
tab$numtab("income", mean_before_sum = FALSE)
tab$numtab("savings")

# results can be resetted, too
tab$reset_cntvars(v = "cnt_males")

# we can then set other parameters and perturb again
tab$params_cnts_set(val = p_cnts1, v = "cnt_males")

tab$perturb(v = "cnt_males")

# write results to a .csv file
tab$freqtab(
  v = c("total", "cnt_males"),
  path = file.path(tempdir(), "outtab.csv")
)

# show results containing weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# utility measures for a count variable
tab$measures_cnts(v = "total", exclude_zeros = TRUE)

# modifications for perturbed count variables
tab$mod_cnts()

# display a summary about utility measures
tab$summary()
```

`ck_generate_rkeys` *Generate random record keys*

Description

This function allows to create random record keys from a uniform distribution. If no seed is specified, a seed value is computed from the input data set to allow for reproducability depending on the input data set.

Usage

```
ck_generate_rkeys(dat, nr_digits = 8, seed = NULL)
```

Arguments

dat	microdata used to generated hash for random seed
nr_digits	maximum number of digits in the record keys. The default setting (8) corresponds with the default setting of the method in tau-argus.
seed	if not NULL, a number specifying the initial seed value for the random number generator. If NULL, a seed is computed from dat itself.

Value

a numeric vector with nrow(dat) record keys

Examples

```
dat <- ck_create_testdata()
dat$rkeys <- ck_generate_rkeys(dat = ck_create_testdata(), nr_digits = 8)
```

---

ck_params_cnts *Create perturbation parameters for count variables*

Description

This function allows to generate required perturbation parameters that are used to perturb count variables.

Usage

```
ck_params_cnts(ptab, path = NULL)
```

Arguments

ptab	an object created with ptable::create_ptable(), or ptable::create_cnt_ptable()
path	a scalar character specifying a path to which the parameters created with this functions should be written to (in yaml format)

Value

an object suitable as input to method $params_cnts_set() for the perturbation of counts and frequencies.

See Also

This function uses functionality from package ptable (https://github.com/sdcTools/ptable), expecially ptable::create_ptable() and ptable::create_cnt_ptable(). More detailed information on the parameters is available from the respective help-pages of these functions.

Examples

```
x <- ck_create_testdata()

# create some 0/1 variables that should be perturbed later
x[, cnt_females := ifelse(sex == "male", 0, 1)]
x[, cnt_males := ifelse(sex == "male", 1, 0)]
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
# a variable with positive and negative contributions
x[, mixed := sample(-10:10, nrow(x), replace = TRUE)]

# create record keys
x$rkey <- ck_generate_rkeys(dat = x)

# define required inputs

# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_sex <- hier_add(d_sex, root = "female", "f")
d_sex <- hier_add(d_sex, root = "male", "m")

d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
d_age <- hier_add(d_age, root = "age_group1", "ag1a")
d_age <- hier_add(d_age, root = "age_group2", "ag2a")

# define the cell key object
countvars <- c("cnt_females", "cnt_males", "cnt_highincome")
numvars <- c("expend", "income", "savings", "mixed")
tab <- ck_setup(
  x = x,
  rkey = "rkey",
  dims = list(sex = d_sex, age = d_age),
  w = "sampling_weight",
  countvars = countvars,
  numvars = numvars)

# show some information about this table instance
tab$print() # identical with print(tab)

# information about the hierarchies
tab$hierarchy_info()

# which variables have been defined?
tab$allvars()

# count variables
tab$cntvars()

# continuous variables
tab$numvars()

# create perturbation parameters for "total" variable and
# write to yaml-file
```

```
# create a ptable using functionality from the ptable-pkg
f_yaml <- tempfile(fileext = ".yaml")
p_cnts1 <- ck_params_cnts(
  ptab = ptable::pt_ex_cnts(),
  path = f_yaml)

# read parameters from yaml-file and set them for variable `"total"`
p_cnts1 <- ck_read_yaml(path = f_yaml)

tab$params_cnts_set(val = p_cnts1, v = "total")

