CONFERENCE OF EUROPEAN STATISTICIANS

Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality Session 7

29-31 October 2019, the Hague, the Netherlands 24 September 2019

cellKey - consistent perturbation of statistical tables

Bernhard Meindl (Statistics Austria), Tobias Enderle (Destatis, Germany)

bernhard.meindl@statistik.gv.at, *Tobias.Enderle@destatis.de*

Abstract and Paper

This contribution presents the R (R Core Team 2019) package cellKey which can be used to apply the cellkey method (CKM) (Leaver and Marley 2011) to statistical tables. The main idea of this post-tabular Statistical Disclosure Control (SDC) technique is to perturb table cells with random noise which is defined via some noise distribution in a consistent and reproducable way. Thus it is ensured that any given table cell to which the same units contribute is always perturbed identically.

The package makes use of the ptable package which allows to define probabilities of the random noise in form of ptables. Such ptables define probabilities of transitioning from a start value (which are often the orginal cell counts) to a target values and can be used as input for R package cellKey and also τ-argus (Hundepool et al. 2004).

The package was developed within the framework of the Eurostat-funded project "*Open Source tools for perturbative confidentiality methods*". This project involves the practical implementation of the cellkey method into SDC tools. To simplify testing and development, separate packages are provided that implement the method (cellKey) and the generation of the required perturbation tables (ptable) in R. As suggested in (Leaver and Marley 2011) and illustrated in Giessing (2016), the (ptable) package implements a maximum entropy approach to compute the transition probabilities taking into account certain thresholds of the desired noise distribution. cellKey makes use of these perturbation tables and allows to compute perturbed (weighted) frequency and magnitude tables. It allows for an easy way to specify complex hierarchies using the sdcHierarchies and proves to be quite fast

cellKey - consistent perturbation of statistical tables

Bernhard Meindl (bernhard.meindl@statistik.gv.at)
Tobias Enderle (Tobias.Enderle@destatis.de)

September 2019

Contents

1 Abstract

This contribution presents the `R` (R Core Team 2019) package `cellKey` which can be used to apply the cellkey method (`CKM`) (Leaver and Marley 2011) to statistical tables. The main idea of this post-tabular Statistical Disclosure Control (`SDC`) technique is to perturb table cells with random noise which is defined via some noise distribution in a consistent and reproducable way. Thus it is ensured that any given table cell to which the same units contribute is always perturbed identically.

The package makes use of the `ptable` package which allows to define probabilities of the random noise in form of perturbation tables (`ptables`). Such `ptables` define probabilities of transitioning from a start value (which are often the orginal cell counts) to a target values and can be used as input for package `cellKey` and also τ-argus (Hundepool et al. 2004).

The package was developed within the framework of the Eurostat-funded project *"Open Source tools for perturbative confidentiality methods"* (Eurostat 2019). This project involves the practical implementation of the cellkey method into SDC tools. To simplify testing and development, separate packages are provided that implement the method (`cellKey`) and the generation of the required perturbation tables (`ptable`) in `R`. As suggested in (Leaver and Marley 2011) and illustrated in Giessing (2016), the (`ptable`) package implements a maximum entropy approach to compute the transition probabilities taking into account certain thresholds of the desired noise distribution. `cellKey` makes use of these perturbation tables and allows to compute perturbed (weighted) frequency and magnitude tables. It allows for an easy way to specify complex hierarchies using the `sdcHierarchies` package (Meindl 2019a) and proves to be quite fast.

2 Methodological aspects

In this section we are summarizing some differences between the approach that was implemented in `R` packages `ptable` and `cellKey` and the CKM proposed by the Australian Bureau of Statistics (ABS) (Leaver and Marley 2011). The method was developed in order to protect tables that were produced "on-the-fly" and are available online via a web front end. The main advantage of this post-tabular anonymisation technique is that it provides consistent perturbation across all table cells by design. The method builds on microdata. Each record is then assigned a record key (`rkey`). To all records contributing to a cell, some function is applied to compute a cell key (`ckey`). These cell keys are then used to look up a perturbation value in a `ptable` that is added to the original (weighted) cell counts to compute the perturbed cell value. Thus, in any dynamically produced table identical cells (those to which the same records and, hence, same record keys contribute) have the same cell key and thus the same perturbed value.

The main disadvantage of the `CKM` is that is is typically not additive. As cells are perturbed independently it is not guaranteed that linear constraints are respected. Therefore, perturbed values in marginal sums usually differ from the sum of the inner cells contributing to this (sub)total.

