Use of function `tuneParameters` in `SamplingStrata` package

Package version 1.1

Giulio Barcaroli

October 27, 2015

Abstract

The package `SamplingStrata` *allows to determine the best stratification of a target population, the one that ensures the minimum sample cost necessary to satisfy precision constraints in a multivariate and multidomain case. The underlying algorithm is based on a non deterministic evolutionary approach, making use of the genetic algorithm paradigm. The performance of the optimization heavily depends on the values given to the different parameters of the function* `optimizeStrata`*. In order to let the user choose a convenient combination of these values, the function* `tuneParameters` *available in the package permits to run a number of optimization tasks by changing in a planned way the parameters values, and to compare the results in order to choose the best combination.*

1 Tuning of parameters and improvement of solutions

The package `SamplingStrata` allows to determine the best stratification of a target population, the one that ensures the minimum sample cost necessary to satisfy precision constraints in a multivariate and multidomain case. The overall approach together with the optimization algorithm are fully described in Ballin and Barcaroli (2013) and Barcaroli (2014).

The performance of the optimization step strongly depends on the values of the parameters that are given to the function `optimizeStrata`. It is worth while to run this step a number of times, by varying the values of the parameters in a planned way, and then to analyze the obtained results in order to choose the best values of the parameters. The function `tuneParameters` allows to run many optimization steps and compare the results in terms of the minimal sample cost required, and also in terms of the CV's that may be expected once having chosen a particular solution. `tuneParameters` can be applied only to a given domain per time. The parameters of this function are of the same nature than those of the function `optimizeStrata`, but they are given in a vectorial format, whose length is given by the number of optimizations to be run: it is therefore possible to define different combination of values of the parameters for each execution of `optimizeStrata`. After each optimization run, from the corresponding optimized frame a given number of samples are drawn. For each of them, the estimates of the target variables Y's are computed, together with the absolute differences between the values of the estimates and the true values in the population, in order to permit a compared evaluation of the different solutions found in the different runs. As the optimal solution is stored for each run, after the evaluation it is possible to use it directly, or as a "suggestion" for a new optimization with more iterations (in order to improve it). First, it is necessary to prepare the data, taking into account the fact that it is possible to apply the function `tuneParameters` to a single domain per time:

```
> library(SamplingStrata)
> data(swissstrata)
> data(swisserrors)
> data(swissframe)
> frame <- swissframe[swissframe$domainvalue == 1,]
> strata <- swissstrata[swissstrata$DOM1 == 1,]
> errors <- swisserrors[swisserrors$domainvalue == 1,]
```

In this case, the first domain has been selected from the `swissframe` dataframe and, consequently, the related strata and constraints on the CV's. Then, we have to set the different parameters necessary to the function. The first two parameters are peculiar to this function:

1. `noptim` defines the number of different optimization runs;
2. `nsampl` indicates how many samples have to be drawn from the population frame whose stratification has been optimized by the current solution.

The remaining parameters are the same utilized by the function `optimizeS-trata`. The only difference is that they must be passed in a vectorial format, whose length is determined by the number of optimization runs. In this way, in the i-th run `optimizeStrata` will receive the combination of parameters values corresponding to the i-th position in all vectors. It is possible to give the same values of some of the parameters for all the runs. In the example, only the parameter related to the `initialStrata` is varying, from a minimum of 10% of the number of initial strata, to a maximum of all of them. We choose to do so because many experiences showed that the optimization step is very sensitive to the values of this parameter. The other parameter which is very important is the 'mutation chance'. So, we define the values of the parameters in this way:

```
> # Number of runs
> noptim <- 8
> # Number of samples to be drawn after each optimization
> nsampl <- 500
> # Number of initial strata
> initialStrata <- ceiling(c(1:noptim)*0.1*(nrow(strata)))
> # Rate for increasing the number of initial strata
> addStrataFactor <- rep(0.01,noptim)
> # Minimum number of units per stratum
> minnumstr <- rep(2,noptim)
> # Number of iterations for each optimization
> iter <- rep(200,noptim)
> # Number of solutions for each iteration
> pops <- rep(20,noptim)
> # Mutation chance
> mut_chance <- rep(0.004,noptim)
> # Elitism rate
> elitism_rate <- rep(0.2,noptim)
```