# create alternative perturbation parameters by specifying parameters
para2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, create = FALSE)

p_cnts2 <- ck_params_cnts(ptab = para2)

# use these ptable it for the remaining variables
tab$params_cnts_set(val = p_cnts2, v = countvars)

# perturb a variable
tab$perturb(v = "total")

# multiple variables can be perturbed as well
tab$perturb(v = c("cnt_males", "cnt_highincome"))

# return weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# numerical variables (positive variables using flex-function)
# we also write the config to a yaml file
f_yaml <- tempfile(fileext = ".yaml")

# create a ptable using functionality from the ptable-pkg
# a single ptable for all cells
ptab1 <- ptable::pt_ex_nums(parity = TRUE, separation = FALSE)

# a single ptab for all cells except for very small ones
ptab2 <- ptable::pt_ex_nums(parity = TRUE, separation = TRUE)

# different ptables for cells with even/odd number of contributors
# and very small cells
ptab3 <- ptable::pt_ex_nums(parity = FALSE, separation = TRUE)

p_nums1 <- ck_params_nums(
  ptab = ptab1,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.30, 0.03),
    epsilon = c(1, 0.5, 0.2),
```

```
    q = 3),
  mu_c = 2,
  same_key = FALSE,
  use_zero_rkeys = FALSE,
  path = f_yaml)

# we read the parameters from the yaml-file
p_nums1 <- ck_read_yaml(path = f_yaml)

# for variables with positive and negative values
p_nums2 <- ck_params_nums(
  ptab = ptab2,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.15, 0.02),
    epsilon = c(1, 0.4, 0.15),
    q = 3),
  mu_c = 2,
  same_key = FALSE)

# simple perturbation parameters (not using the flex-function approach)
p_nums3 <- ck_params_nums(
  ptab = ptab3,
  type = "mean",
  mult_params = ck_simpleparams(p = 0.25),
  mu_c = 2,
  same_key = FALSE)

# use `p_nums1` for all variables
tab$params_nums_set(p_nums1, c("savings", "income", "expend"))

# use different parameters for variable `mixed`
tab$params_nums_set(p_nums2, v = "mixed")

# identify sensitive cells to which extra protection (`mu_c`) is added.
tab$supp_p(v = "income", p = 85)
tab$supp_pq(v = "income", p = 85, q = 90)
tab$supp_nk(v = "income", n = 2, k = 90)
tab$supp_freq(v = "income", n = 14, weighted = FALSE)
tab$supp_val(v = "income", n = 10000, weighted = TRUE)
tab$supp_cells(
  v = "income",
  inp = data.frame(
    sex = c("female", "female"),
    "age" = c("age_group1", "age_group3")
  )
)

# perturb variables
tab$perturb(v = c("income", "savings"))
```

```
# extract results
tab$numtab("income", mean_before_sum = TRUE)
tab$numtab("income", mean_before_sum = FALSE)
tab$numtab("savings")

# results can be resetted, too
tab$reset_cntvars(v = "cnt_males")

# we can then set other parameters and perturb again
tab$params_cnts_set(val = p_cnts1, v = "cnt_males")

tab$perturb(v = "cnt_males")

# write results to a .csv file
tab$freqtab(
  v = c("total", "cnt_males"),
  path = file.path(tempdir(), "outtab.csv")
)

# show results containing weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# utility measures for a count variable
tab$measures_cnts(v = "total", exclude_zeros = TRUE)

# modifications for perturbed count variables
tab$mod_cnts()

# display a summary about utility measures
tab$summary()
```

---

ck_params_nums *Set perturbation parameters for continuous variables*

Description

This function allows to define perturbation parameters used to perturb cells in magnitude tables.

Usage