2.1 Perturbing count tables

One of the main differences between the original method and the `CKM` implemented in package `cellKey` refers to the generation of the record keys. In this implementation, the record keys are assumed to be derived from the standard uniform distribution with minimum 0 and maximum 1.

The cell keys are then derived using the following formula:

$$ckey_i = \sum_{i=1}^{n} rkey_i \mod 1$$

The computation of a cellkey for a specific cell i is performed by adding up the record keys of the n contributing units to this cell and using the remainder (modulo operation) of a division by 1 as the cell key.

Another simplification refers to the format of the perturbation tables which can be easily generated using `R` package `ptable`. The package returns perturbation tables in a quite generic format that can (and will) be also used to hold perturbation values for magnitude tables. The format is as simple as shown in Table `1`:

Table 1: First 6 Rows of a perturbation table to illustrate its format

i	p	v	p_int_lb	p_int_ub
0	1.0000000	0	0.0000000	1.0000000
1	0.7048744	-1	0.0000000	0.7048744
1	0.1805023	2	0.7048744	0.8853767
1	0.1146233	3	0.8853767	1.0000000
2	0.4103289	-2	0.0000000	0.4103289
2	0.3967114	1	0.4103289	0.8070402

The columns of the `ptable` are described below:

- `i`: defines a `"block"` in which a specific perturbation value can be found.
- `p`: probability that this perturbation value is selected for a given block `i`
- `v`: the perturbation value (i.e. noise) that needs to be added to an unperturbed (weighted) cell value

Columns `p_int_lb` and `p_int_ub` directly depend on `p` and are lower and upper bounds of a probability interval:

- `p_int_lb`: lower (inclusive) bound of a cumulative probability interval for noise `v` in a given block `i`
- `p_int_ub`: upper (exclusive) bound of a cumulative probability interval for noise `v` in a given block `i`

To find a perturbation value (column `v`) for a given cell it is required to compare the unperturbed cell value f with column `i` of the perturbation table. We identify all rows where `min(f, max(i)) == i` holds. The next step is to compare the cell key $ckey_i$ of this cell to the cumulative probability interval defined by columns `p_int_lb` and `p_int_ub`. We finally select the perturbation value (column `v`) of the row where the interval contains the cell key. This value `v` is then added to `f` to derive the perturbed cell value that can be published.

Having a look at the first few rows of the perturbation table shown in Table `1`, we can say that for example an original cell value of `0` will never be perturbed as there is only one row where `i == 0` and the probability intervall of this row is [0; 1). Thus, any cell with unperturbed frequency of `0` will be perturbed with value `0` (column `v`) and remains unchanged. If a cell has an original value of `1`, there are three possible cases:

- case 1: `v = -1` with probability `p` being ~ 70.49%
- case 2: `v = 2` with probability `p` being ~ 18.05%
- case 3: `v = 3` with probability `p` being ~ 11.46%

This would result in perturbed cell values of `0`, `3` and `4` for cases 1, 2 and 3, respectively.

2.2 Perturbing magnitude tables

An important task in the project *"Open Source tools for perturbative confidentiality methods"* was to generalize the method to perturb continuous variables and to produce perturbed magnitude tables. The project team came up with several propositions that differed from the original `ABS` implementation while still trying to be close to the original proposition. The work to create perturbed magnitude tables is however ongoing so details may still change.

The basic idea is to compute a perturbed cell value based on the following formula

$$pWY = Y + \sum_{i=1}^{top_k} f(y_i) \cdot m_i \cdot d_i \cdot v_i$$

where $Y = \sum_{i=1}^{n} w_i \cdot y_i$ is the unperturbed weighted cell value consisting of values y_i of observation i with corresponding weights w_i. The second term is the total amount of perturbation that should be added to cell `i` for the top_k top-contributors to this cell. In the implementation it is assumed that the contributions y_i are ordered by absolute descending values.

