Package FunQuant part I

Charlie SIRE[1,2,3]

Supervisors: R. LE RICHE[3], D. RULLIERE[3], J. ROHMER[2], L. PHEULPIN[1], Y. RICHET[1]

[1]IRSN

[2]BRGM

[3]Mines Saint-Etienne and CNRS,LIMOS

September 21, 2023

Section 1

Quantization

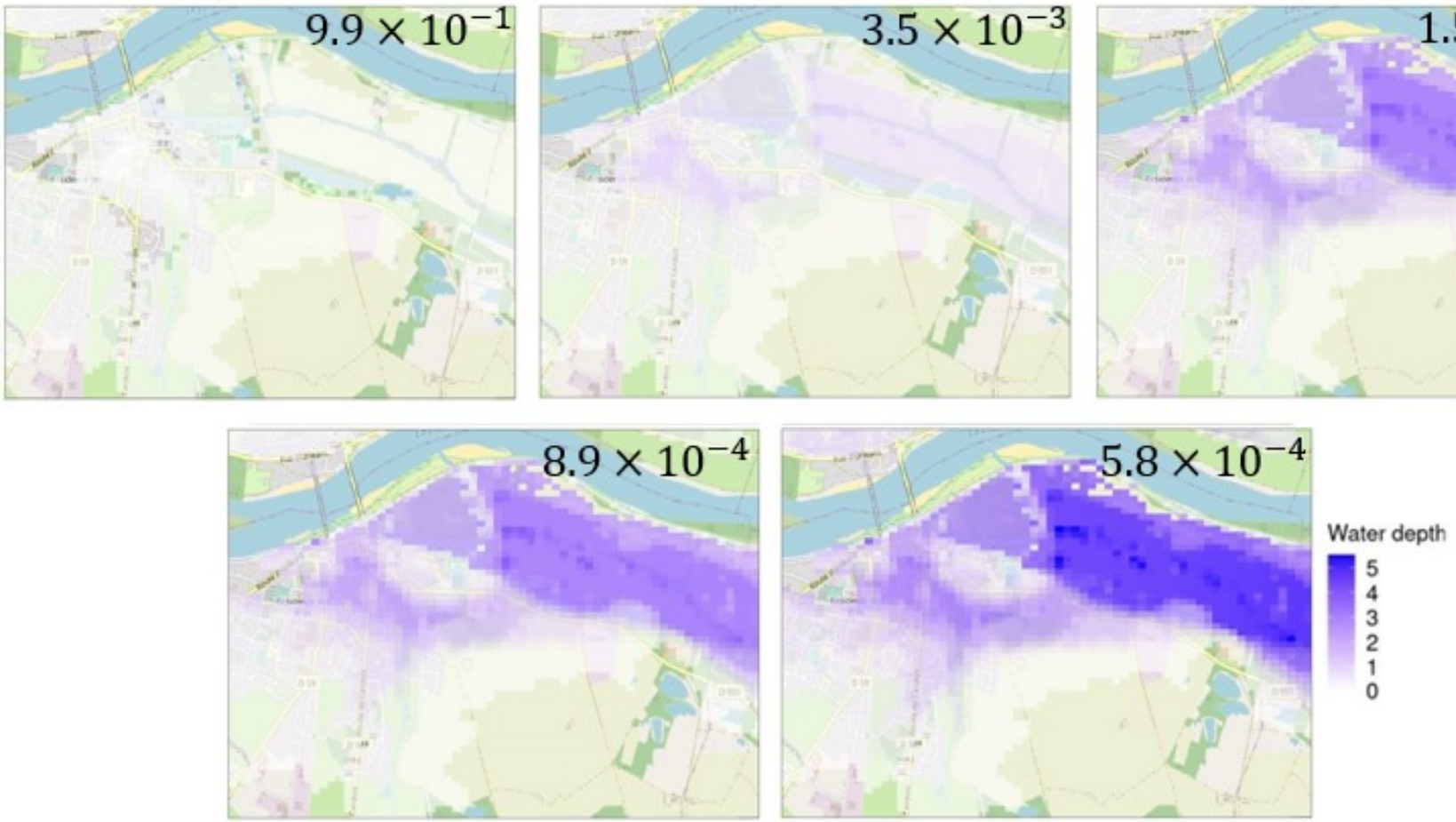
Objectives

Overall objective: Perform quantization in the context of rare events and time-consuming simulations.