The function is invoked in this way:

```
> tuneParameters (
+                 noptim,
+                 nsampl,
+                 frame,
+                 errors,
+                 strata,
+                 cens = NULL,
+                 strcens = FALSE,
+                 alldomains = FALSE,
+                 dom = 1,
+                 initialStrata,
+                 addStrataFactor,
+                 minnumstr,
+                 iter,
+                 pops,
+                 mut_chance,
+                 elitism_rate
+                 )

Input data have been checked and are compliant with requirements

GA Settings
  Population size       = 20
  Number of Generations = 200
  Elitism               = 4
  Mutation Chance       = 0.004

[1] FALSE
```

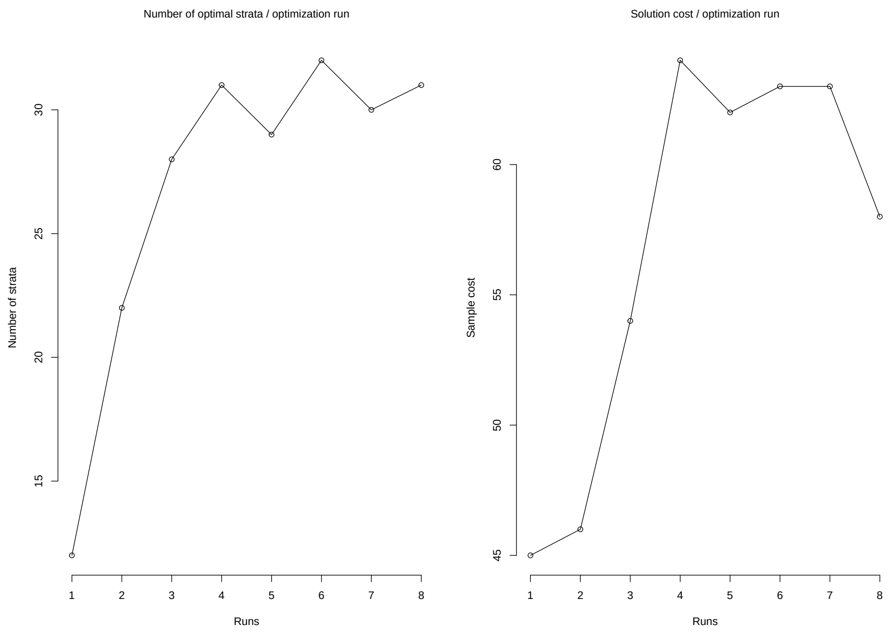

Figure 1: Results of the different optimizations in terms of number of strata and minimal sample cost required varying the values of the parameter 'initial strata'

As already said, for each optimization a number of samples are drawn from the frame stratified accordingly to the corresponding optimal stratification: this allows to calculate the sampling variance for each target variable, expressed in terms of their CV's. The computed CV's are stored in the file 'results.csv', together with the number of strata and the required sample cost:

```
> results <- read.csv("results_1.csv")
> results

  nsimul nstrati cost        CV1        CV2        CV3
1      1      12   43 0.07387714 0.07254699 0.07074644
2      2      22   45 0.07029701 0.06780761 0.06798295
3      3      28   54 0.06314823 0.06383479 0.06326419
4      4      28   53 0.06356393 0.06316806 0.05950538
5      5      34   62 0.05994371 0.05991155 0.05695051
6      6      32   57 0.06610824 0.06545774 0.06570065
7      7      32   61 0.06494032 0.06190481 0.06529612
8      8      32   63 0.06636291 0.06712673 0.06538391
         CV4
1 0.07356932
2 0.06622405
```

```
3 0.06663607
4 0.06456820
5 0.05767928
6 0.07581170
7 0.06657378
8 0.07260232
```

```
> ind <- which(results$cost==min(results$cost))
```