According to the formula, the noise term is computed by adding top_k separate noise components. Each of these components is in turn constituted by three multiplicative components:

1. data dependent components $f(y_i)$
2. fixed, user defined factors m_i
3. random components $d_i \cdot v_i$ where d_i refers to the direction and v_i to the amount

The project suggested and will implement extensions and enhancements to cases `1-3` as described below:

1. In the original `CKM` proposition, the data-dependent component only depended on the top_k largest contributors to a cell. The generalisation we propose is to additionally offer the following alternatives:

- **cell mean**: $f(y_i) = \dfrac{\sum_{i=1}^{n} y_i \cdot w_i}{\sum_{i=1}^{n} w_i}$

- **cell spread**: $f(y_i) = y_i - y_n$
- **original cell value**: $f(y_i) = \sum_{i=1}^{n} y_i \cdot w_i$

In these cases, top_k is set to `1`. These variants would allow to implement the proposed method also for future table builders that do not internally have the possibility to identify the top contributions to a table cell.

2. The factors m_i defining the magnitude of the perturbation should depend on the cell values. Thus, it was proposed to use a so called '*"flex function"* approach. The idea is that users need to provide a flexpoint fp and two percentages (σ_0 and σ_1) referring to desired magnitudes for large and small cell values. The flex function is designed in a way that for large $x_j = f(y_j) > fp$, the function tends to σ_0 while for smaller values of x_j tending to the flexpoint the functions approaches σ_1.

The formula below shows the two cases that are considered:

$$m_j(x_j) = \begin{cases} \sigma_0 \cdot \left(1 + \dfrac{\sigma_1 \cdot x_j - \sigma_0 \cdot fp}{\sigma_0 \cdot fp} \cdot \left(\dfrac{2 \cdot fp}{fp + x_j}\right)^q\right) & x_j > fp \\ \sigma_1 & x_j \leq fp \end{cases}$$

We apply the flex-function for all values of $x_j > fp$ and use a fixed percentage σ_1 otherwise. This approach can easily be adapted to the case where users request a fixed *"minimal"* noise variance for very small cells.

3. Referring to the third component, the project generalizes the approach proposed in (Leaver and Marley 2011) by suggesting how to compute a random component $d_i \cdot (\mu_c + |v_i|)$ depending on record- or rather cell keys. The approach also included the enhancement of including an additional parameter μ_c which is defined as some extra, fixed noise for sensitive cells. Such cells may be identified for example by computing dominance rules for each cells and setting $\mu_c > 0$ for all sensitive cells. This amount is then added only to the largest contributor of each sensitive cell. As direction value d_i we take the $sign(v_i)$ where v_i is the result of a lookup-step where the amount of perturbation v_i is extracted from a given perturbation table.

The proposal for the perturbation of continuous variable contains some additional features, too. For example it is described that users should be able to select between two different choices of record keys depending on how to deal with units that contribute to the original cell key but do not contribute to some specific magnitude table. In this case, users will be able to select if the cell key is computed based on all units or only on those with a real contribution.

When defining the perturbation tables for magnitude tables, it is easy to see that it is not feasible to use exactly the same approach as in for frequency tables because it is virtually impossible to include blocks for each possible cell value. Thus, the suggested approach is based on computing convex combinations of two different lookup results. In order to illustrate this approach, Table 2 gives an example of a possible perturbation table for continuous variables:

Table 2: A possible perturbation table for continuous variables

i	p	p_int_lb	p_int_ub	v
0	1.0000000	0.0000000	1.0000000	0.0
1	0.0279104	0.0000000	0.0279104	-1.0
1	0.0946509	0.0279104	0.1225613	-0.5
1	0.1661569	0.1225613	0.2887181	0.0
1	0.0554030	0.2887181	0.3441212	0.5
1	0.1262204	0.3441212	0.4703416	1.0
1	0.0863732	0.4703416	0.5567148	1.5
1	0.0604517	0.5567148	0.6171665	2.0

i	p	p_int_lb	p_int_ub	v
1	0.1010505	0.6171665	0.7182170	2.5
1	0.0000280	0.7182170	0.7182450	3.0
1	0.0829378	0.7182450	0.8011827	3.5
1	0.0507807	0.8011827	0.8519634	4.0
1	0.0057703	0.8519634	0.8577337	4.5
1	0.0507183	0.8577337	1.0000000	5.0
5	0.0102148	0.0000000	0.0102148	-5.0
5	0.0256971	0.0102148	0.0359119	-4.5
5	0.0403097	0.0359119	0.0762216	-4.0
5	0.0188341	0.0762216	0.0950557	-3.5
5	0.0278931	0.0950557	0.1229488	-3.0
5	0.0227406	0.1229488	0.1456894	-2.5
5	0.0199531	0.1456894	0.1656425	-2.0
5	0.0680307	0.1656425	0.2336732	-1.5
5	0.0414619	0.2336732	0.2751350	-1.0
5	0.1223925	0.2751350	0.3975275	-0.5
5	0.0886387	0.3975275	0.4861662	0.0
5	0.0213716	0.4861662	0.5075379	0.5
5	0.0212988	0.5075379	0.5288367	1.0
5	0.0577759	0.5288367	0.5866125	1.5
5	0.0727247	0.5866125	0.6593373	2.0
5	0.0115308	0.6593373	0.6708680	2.5
5	0.0212908	0.6708680	0.6921588	3.0
5	0.0186559	0.6921588	0.7108147	3.5
5	0.0653811	0.7108147	0.7761958	4.0
5	0.0043178	0.7761958	0.7805136	4.5
5	0.1464493	0.7805136	1.0000000	5.0