```
ck_params_nums(
  type = "top_contr",
  top_k = NULL,
  ptab,
  mult_params,
  mu_c = 0,
  same_key = TRUE,
  use_zero_rkeys = FALSE,
  path = NULL
)
```

Arguments

type	a character value defining the way to identify the magnifier, e.g which contributions/values in a cell should be the used in the perturbation procedure. Possible choices are: • top_contr: the k largest contributions are used. In this case, it is also required to specify argument top_k • mean: the (weighted) cellmean is used as starting point • range: the difference between largest and smallest contribution is used. • sum: the (weighted) cellvalue itself is used as starting point
top_k	it is ignored if variant is different from top_contr. Otherwise, a scalar number >0 is expected.
ptab	in this argument, one ore more perturbation tables are given as input. the following choices are possible: • an object derived from ptable::create_ptable() or ptable::create_num_ptable(): this case is the same as specifying a named list with only a single element "all" (as described below) • a named list where the allowed names are shown below and each element must be the output of [ptable::create_ptable())] or ptable::create_num_ptable() – "all": this ptable will be used for all cells; if specified, no elements named "even" or "odd" must exist. – "even": will be used to look up perturbation values for cells with an even number of contributors. if specified, also list-element "odd" must exist. – "odd": will be used to look up perturbation values for cells with an odd number of contributors; if specified, also list-element "even" must exist. – "small_cells": if specified, this ptable will be used to extract perturbation values for very small cells [ptable::create_ptable())]: R:ptable::create_ptable()) ptable::create_num_ptable(): R:ptable::create_num_ptable()
mult_params	an object derived with ck_flexparams() or ck_simpleparams() that contain required parameters for the computation of the perturbation multiplier
mu_c	fixed extra protection amount (>= 0) applied to the absolute of the perturbation value of the first (largest) noise component if the cell is sensitive. This value defaults to 0 (no additional protection). Please note that sensitive cells can be defined according using the supp_freq(), supp_val, supp_p(), supp_nk() and supp_pq() methods. An examples is given in ?cellkey_pkg.
same_key	(logical) should original cell key (TRUE) used for for finding perturbation values of the largest contributor to a cell or should a perturbation of the cellkey itself (FALSE) take place.
use_zero_rkeys	(logical) scalar defining if record keys of units not contributing to a specific numeric variables should be used (TRUE) or ignored (FALSE) when computing cell keys.
path	a scalar character specifying a path to which the parameters created with this functions should be written to (in yaml format)

Value

an object suitable as input to method `$params_nums_set()` for the perturbation of continous variables.

See Also

`ck_flexparams()`

Examples

```
# create a perturbation table using
# functionality from ptable-pkg; see help(pa = "ptable")
# this returns an extra ptable for very small cells
ptab <- ptable::pt_ex_nums(separation = TRUE)

# create parameters
ck_params_nums(
  type = "top_contr",
  top_k = 3,
  ptab = ptab,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.20, 0.03),
    epsilon = c(1, 0.5, 0.2),
    q = 2),
  use_zero_rkeys = TRUE,
  mu_c = 3)
```

`ck_read_yaml` *Read perturbation parameters from yaml-files*

Description

`ck_read_yaml()` allows to create perturbation parameter inputs from yaml-files that were previously created using `ck_params_cnts()` or `ck_params_nums()`.

Usage

```
ck_read_yaml(path)
```

Arguments

`path`	a path to a yaml-input file

Value

an object object suitable as input to method `$params_nums_set()` for the perturbation of continous variables in case `path` was created using `ck_params_nums()` or an object suitable as input for `$params_cnts_set()` for the perturbation of counts and frequencies if the input file was generated using `ck_params_cnts()`.

Examples

```
x <- ck_create_testdata()

# create some 0/1 variables that should be perturbed later
x[, cnt_females := ifelse(sex == "male", 0, 1)]
x[, cnt_males := ifelse(sex == "male", 1, 0)]
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
# a variable with positive and negative contributions
x[, mixed := sample(-10:10, nrow(x), replace = TRUE)]

# create record keys
x$rkey <- ck_generate_rkeys(dat = x)

# define required inputs

# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_sex <- hier_add(d_sex, root = "female", "f")
d_sex <- hier_add(d_sex, root = "male", "m")

d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
d_age <- hier_add(d_age, root = "age_group1", "ag1a")
d_age <- hier_add(d_age, root = "age_group2", "ag2a")