The value of 'ind' indicates the best found solution, and from this we can recall the associated values of the parameters:

```
------------------------------

Best optimization: run #  1

Required sample cost:  43

------------------------------

Values of parameters

------------------------------

initialStrata:  12

addStrataFactor:  0.01

minnumstr:  2

iter:  200

pops:  20

mut_chanc:  0.004

elitism_rate:  0.2

------------------------------
```

Moreover, for each sample, the difference between the sample estimates and the true values of the parameters in the population are also calculated. The distributions of these differences are reported in figure 2.

By comparing the performance of the different optimization runs, the best solutions in terms of sample cost required to comply the constraints on the maximum CV's, can be found. It is related to a precise value of the parameter *initial strata*. We can decide to assume this value as the best one, and then proceed to the tuning of the parameter *mutation chance*. So, we leave unchanged the previous values of the other parameters (with the exception of *initial strata*), and let the values of *mutation chance* vary:

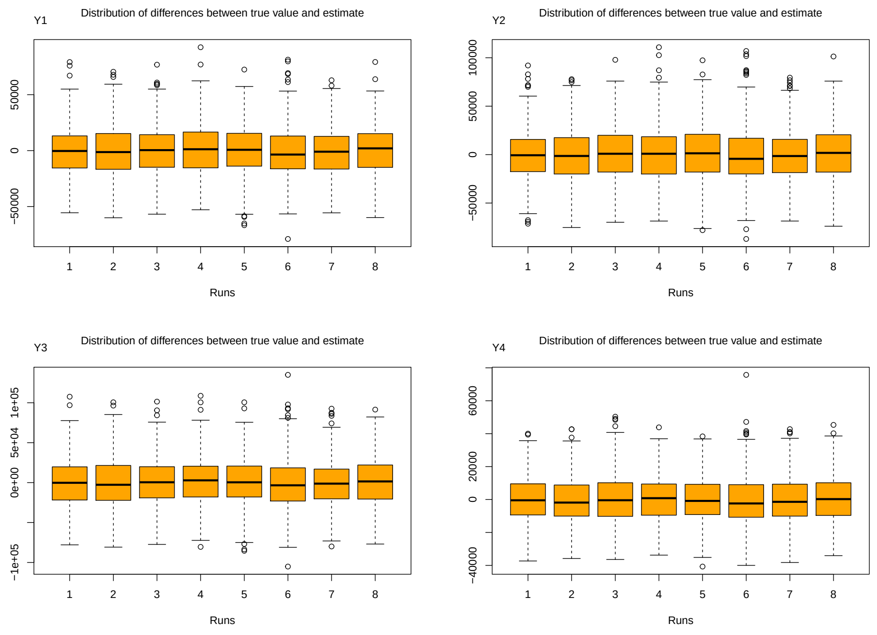

Figure 2: Distributions of the differences between the sampling estimates and the true value of the parameter in the population under each optimization run and for each target variable Y

```
> # Number of initial strata
> bestInitialStrata <- initialStrata[ind]
> initialStrata <- rep(bestInitialStrata,noptim)
> # Mutation chance
> mut_chance <- c(1:noptim)*0.002
```

Then we run again the function `tuneParameters`:

```
> tuneParameters (
+          noptim,
+                nsampl,
+                frame,
+                errors,
+                strata,
+                cens = NULL,
+                strcens = FALSE,
+                alldomains = FALSE,
+                dom = 1,
+                initialStrata,
+                addStrataFactor,
+                minnumstr,
+                iter,
+                pops,
+                mut_chance,
+                elitism_rate
+                )

Input data have been checked and are compliant with requirements

GA Settings
  Population size       = 20
  Number of Generations = 200
  Elitism               = 4
  Mutation Chance       = 0.002

[1] FALSE
```