Main functionalities:

- Perform Lloyd's algorithm with probabilistics weights
- Assess the variances of the importances sampling to choose the best sampling densities
- Build a metamodel to predict 2D outputs and tune its hyperparameters for quantization

Example of industrial application



Problem formulation

Quantization problem : Find for a given $\ell \in \mathbb{N}$, $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_\ell\} \in \mathcal{Y}^\ell$ ℓ representatives of $Y(X)$

Closest representative map function:

$$\begin{aligned} q_\Gamma : \mathcal{Y} &\to \Gamma \\ y &\mapsto q_\Gamma(y) = \underset{\gamma_i \in \Gamma}{\operatorname{argmin}} \|y - \gamma_i\| \end{aligned}$$

Quantization error: $e(\Gamma) = \left[\mathbb{E}(\|Y(X) - q_\Gamma(Y(X))\|^2)\right]^{\frac{1}{2}}$

Objective: Find

$$\begin{aligned} \Gamma^\star = \{\Gamma_1^\star, .., \Gamma_\ell^\star\} &= \underset{\Gamma \in \mathcal{Y}^\ell}{\operatorname{argmin}} \left(e(\Gamma)\right) \\ &= \underset{\Gamma \in \mathcal{Y}^\ell}{\operatorname{argmin}} \left[\mathbb{E}(\min_{i \in \{1..\ell\}} \|Y(X) - \gamma_i\|^2)\right]^{\frac{1}{2}} \end{aligned}$$

Lloyd's algorithm

Algorithm 1 Lloyd's algorithm

$\Gamma^{[0]} \leftarrow \{\gamma_0^{[0]}, \ldots, \gamma_\ell^{[0]}\}$, $k \leftarrow 0$

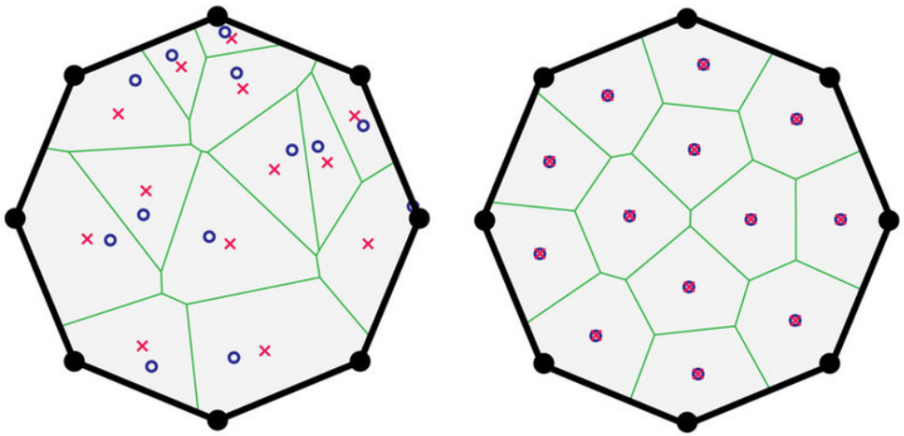
1: **while** not stop **do**

$C_j^{\Gamma^{[k]}} = \{y \in \mathcal{Y} : j = \underset{i \in \{1,\ldots,\ell\}}{\operatorname{argmin}} \|y - \gamma_i^{[k]}\|_{\mathcal{Y}}\}$

$\forall j \in \{1, \ldots, \ell\}, \gamma_j^{[k+1]} \leftarrow \mathbb{E}\left[Y(X) \mid Y(X) \in C_j^{\Gamma^{[k]}}\right]$

$k \leftarrow k + 1$

2: **end while**



Lloyd in case of rare event

The main point is to compute at each iteration the conditional expectation $\mathbb{E}[Y(X) \mid Y(X) \in C_j^{\Gamma}]$

Problem here : In the case of rare events, one prevailing cluster (for instance cluster of empty maps).