# define the cell key object
countvars <- c("cnt_females", "cnt_males", "cnt_highincome")
numvars <- c("expend", "income", "savings", "mixed")
tab <- ck_setup(
  x = x,
  rkey = "rkey",
  dims = list(sex = d_sex, age = d_age),
  w = "sampling_weight",
  countvars = countvars,
  numvars = numvars)

# show some information about this table instance
tab$print() # identical with print(tab)

# information about the hierarchies
tab$hierarchy_info()

# which variables have been defined?
tab$allvars()

# count variables
tab$cntvars()

# continuous variables
tab$numvars()

# create perturbation parameters for "total" variable and
# write to yaml-file
```

```
# create a ptable using functionality from the ptable-pkg
f_yaml <- tempfile(fileext = ".yaml")
p_cnts1 <- ck_params_cnts(
  ptab = ptable::pt_ex_cnts(),
  path = f_yaml)

# read parameters from yaml-file and set them for variable `"total"`
p_cnts1 <- ck_read_yaml(path = f_yaml)

tab$params_cnts_set(val = p_cnts1, v = "total")

# create alternative perturbation parameters by specifying parameters
para2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, create = FALSE)

p_cnts2 <- ck_params_cnts(ptab = para2)

# use these ptable it for the remaining variables
tab$params_cnts_set(val = p_cnts2, v = countvars)

# perturb a variable
tab$perturb(v = "total")

# multiple variables can be perturbed as well
tab$perturb(v = c("cnt_males", "cnt_highincome"))

# return weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# numerical variables (positive variables using flex-function)
# we also write the config to a yaml file
f_yaml <- tempfile(fileext = ".yaml")

# create a ptable using functionality from the ptable-pkg
# a single ptable for all cells
ptab1 <- ptable::pt_ex_nums(parity = TRUE, separation = FALSE)

# a single ptab for all cells except for very small ones
ptab2 <- ptable::pt_ex_nums(parity = TRUE, separation = TRUE)

# different ptables for cells with even/odd number of contributors
# and very small cells
ptab3 <- ptable::pt_ex_nums(parity = FALSE, separation = TRUE)

p_nums1 <- ck_params_nums(
  ptab = ptab1,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.30, 0.03),
    epsilon = c(1, 0.5, 0.2),
```

```
    q = 3),
  mu_c = 2,
  same_key = FALSE,
  use_zero_rkeys = FALSE,
  path = f_yaml)

# we read the parameters from the yaml-file
p_nums1 <- ck_read_yaml(path = f_yaml)

# for variables with positive and negative values
p_nums2 <- ck_params_nums(
  ptab = ptab2,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.15, 0.02),
    epsilon = c(1, 0.4, 0.15),
    q = 3),
  mu_c = 2,
  same_key = FALSE)

# simple perturbation parameters (not using the flex-function approach)
p_nums3 <- ck_params_nums(
  ptab = ptab3,
  type = "mean",
  mult_params = ck_simpleparams(p = 0.25),
  mu_c = 2,
  same_key = FALSE)

# use `p_nums1` for all variables
tab$params_nums_set(p_nums1, c("savings", "income", "expend"))

# use different parameters for variable `mixed`
tab$params_nums_set(p_nums2, v = "mixed")

# identify sensitive cells to which extra protection (`mu_c`) is added.
tab$supp_p(v = "income", p = 85)
tab$supp_pq(v = "income", p = 85, q = 90)
tab$supp_nk(v = "income", n = 2, k = 90)
tab$supp_freq(v = "income", n = 14, weighted = FALSE)
tab$supp_val(v = "income", n = 10000, weighted = TRUE)
tab$supp_cells(
  v = "income",
  inp = data.frame(
    sex = c("female", "female"),
    "age" = c("age_group1", "age_group3")
  )
)

# perturb variables
tab$perturb(v = c("income", "savings"))
```

```
# extract results
tab$numtab("income", mean_before_sum = TRUE)
tab$numtab("income", mean_before_sum = FALSE)
tab$numtab("savings")