Table 2 features 3 different blocks with values of `i` being `0`, `1` and `5`. We note however, that more general perturbation tables could contain more than three different blocks to fine-tune the desired perturbation results. To lookup a perturbation value for a given cell `i`, we cannot use the (weighted) cell value but a function. Let $a = y_1/x$ with x being the weighted cell value of cell `i` and y_1 the largest contribution. We then have two possibilities. If $a > max(i)$ or a exactly matches a unique value in column `i`, we only look in the block of the perturbation table where `a == i` or `a == max(i)` in the second case where $a > max(i)$. We then select the perturbation value just as in the case for the frequency variables depending on which cumulative perturbation interval (variables `p_int_lb` and `p_int_ub`) contains the value of the cell key for the given cell.

In the other case the idea is to compute a weighted combination of two different perturbation values as it will now be discussed in an example. We assume that for a specific cell we obain $a = 3.5$ and the cell key equals `0.2`. We can compute a perturbation value v using the following steps:

1. let a_0 the largest i smaller than a. In this example we obain $a_0 = 1$.
2. let a_1 the smallest i larger than a; we get $a_1 = 5$.
3. find perturbation values $V(a_0, ckey = 0.2)$ and $V(a_1, ckey = 0.2)$. We obain two different perturbation values from column `v`: 0 for the block with `i == 1` and -1.5 for block where `i == 5` holds true.
4. the last step is to compute a weighted combination of $V(a_0, ckey = 0.2) = 0$ and $V(a_1, ckey = 0.2) =$ -1.5. With $\lambda = \frac{a-a_0}{a_1-a_0}$ which equals 0.625 in our example, we can compute a weighted average with the following formula:

$$V(z, a) = (1 - \lambda) \cdot V(z, a_0) + \lambda \cdot V(z, a_1)$$

In the example we obain a final perturbation value for this cell as -0.9375.

We note that in contrast to the `ptable` for frequency variables, column `i` is not limited to integers but can contain real numbers, too. The methodological work in the project also showed how this table sturucture may be even more generalized by providing different perturbation tables for example for even and odd numbers. This could simply be achieved by adding an additional column specifying if a block is valid for even/odd numbers.

3 A practical example

In this section, we demonstrate the features of the `R` package `cellKey`. This package is at the time of writing this contribution under heavy development. For the following examples, the package was build from the latest development snapshot. The package implements the `CKM` method for count and continuous variables. Internally, it makes use of package `ptable` which contains features to generate appropriate perturbation tables.

In general, to implement the `CKM` method, two pillars need to be brought together.

1. Rules model: generate transition and/or perturbation matrix $\rightarrow$ Package `ptable`
2. Implementation model: microdata, record- and cell keys/lookups $\rightarrow$ Package `cellKey`

In the project, the `CKM` is to implemented both in τ-argus and in `R`. Thus it is especially important to make sure that both implementations return exactly the same results. Thus it was decided that both `cellKey` and τ-argus use perturbation tables generated from the `ptable` package as input.

As already mentioned, the `cellKey` package is currently under heavy development in order to make its application as easy as possible. For this reason, the package was rewritten in an object-oriented way using R6 reference classes (Chang 2019). This approach allows it to bind methods directly to objects and call them in a very user-friendly way as shown later.

The general workflow of using the package is to define the structure of the table by specifying hierarchies. This can be achieved using functionality from the `sdcHierarchies` package. Then a suitable object needs to be defined that contains information on the hierarchies as well as the count- and continuous variables that may be perturbed. Internally, functions from the `sdcTable` package (Meindl 2019b) are used to build the complete table based on the micro data input. After such an object has been created, several methods can be applied to such objects such as assigning perturbation parameters to variables, computing the perturbation patterns, listing results and computing utility measures.