Section 2

Integration of Importance sampling

Principle of Importance Sampling

Objective : Estimate $\mathbb{E}[h(Y(X))]$ with $g : \mathcal{Y} \to \mathbb{R}^p$ such as $\mathbb{E}\left[h(Y(X))^2\right] < +\infty$

Examples of function g :

- $g = \mathbb{1}_A$ with $A \subset \mathcal{Y}$ to compute $P(Y(X) \in A)$
- $g : y \to y\mathbb{1}_A$

Main idea : The representation of $\mathbb{E}[h(Y(X))]$ as an expectation is not unique :

$$\mathbb{E}[h(Y(X))] = \mathbb{E}\left[h(Y(\tilde{X}))\frac{f_X(\tilde{X})}{g(\tilde{X})}\right]$$

with $\tilde{X}$ a random variable with density function g with $supp(f_X) \subset supp(g)$

Estimator with importance sampling

From this last representation :

$$\hat{E}_n^{IS} = \frac{1}{n} \sum_{k=1}^{n} h(Y(\tilde{X}^k)) \frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$$

with $(\tilde{X}^k)_{k=1}^n$ be a n-sample of $\tilde{X}$

Its covariance matrix is : $\mathbb{V}(\hat{E}_n^{IS}) = \frac{1}{n}\mathbb{V}(h(Y(\tilde{X}))\frac{f_X(\tilde{X})}{g(\tilde{X})})$

In comparison to $\frac{1}{n}\mathbb{V}(h(Y(X)))$ in a classical MC

Idea : Choose g that minimises variance

Importance sampling combined with quantization

$$\mathbb{E}\left[Y(X) \mid Y(X) \in C_j^\Gamma\right] = \frac{\mathbb{E}\left[Y(X)\mathbb{1}_{Y(X)\in C_j^\Gamma}\right]}{\mathbb{E}\left[\mathbb{1}_{Y(X)\in C_j^\Gamma}\right]}$$

And an estimator of $\mathbb{E}\left[Y(X) \mid Y(X) \in C_j^\Gamma\right]$:

$$\hat{E}_n^{IS}(\Gamma, j, g) = \frac{\frac{1}{n}\sum_{k=1}^n Y(\tilde{X}^k)\mathbb{1}_{Y(\tilde{X}^k)\in C_j^\Gamma}\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}}{\hat{P}_n(\Gamma, j, g)}$$

with $\hat{P}_n(\Gamma, j, g) = \frac{1}{n}\sum_{k=1}^n \mathbb{1}_{Y(\tilde{X}^k)\in C_j^\Gamma}\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$

Algorithm

Algorithm 2 Find prototypes

Input: f_X, ℓ, nb_starts, function_g, threshold

for start in 1:nb_starts do
  Initialize $\Gamma \leftarrow \{\gamma_0, \ldots, \gamma_\ell\} \in \mathcal{Y}^\ell$
  $\Gamma_{old} \leftarrow \Gamma$, dist = $+\infty$, error = $+\infty$
  while dist > threshold do
    for j in $1:\ell$ do
      $g \leftarrow$ function_g(C_j^Γ)
      Sample $(\tilde{X}^k)_{k\in\{1,\ldots,n\}}$ i.i.d of density function g
      Compute $(Y(\tilde{X}^k))_{1\leq k\leq n}$
      $\gamma_j \leftarrow \hat{E}_n^{IS}(\Gamma, j, g)$, $p_j = \hat{P}_n(\Gamma, j, g)$
    end for
    dist = distance(Γ_{old}, Γ), $\Gamma_{old} \leftarrow \Gamma$
  end while
  if $\hat{\epsilon}(\Gamma) \leq$ error then $\Gamma^\star \leftarrow \Gamma$, $P^\star = (p_1, \ldots, p_\ell)$, error $\leftarrow \hat{\epsilon}(\Gamma)$
  end if
end for

Output: $\Gamma^\star$, $P^\star$

Very simple implementation when data is generated before the iterations.

For instance with a unique density g:

Algorithm 3 Find prototypes with a unique density

Input: $(\tilde{X}^k)_{k\in\{1,\dots,n\}}$ i.i.d of density function g, $(Y(\tilde{X}^k))_{1\leq k\leq n}$, $(\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)})_{1\leq k\leq n}$, ℓ, nb_starts, threshold

for start in 1:nb_starts **do**
  Initialize $\Gamma \leftarrow \{\gamma_0, \dots, \gamma_\ell\} \in \mathcal{Y}^\ell$
  $\Gamma_{old} \leftarrow \Gamma$,dist $= +\infty$, error $= +\infty$
  while dist $>$ threshold **do**
    for j in $1:\ell$ **do**
      $\gamma_j \leftarrow \hat{E}_n^{IS}(\Gamma, j,\ g)$, $p_j = \hat{P}_n(\Gamma, j, g)$
    end for
    dist $=$ distance(Γ_{old}, Γ), $\Gamma_{old} \leftarrow \Gamma$
  end while
  if $\hat{\epsilon}(\Gamma) \leq$ error **then** $\Gamma^\star \leftarrow \Gamma$, $P^\star = (p_1, \dots, p_\ell)$, error $\leftarrow \hat{\epsilon}(\Gamma)$
  end if
end for