# results can be resetted, too
tab$reset_cntvars(v = "cnt_males")

# we can then set other parameters and perturb again
tab$params_cnts_set(val = p_cnts1, v = "cnt_males")

tab$perturb(v = "cnt_males")

# write results to a .csv file
tab$freqtab(
  v = c("total", "cnt_males"),
  path = file.path(tempdir(), "outtab.csv")
)

# show results containing weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# utility measures for a count variable
tab$measures_cnts(v = "total", exclude_zeros = TRUE)

# modifications for perturbed count variables
tab$mod_cnts()

# display a summary about utility measures
tab$summary()
```

`ck_simpleparams` *Set parameters required to perturb numeric variables using a simple approach*

Description

`ck_simpleparams()` allows to define parameters for a simple perturbation approach based on a single magnitude parameter (`m`). The values of `epsilon` are used to `"weight"` parameter `m` in case `type == "top_contr"` is set in `ck_params_nums()`.

Usage

```
ck_simpleparams(p, epsilon = 1)
```

Arguments

`p`	a percentage value used as magnitude for perturbation

epsilon — a numeric vector in descending order with all values >= 0 and <= 1 with the first element forced to equal 1. The length of this vector must correspond with the number top_k specified in ck_params_nums() when creating parameters for type == "top_contr" which is checked at runtime. This setting allows to use different flex-functions for the largest top_k contributors.

Details

details about the flex function can be found in Deliverable D4.2, Part I in SGA *"Open Source tools for perturbative confidentiality methods"*

Value

an object suitable as input for ck_params_nums().

See Also

ck_flexparams(), ck_params_nums()

Examples

```
x <- ck_create_testdata()

# create some 0/1 variables that should be perturbed later
x[, cnt_females := ifelse(sex == "male", 0, 1)]
x[, cnt_males := ifelse(sex == "male", 1, 0)]
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
# a variable with positive and negative contributions
x[, mixed := sample(-10:10, nrow(x), replace = TRUE)]

# create record keys
x$rkey <- ck_generate_rkeys(dat = x)

# define required inputs

# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_sex <- hier_add(d_sex, root = "female", "f")
d_sex <- hier_add(d_sex, root = "male", "m")

d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
d_age <- hier_add(d_age, root = "age_group1", "ag1a")
d_age <- hier_add(d_age, root = "age_group2", "ag2a")

# define the cell key object
countvars <- c("cnt_females", "cnt_males", "cnt_highincome")
numvars <- c("expend", "income", "savings", "mixed")
tab <- ck_setup(
  x = x,
  rkey = "rkey",
  dims = list(sex = d_sex, age = d_age),
  w = "sampling_weight",
```

```
  countvars = countvars,
  numvars = numvars)

# show some information about this table instance
tab$print() # identical with print(tab)

# information about the hierarchies
tab$hierarchy_info()

# which variables have been defined?
tab$allvars()

# count variables
tab$cntvars()

# continuous variables
tab$numvars()

# create perturbation parameters for "total" variable and
# write to yaml-file

# create a ptable using functionality from the ptable-pkg
f_yaml <- tempfile(fileext = ".yaml")
p_cnts1 <- ck_params_cnts(
  ptab = ptable::pt_ex_cnts(),
  path = f_yaml)

# read parameters from yaml-file and set them for variable `"total"`
p_cnts1 <- ck_read_yaml(path = f_yaml)

tab$params_cnts_set(val = p_cnts1, v = "total")

# create alternative perturbation parameters by specifying parameters
para2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, create = FALSE)

p_cnts2 <- ck_params_cnts(ptab = para2)

# use these ptable it for the remaining variables
tab$params_cnts_set(val = p_cnts2, v = countvars)

# perturb a variable
tab$perturb(v = "total")

# multiple variables can be perturbed as well
tab$perturb(v = c("cnt_males", "cnt_highincome"))

# return weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# numerical variables (positive variables using flex-function)
# we also write the config to a yaml file
f_yaml <- tempfile(fileext = ".yaml")
```

```
# create a ptable using functionality from the ptable-pkg
# a single ptable for all cells
ptab1 <- ptable::pt_ex_nums(parity = TRUE, separation = FALSE)