We now walk through the process of computing perturbed frequency and magnitude tables starting from a micro data set.

3.1 Preparations

We now show a typical application of the `cellKey` package. The goal is to define a simple two-dimensional table and creating perturbed count- and magnitude tables using different sets of parameters. This example is based on development code and might slightly change for the final version.

After loading the package, we can start the example by loading an included test microdata set to which we will add two more variables.

```
library(cellKey) # devtools::install_github("sdcTable/cellKey", ref = "next")
x <- ck_create_testdata()

# create an additional count variable
x[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
```

The first few rows of important variables of the dataset `x` look like this:

sex	age	cnt_highincome	income	savings	sampling_weight
male	age_group3	0	5780	12	96
female	age_group3	0	2530	28	57
male	age_group1	0	6920	550	49
male	age_group1	0	7960	870	45
male	age_group4	1	9030	20	57
female	age_group3	0	3290	102	44

The next required step is to add record keys to the dataset. As previously mentioned, the record keys are assumed to come from a standard uniform distribution which will be internally enforced in the package. Function `ck_generate_rkeys()` allows to generate such keys. We note that in order to achieve consistency, a seed depending on a hash of the input data object is internally used. So it is ensured that the same record keys are always generated for the same input object.

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

However, users must be aware that any changes in the microdata set regarding the columns (e.g. editing or adding variables) will change the input object which causes a new hash and, hence, different record keys. To ensure consistent perturbed tables, record keys should not be changed or replaced. Since variables will be changed, we can set the argument `seed=` that ensures identical record keys independent from such changes.

```
# create record keys using the seed argument
x$rkey <- ck_generate_rkeys(dat = x, seed=1234)
```

Any changes regarding the rows of the microdata (e.g. adding or removing observations) after record keys have been attached must be carried out very carefully.

After adding the record keys, the next step is to specify the hierarchies. In this example, we have two variables - `age` and `sex` - that define the tables we want to generate. The hierarchies can be created using the `hier_*` namespace from the `sdcHierarchies` package which is automatically available when the `cellKey` package was loaded.

```
# hierarchy with some bogus codes
d_sex <- hier_create(root = "Total", nodes = c("male", "female"))
d_age <- hier_create(root = "Total", nodes = paste0("age_group", 1:6))
```

In our example, both dimensional variables `sex` and `age` only contain a single top-node that is computed from some levels. Thus, the hierarchies can be generated using `hier_create()`. In case of more complex hierarchies, it may be required to use `hier_add()` to add more levels to a hierarchy. For examples, see `?hier_add`.

We continue our example by calling `ck_setup()` which creates a cell key object we can modify afterwards. The function requires a data set (`x`) and a named list containing dimensions in each element (argument dims) as intput. Furthermore, additional and optional arguments can be the name of the variable containing record keys (rkey), sampling weights (w), count- and numeric variables (argument countvars and numvars).

```
tab <- ck_setup(
  x = x, rkey = "rkey", dims = list(sex = d_sex, age = d_age), w = "sampling_weight",
  countvars = c("cnt_highincome"), numvars = c("income", "savings"))
```

Once an object is created, all implemented methods can be called with the synax objectname$methodname(). For example, we can get some information about the created object with the print-method. This method shows for example a square [] before each variable which is empty by default. For variables that have already been perturbed, [x] is displayed.

```
cat(tab$print())
```

```
## -- Table Information ----------------------------------------------------
## v 21 cells in 2 dimensions ('sex', 'age')
## v weights: yes
## -- Tabulated / Perturbed countvars --------------------------------------
## [ ] 'total'
## [ ] 'cnt_highincome'
## -- Tabulated / Perturbed numvars ----------------------------------------
## [ ] 'income'
## [ ] 'savings'
```

Wecan also ask which variables have been defined using methods allvars(), cntvars() and numvars(). We note that it is not required to generate count variable for the entire table as variable total is automatically generated.

```
tab$cntvars() # count-variables
```

```
## [1] "total"          "cnt_highincome"
```

```
tab$numvars() # continuous variables
```

```
## [1] "income"  "savings"
```

```
# tab$allvars() # would show both count- and numvars
```

3.2 Perturbing count variables

We have already generated a table instance, but in order to actually perturb variables, we need to create suitable perturbation tables. For count variables, we can use function ck_params_cnts() to generated a suitable input and method params_cnts_set() to attach such parameters to one or more count variables. Thus, it is possible to use different perturbation tables for different variables. Function ‘ck_params_cnts() basically just wraps functionality from the ptable package. Thus, to get information about the required parameters one could use either ?ptable::pt_create_pTable or ?ck_params_cnts.