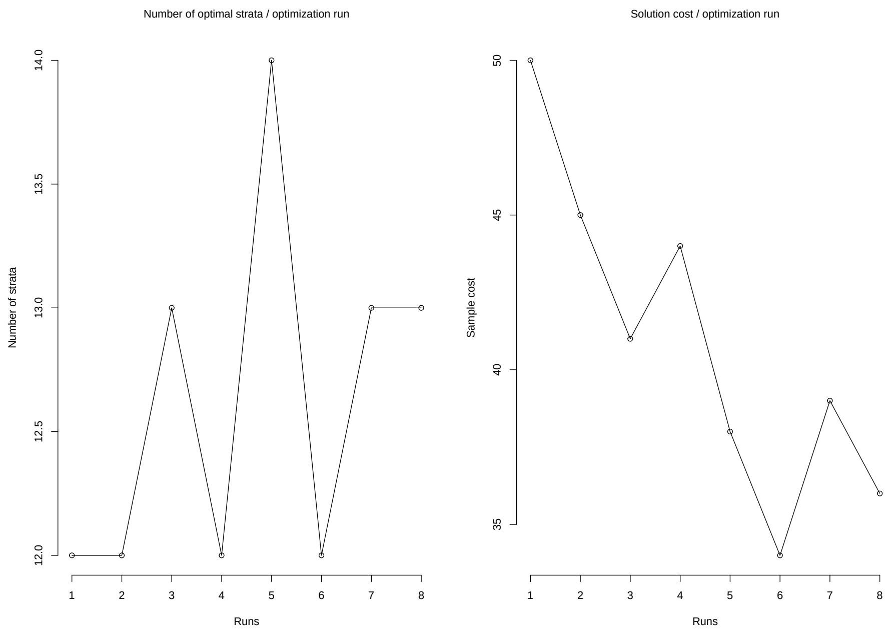

Figure 3: Results of the different optimizations in terms of number of strata and minimal sample size required varying the values of the parameter 'mutation chance'

These are the results:

```
  nsimul nstrati cost        CV1        CV2        CV3
1      1      13   51 0.07233588 0.06770695 0.07034980
2      2      12   47 0.07292869 0.06979252 0.07151529
3      3      12   39 0.07164565 0.07023369 0.06858932
4      4      12   39 0.07397722 0.07368510 0.07019791
5      5      12   40 0.07245175 0.07196241 0.07072005
6      6      12   37 0.07430896 0.07261058 0.07410794
7      7      13   33 0.07020422 0.06870895 0.06726404
8      8      13   39 0.07330091 0.07243773 0.07148431
         CV4
1 0.07089717
2 0.07197497
3 0.07679662
4 0.07098884
5 0.07282606
6 0.07687320
7 0.07206146
8 0.07515864
```

```
------------------------------

Best optimization: run #  7

Required sample cost:  33

------------------------------

Values of parameters

------------------------------

initialStrata:  12

addStrataFactor:  0.01

minnumstr:  2

iter:  200

pops:  20

mut_chance:  0.014

elitism_rate:  0.2

------------------------------
```

So, we can read the best value of the parameter *mutation chance*. At this point we can decide to stop the tuning of the parameters (actually, the two that we considered are the most important, the others being much less influent on the results). As an option, we can improve the best solution so far obtained, by running again the function `optimizeStrata` while increasing the number of iterations. To speed up the convergence of the genetic algorithm, we give the solution previously obtained as a *suggestion* to the genetic algorithm. The solution is read from the .txt external file written at the end of the ind-th optimization run carried out by `tuneParameters`, and introduced in the first row of a matrix (this is the format required by the internal genetic algorithm).

```
> stmt <- paste("suggestions <- read.table('solution_dom1_iter",ind,".txt')",sep="")
> eval(parse(text=stmt))
> sugg <- matrix(nrow=1,ncol=nrow(suggestions),data=suggestions[,1])
```