# a single ptab for all cells except for very small ones
ptab2 <- ptable::pt_ex_nums(parity = TRUE, separation = TRUE)

# different ptables for cells with even/odd number of contributors
# and very small cells
ptab3 <- ptable::pt_ex_nums(parity = FALSE, separation = TRUE)

p_nums1 <- ck_params_nums(
  ptab = ptab1,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.30, 0.03),
    epsilon = c(1, 0.5, 0.2),
    q = 3),
  mu_c = 2,
  same_key = FALSE,
  use_zero_rkeys = FALSE,
  path = f_yaml)

# we read the parameters from the yaml-file
p_nums1 <- ck_read_yaml(path = f_yaml)

# for variables with positive and negative values
p_nums2 <- ck_params_nums(
  ptab = ptab2,
  type = "top_contr",
  top_k = 3,
  mult_params = ck_flexparams(
    fp = 1000,
    p = c(0.15, 0.02),
    epsilon = c(1, 0.4, 0.15),
    q = 3),
  mu_c = 2,
  same_key = FALSE)

# simple perturbation parameters (not using the flex-function approach)
p_nums3 <- ck_params_nums(
  ptab = ptab3,
  type = "mean",
  mult_params = ck_simpleparams(p = 0.25),
  mu_c = 2,
  same_key = FALSE)

# use `p_nums1` for all variables
tab$params_nums_set(p_nums1, c("savings", "income", "expend"))
```

```
# use different parameters for variable `mixed`
tab$params_nums_set(p_nums2, v = "mixed")

# identify sensitive cells to which extra protection (`mu_c`) is added.
tab$supp_p(v = "income", p = 85)
tab$supp_pq(v = "income", p = 85, q = 90)
tab$supp_nk(v = "income", n = 2, k = 90)
tab$supp_freq(v = "income", n = 14, weighted = FALSE)
tab$supp_val(v = "income", n = 10000, weighted = TRUE)
tab$supp_cells(
  v = "income",
  inp = data.frame(
    sex = c("female", "female"),
    "age" = c("age_group1", "age_group3")
  )
)

# perturb variables
tab$perturb(v = c("income", "savings"))

# extract results
tab$numtab("income", mean_before_sum = TRUE)
tab$numtab("income", mean_before_sum = FALSE)
tab$numtab("savings")

# results can be resetted, too
tab$reset_cntvars(v = "cnt_males")

# we can then set other parameters and perturb again
tab$params_cnts_set(val = p_cnts1, v = "cnt_males")

tab$perturb(v = "cnt_males")

# write results to a .csv file
tab$freqtab(
  v = c("total", "cnt_males"),
  path = file.path(tempdir(), "outtab.csv")
)

# show results containing weighted and unweighted results
tab$freqtab(v = c("total", "cnt_males"))

# utility measures for a count variable
tab$measures_cnts(v = "total", exclude_zeros = TRUE)

# modifications for perturbed count variables
tab$mod_cnts()

# display a summary about utility measures
tab$summary()
```

ck_vignette *ck_vignette*

Description

starts the package vignette that gets you started with the package

Usage

```
ck_vignette()
```

Value

a browser windows/tab with showing the vignette

Examples

```
## Not run:
ck_vignette()

## End(Not run)
```

testdata *A real-world data set on household income and expenditures*

Description

4580 Obervations in 15 Variables; This dataset also contains sampling weights!

Format

testdata: a data frame with 4580 observations on the following 15 variables.

- `urbrur`: a numeric vector
- `roof`: a numeric vector
- `walls`: a numeric vector
- `water`: a numeric vector
- `electcon`: a numeric vector
- `relat`: a numeric vector
- `sex`: a numeric vector
- `age`: a numeric vector
- `hhcivil`: a numeric vector
- `expend`: a numeric vector

- income: a numeric vector
- savings: a numeric vector
- ori_hid: a numeric vector
- sampling_weight: a numeric vector
- household_weights: a numeric vector

References

The International Household Survey Network, www.ihsn.org

Examples

```
data(testdata)
head(testdata)
```

Index