We now define two different perturbation parameters for count variables as shown below and attach them to variables total and cnt_highincome respectively.

```r
# first parameter set used for count-variable `total`
p_cnts1 <- ck_params_cnts(D = 5, V = 3, js = 2, pstay = 0.5)
tab$params_cnts_set(val = p_cnts1, v = "total")

# different parameters used for count-variable `cnt_highincome`
p_cnts2 <- ck_params_cnts(D = 8, V = 3, js = 2, pstay = 0.5)
tab$params_cnts_set(val = p_cnts2, v = "cnt_highincome")
```

After perturbation parameters have been attached to a variable, we can use the `perturb()` method to actually perturb those variables as shown below.

```r
tab$perturb(v = c("total", "cnt_highincome"))
```

We note that it is perfectly possible to perturb just a single variable in one run by only specifying a single variable name in argument `v`. After the variables have been perturbed, we can have a look at the results with the `freqtab()` method.

```r
head(tab$freqtab(v = "total"))
```

```
##      sex        age vname  uwc     wc puwc       pwc
## 1: Total      Total total 4580 277138 4578 277016.98
## 2: Total age_group1 total 1969 118430 1971 118550.29
## 3: Total age_group2 total 1143  68624 1144  68684.04
## 4: Total age_group3 total  864  52642  864  52642.00
## 5: Total age_group4 total  423  26299  424  26361.17
## 6: Total age_group5 total  168  10477  168  10477.00
```

This method returns a `data.table` that contains for each cell (defined by the dimensional variables `age` and `sex`) and the variable name (column `vname`) the unperturbed unweighted (`uwc`) and weighted (`wc`) as well as perturbed unweighted (`puwc`) and weighted (`pwc`) counts.

After a variable has been perturbed, we can compute some utility measures showing the impact of the perturbation using the `measures_cnts()` method. For both the `freqtab()` and the `measures_cnts()` method it is possible to specify multiple (perturbed) variables in argument `v`. The method returns a list with different measures that are discussed in detail in the help page which can be accessed with `?cellkey_pkg`.

Element `overview` returns for example a `data.table` showing the distribution of noise that has been applied as the result below shows:

```r
tab$measures_cnts(v = "total")$overview
```

```
##    noise cnt        pct
## 1:    -2   1 0.04761905
## 2:    -1   3 0.14285714
## 3:     0  13 0.61904762
## 4:     1   1 0.04761905
## 5:     2   2 0.09523810
## 6:     3   1 0.04761905
```

3.3 Perturbing continuous variables

The required steps to compute perturbed magnitude tables are very similar. Again we have to note however, that the design of the interface is not entirely fixed and may slightly change. The idea is again to define suitable perturbation tables and parameters and attach them to continuous variables defined in `ck_setup()`. To create continous parameters, we can use the `ck_params_nums()` function which takes quite a few parameters that are described in detail in `?ck_params_nums`.

```
p_nums1 <- ck_params_nums(
  D = 10, l = 0.5, type = "top_contr", top_k = 3, pos_neg_var = 0,
  mult_params = ck_flexparams(
    flexpoint = 1000,
    m_small = 0.30,
    m_large = 0.03,
    epsilon = c(1, 0.5, 0.2),
    q = 3))
```

Object `p_nums1` now contains the perturbation parameters for continuous variables based on the `top_k` approach (the total perturbation added to each cell depends on each of the `top_k = 3` contributions to the cell) as well as on the flex-function approach described in this contribution. Auxiliary function `ck_flexparams()` allows to set the parameters of this approach. Furthermore, by setting argument `pos_neg_var = 0` it is ensured that any perturbed variable has values `>= 0` even after the perturbation step.

Finally we attach these parameters to the continuous variables `savings` and `income` using method `params_nums_set()`.

```
tab$params_nums_set(p_nums1, c("savings", "income"))
```

In this example, we use the same parameters for both defined numerical variables. It is however possible - just as for count variables - to attach different parameters to different variables. This is especially useful if we need to differentiate between strictly positive variables (like `savings` and `income` in this example) and variables that can also contain negative values.