Then, a new optimization step with the same values of the parameters, but with a greater number of iterations (the double), and with the use of the parameter `suggestion`, is run:

```
> bestMutationChance <- mut_chance[ind]
> solution <- optimizeStrata(
+   errors ,
+         strata ,
+         cens = NULL,
+         strcens = FALSE,
+         alldomains = FALSE,
+         dom = 1,
+         initialStrata = bestInitialStrata,
+         addStrataFactor = 0.01,
+         minnumstr = 2,
+         iter = 1000,
+         pops = 20,
+         mut_chance = bestMutationChance,
+         elitism_rate = 0.2,
+         highvalue = 1e+08,
+         suggestions = sugg,
+         realAllocation = TRUE,
+         writeFiles = TRUE
+         )

GA Settings
  Population size       = 20
  Number of Generations = 1000
  Elitism               = 4
  Mutation Chance       = 0.014


Suggestions
  1 = (1, 2, 3, 1, 1, 1, 4, 5, 3, 6, 6, 7, 7, 4, 7, 8, 9, 8, 7, 9, 7, 10, 7, 7, 10, 9, 6, 10
```

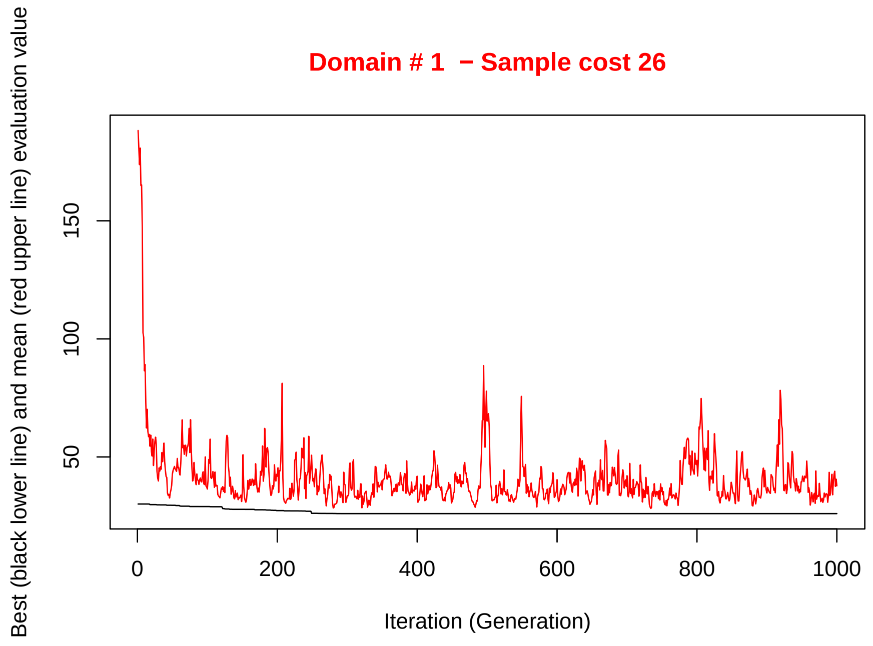

Figure 4: Convergence of the new optimization run

As we can see from the convergence graph, we have improved the previous solution. As usual, we can now proceed with the next operations (updating of the frame and selecting the sample):

```
> newstrata <- updateStrata(strata,solution)
> framenew <- updateFrame(frame,newstrata)
> samp <- selectSample(framenew,solution$aggr_strata)

*** Sample has been drawn successfully ***
 26  units have been selected from  13  strata

==> There have been  1  take-all strata
from which have been selected  2 units
```

Of course, we have to repeat these operations for all the different domains in the frame.

References

Ballin, M. and G. Barcaroli (2013). Joint determination of optimal stratification and sample allocation using genetic algorithm. *Survey Methodology 39*, 369–393.

Barcaroli, G. (2014). SamplingStrata: An R package for the optimization of stratified sampling. *Journal of Statistical Software 61*(4), 1–24.