Output: $\Gamma^\star$, $P^\star$

Constant density for each cell

Or with a constant density g for each cell:

Algorithm 4 Find prototypes with constant density for each cell

Input: $(\tilde{X}_j^k)_{k\in\{1,\dots,n\}}$ i.i.d of density function g_j for $j = 1 : \ell$, $(Y(\tilde{X}_j^k))_{1\leq k\leq n}$, $(\frac{f_X(\tilde{X}_j^k)}{g(\tilde{X}_j^k)})_{1\leq k\leq n}$, nb_starts, threshold

1: **for** start in 1:nb_starts **do**
2: Initialize $\Gamma \leftarrow \{\gamma_0, \dots, \gamma_\ell\} \in \mathcal{Y}^\ell$
3: $\Gamma_{old} \leftarrow \Gamma$,dist $= +\infty$, error $= +\infty$
4: **while** dist $>$ threshold **do**
5: **for** j in $1 : \ell$ **do**
6: $\gamma_j \leftarrow \hat{E}_n^{IS}(\Gamma, j,\ g_j)$, $p_j = \hat{P}_n(\Gamma, j, g_j)$
7: **end for**
8: dist $=$ distance(Γ_{old}, Γ), $\Gamma_{old} \leftarrow \Gamma$
9: **end while**
10: **if** $\hat{\epsilon}(\Gamma) \leq$ error **then** $\Gamma^\star \leftarrow \Gamma$, $P^\star = (p_1, \dots, p_\ell)$, error $\leftarrow \hat{\epsilon}(\Gamma)$
11: **end if**
12: **end for**

Output: $\Gamma^\star$, $P^\star$

Main arguments of find_prototypes

The function to perform this algorithm is *find_prototypes*, with main arguments:

- *data* : the output samples $Y(\tilde{X})$:
  - an array of dim $d_1 \times \cdots \times d_r \times n$ if there is a unique sampling density
  - a list of arrays if they are many sampling densities that do not evolve
  - *NULL* if the sampling evolves
- *density_ratio* : Vector of weights $\frac{f_X}{g}$ or list of vectors, or *NULL*
- *method_IS* : "unique" or "percell", indicating the approach for the sampling. Must be "percell" if $data = NULL$
- *distance_func* The distance between two output elements
- *multistart* : number of starts
- *nb_cells*: number of cells
- *starting_proto* : Optional starting prototypes
- *budget* : Maximum number of iterations for a start
- *threshold* : Threshold distance between Γ and Γ_{old}
- *inputs_function* : The function to build a sample from a Voronoï cell C_j^{Γ}. Required when $data = NULL$
- *outputs_function* : The function returning $Y(X)$; Required when $data = NULL$

Outputs of find_prototypes

- prototypes : the list of optimal prototypes
- probas : a vector indicating the probability mass of the prototypes
- cell_numbers : a vector indicating the cell number associated to each data element
- iterations : an integer indicating the number of iterations performed
- record : a list containing all the centroids computed through the iterations of the best start. Provided only if trace = TRUE.
- all_errors : a vector indicating the quantization error of each start
- all_starts : a list indicating all the best prototypes obtained for each start. Provided only if all_starts = TRUE.

Compute probabilistic weights

compute_density_ratio computes the probabilistic weights $\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$

```
density_ratio = compute_density_ratio(f = fX,
                                      g = g,
                                      inputs = inputs)
```