Perturbing a continuous variable is then again performed by applying the `perturb()` method and specifying the variables that should be perturbed.

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

After this step, we can extract the results using the `numtab()` method. Below we show the results for variable `savings`.

```
head(tab$numtab("savings", mean_before_sum = FALSE))
```

```
##      sex        age   vname     uws        ws       pws
## 1: Total      Total savings 2273532 137343927 137316236
## 2: Total age_group1 savings  982386  58916083  58942992
## 3: Total age_group2 savings  552336  33250976  33280676
## 4: Total age_group3 savings  437101  26585878  26584326
## 5: Total age_group4 savings  214661  13401940  13401664
## 6: Total age_group5 savings   80451   4823814   4826383
```

The `numtab()` method returns a `data.table` containing for each cell in the table defined by the dimensional variables a column identifying the continuous variable (column `vname`) as well as the unperturbed unweighted (`uws`) and weighted (`ws`) and the perturbed weighted (column `pws`) values.

We note that in numtab() it possible to set argument mean_before_sum. If set to TRUE, the perturbed values are adjusted by a factor $\frac{n+p}{n}$ with n being the original weighted cellvalue and p the perturbed cell value. This makes sense if the the accuracy of the variable mean is considered more important than accuracy of sums of the variable. The default value of this argument is FALSE so that no adjustment is done.

If analyzing the results revealed that the perturbation pattern needs to be changed, it is possible to reset variables by using the reset_allvars() (for all perturbed variables), reset_cntvars() or reset_numvars() methods where the latter two methods can also be used to reset specific variables by specifying the desired variables in argument v. Then, some other perturbation parameters can attached to these variables using params_cnts_set() or params_nums_set() and the variable(s) can be perturbed again using the perturb() method.

4 Summary and outlook

This contribution summarized the features of R package cellKey that implements CKM. It is currently under development and once it is stable, a version will be published on the Comprehensive R Archive Network CRAN. Some features that are already implemented have not been shown in this work such as the possibility to enforce strictly positive variables even after perturbation or the fact that cell keys are scrambled in case top_k > 1 holds true. This allows - in difference to the original proposition of the method - only to keep track of cell keys but not individual record keys while still allowing that in the lookup step the perturbation value depends on the amount of the individual contribution as well as a (scrambled) record key. Another feature that is already implemented is that users may opt to use cell keys in order to exclude units that do not contribute to a given continuous variable.

The key priority is to properly implement the main feature set and publish a stable version within the timeframe of the project. However, the package was designed in a way that it will be easily expandable in the future. Several ideas have already been proposed and include the extension of perturbation tables for magnitude tables to allow different perturbation distributions for cell values with even and odd cell values.

Finally we want to mention that the package is well documented and contains many examples which can be accessed via ?cellkey_pkg. The package also will feature a package vignette giving a complete example on how to generate and compute perturbed frequency and magnitude tables with the package. The package vignette can be started by calling the ck_vignette() function.

References

Chang, Winston. 2019. *R6: Encapsulated Classes with Reference Semantics*. https://CRAN.R-project.org/package=R6.

Eurostat. 2019. "Open source tools for perturbative confidentiality methods." 2019. https://ec.europa.eu/eurostat/cros/content/perturbative-confidentiality-methods_en.

Giessing, S. 2016. "Computational Issues in the Design of Transition Probabilities and Disclosure Risk Estimation for Additive Noise." *Privacy in Statistical Databases* 9867: 237–51.

Hundepool, Anco, Aad van de Wetering, Ramya Ramaswamy, Peter-Paul de Wolf, Sarah Giessing, Matteo Fischetti, Juan Jose Salazar-Gonzalez, Jordi Castro, and Philip Lowthian. 2004. "Tau-Argus Users Manual." *CENEX-Project. BPA*, nos. 769-02.

Leaver, V., and J. K Marley. 2011. "A Method for Confidentialising User-Defined Tables: Statistical Properties and a Risk-Utility Analysis." *Proceedings of 58th World Statistical Congress*, 1072–81.

Meindl, Bernhard. 2019a. *sdcHierarchies: Create and (Interactively) Modify Nested Hierarchies*. {https://cran.r-project.org/package=sdcHierarchies}.

———. 2019b. *sdcTable: Methods for Statistical Disclosure Control in Tabular Data*. {https://cran.r-project.org/package=sdcTable}.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.