Section 3

Short example

Short example

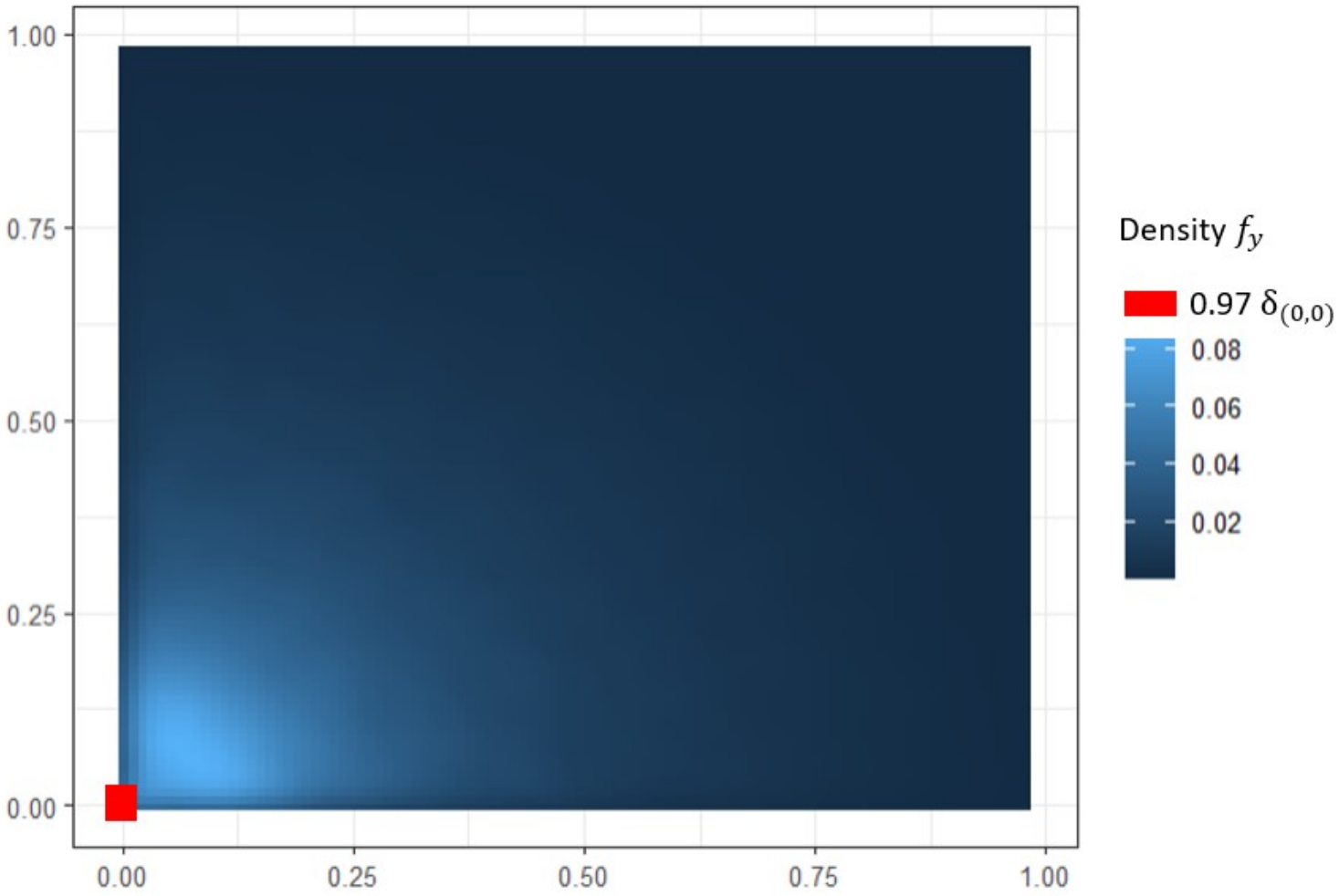
We consider inputs $X = (Rcos(\Theta), Rsin(\Theta)) \in \mathbb{R}^2$ with R and Θ 2 independant random variables defined by the following density functions:

$$\begin{cases} f_R(r) = 0.97 \frac{\mathbb{1}_{[0,0.1]}(r)}{0.1} + 0.01 \times \frac{2}{0.9^2}(1-r)\mathbb{1}_{[0.1,1]}(r) \\ f_\Theta = \frac{2}{\pi}\mathbb{1}_{[0,\frac{\pi}{2}]} \end{cases}$$

And

$$Y(x) = \begin{cases} (0,0) \text{ if } \sqrt{x_1^2 + x_2^2} < 0.1 \\ x \frac{\sqrt{x_1^2+x_2^2}-0.1}{\sqrt{x_1^2+x_2^2}-0.09} \text{ otherwise.} \end{cases}$$

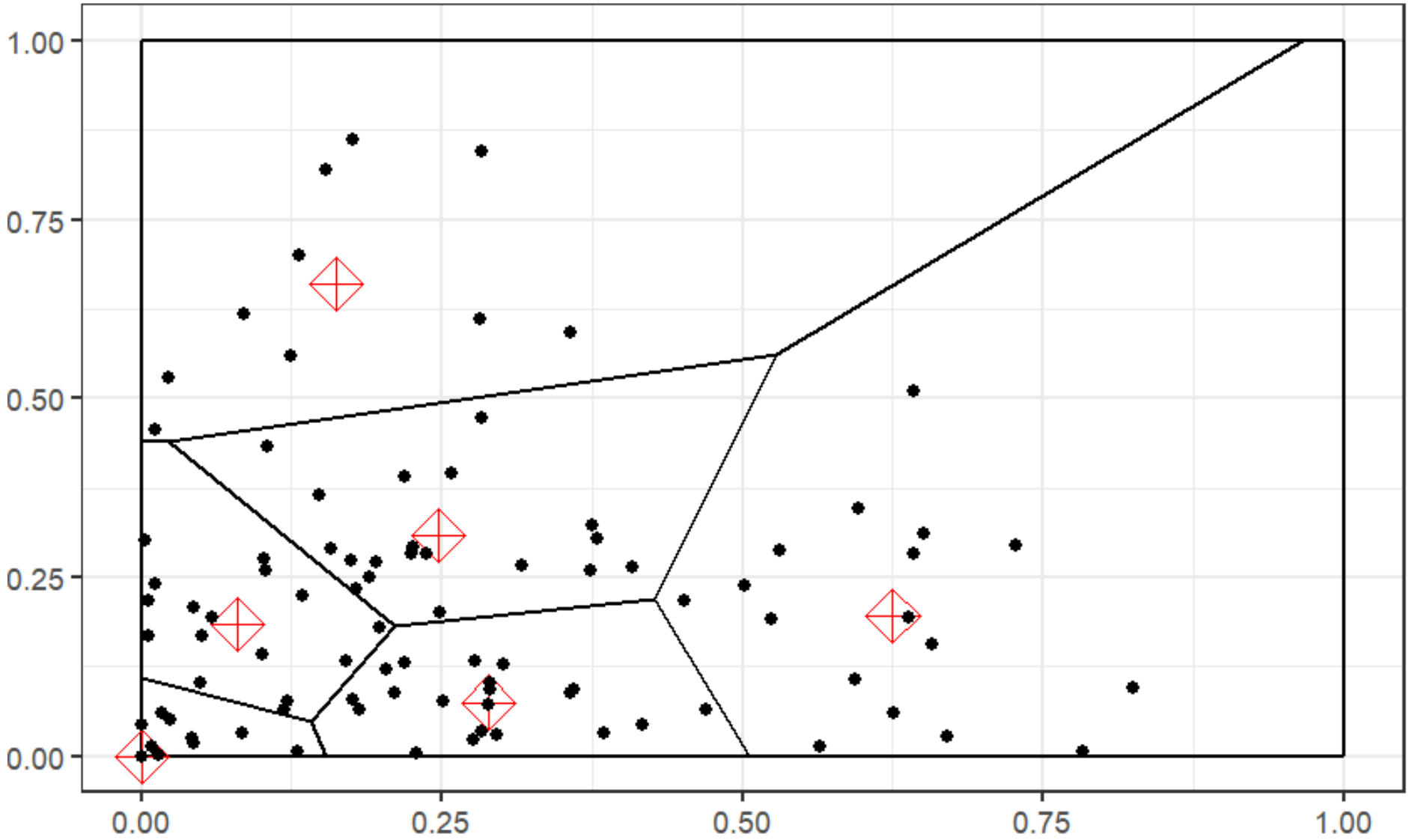
Density of outputs



Classical lloyd

```
inputs_fX = sample_fX(3*10^3)
outputs_fX = apply(inputs_fX, 1, Y)

density_ratio_fX = rep(1,nrow(inputs_fX))

distance_func = function(A1,A2){return(sqrt(sum((A1-A2)^2)))}

res_proto_fX = find_prototypes(nb_cells = 6,
        data = outputs_fX,
        density_ratio = density_ratio_fX,
        distance_func = distance_func,
        multistart = 3)
```

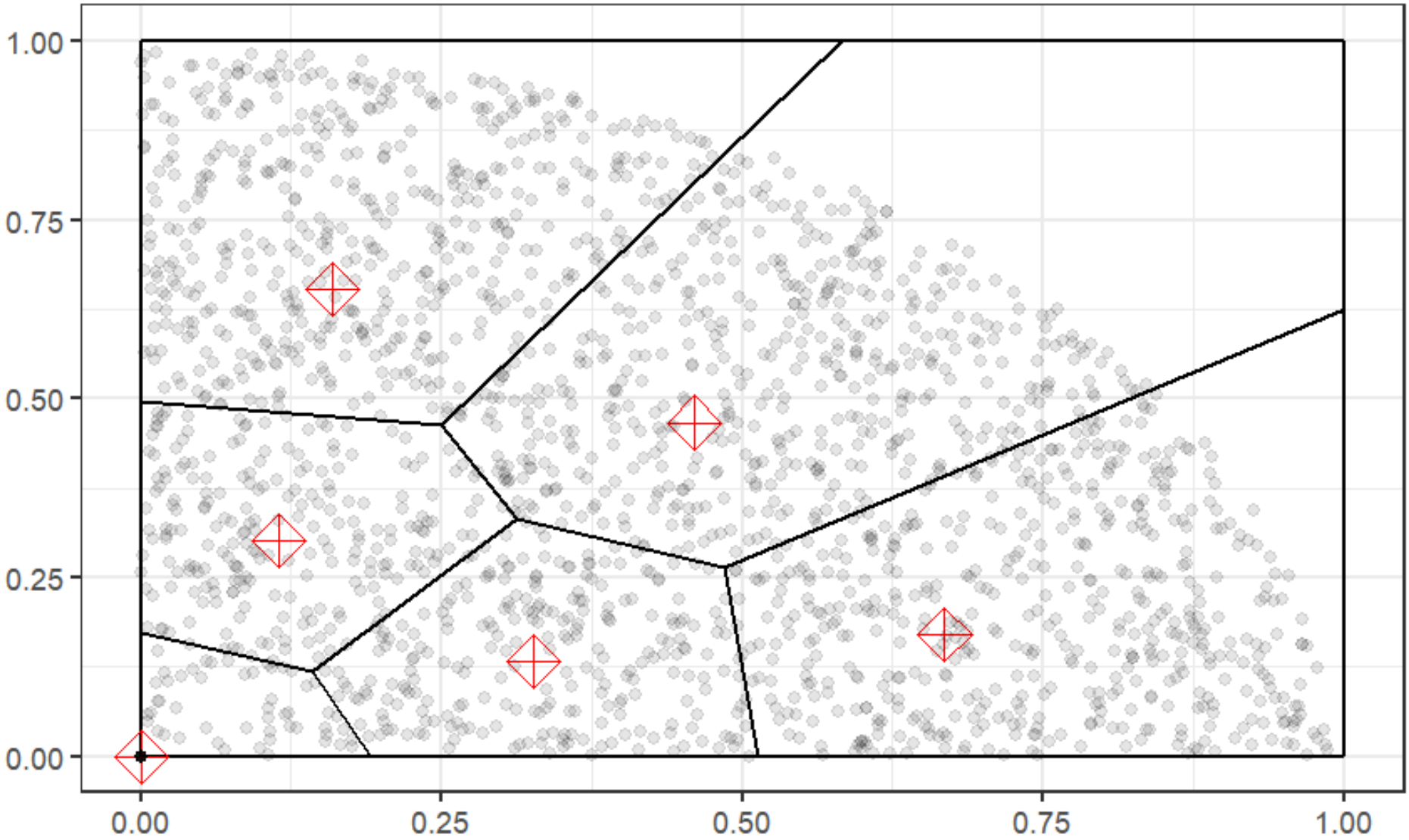
Voronoï cells with Lloyd



Uniform sampling

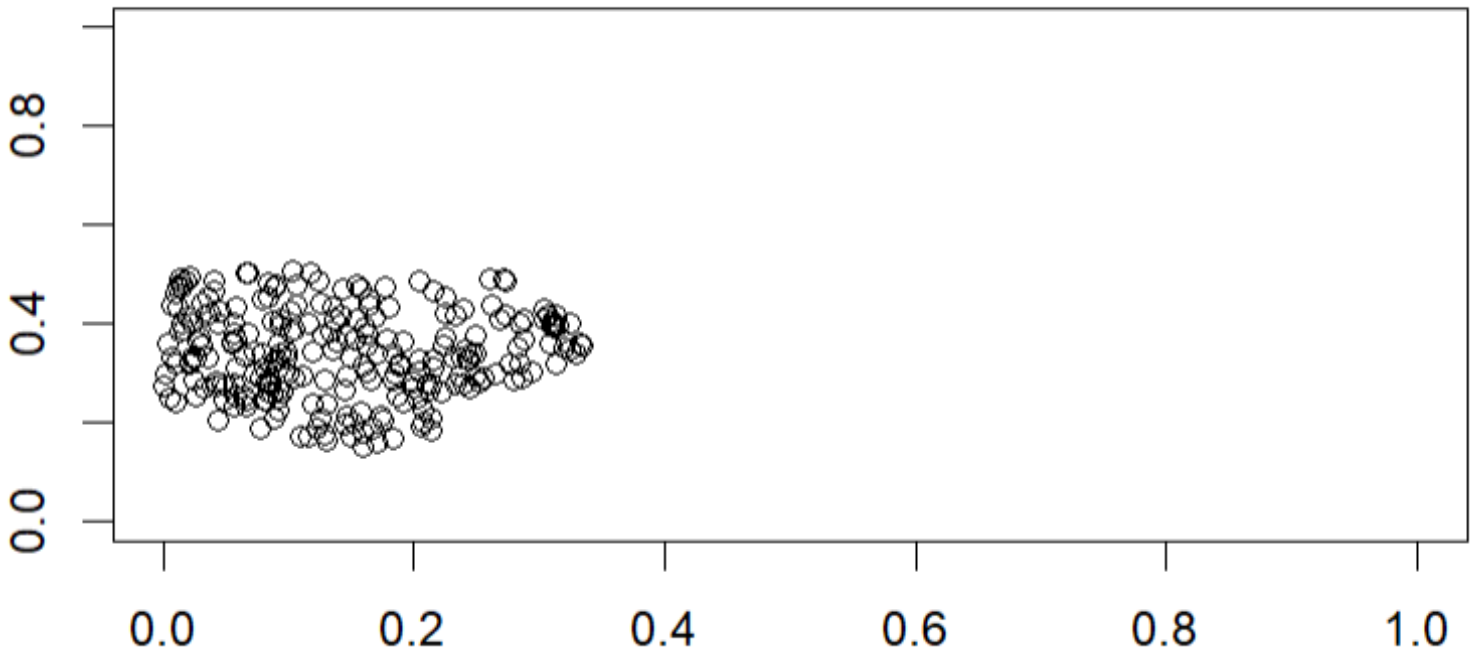
```
r_theta = cbind(sqrt(runif(2000)),runif(2000)*pi/2)
inputs_unif = cbind(r_theta[,1]*cos(r_theta[,2]),r_theta[,1]*sin(r_theta[,2]))
outputs_unif = apply(inputs_unif, 1, Y)
g_unif = function(x){
    r = sqrt(sum(x^2))
    if(r >1){return(0)}
    else{return(4/pi)}
}
density_ratio_unif = compute_density_ratio(f = fX,
    g = g_unif,
    inputs = inputs_unif)
res_proto_unif = find_prototypes(
nb_cells = 6,
data = outputs_unif,
density_ratio = density_ratio_unif,
distance_func = distance_func,
multistart = 3)
```

Voronoï cells with uniform sampling



Evolving sampling

An idea is to create a different density for each cell. Example cell 2:



- Store the limits of x_1 and x_2 in the cell
- Sample with density c times higher in the square than in its complementary in $[-1, 1]^2$ with $c >> 1$

Evolving sampling II

```
g_adapt = function(x,cell,coeff){
  maxs = apply(abs(cell), 2, max)
  mins = apply(abs(cell),2,min)
  A = prod(maxs-mins)
  c1 = 1/(coeff*A+1-A)
  if(sum((maxs-x)>0) == 2 & sum((x-mins)>0) == 2) {return(coeff*c1)}
  else{return(c1)}
}
create_sample = function(n, cell, coeff){
  maxs = apply(abs(cell), 2, max)
  mins = apply(abs(cell),2,min)
  A = prod(maxs-mins)
  prob1 = 1/(A*(coeff-1)+1)
  u = runif(n)
  tirage1 = matrix(runif(sum(u<prob1)*2),ncol=2)
  tirage2 = cbind(runif(sum(u>prob1))*(maxs[1]-mins[1])+mins[1],runi
  return(as.data.frame(rbind(tirage1, tirage2)))
}
```

Evolving sampling III

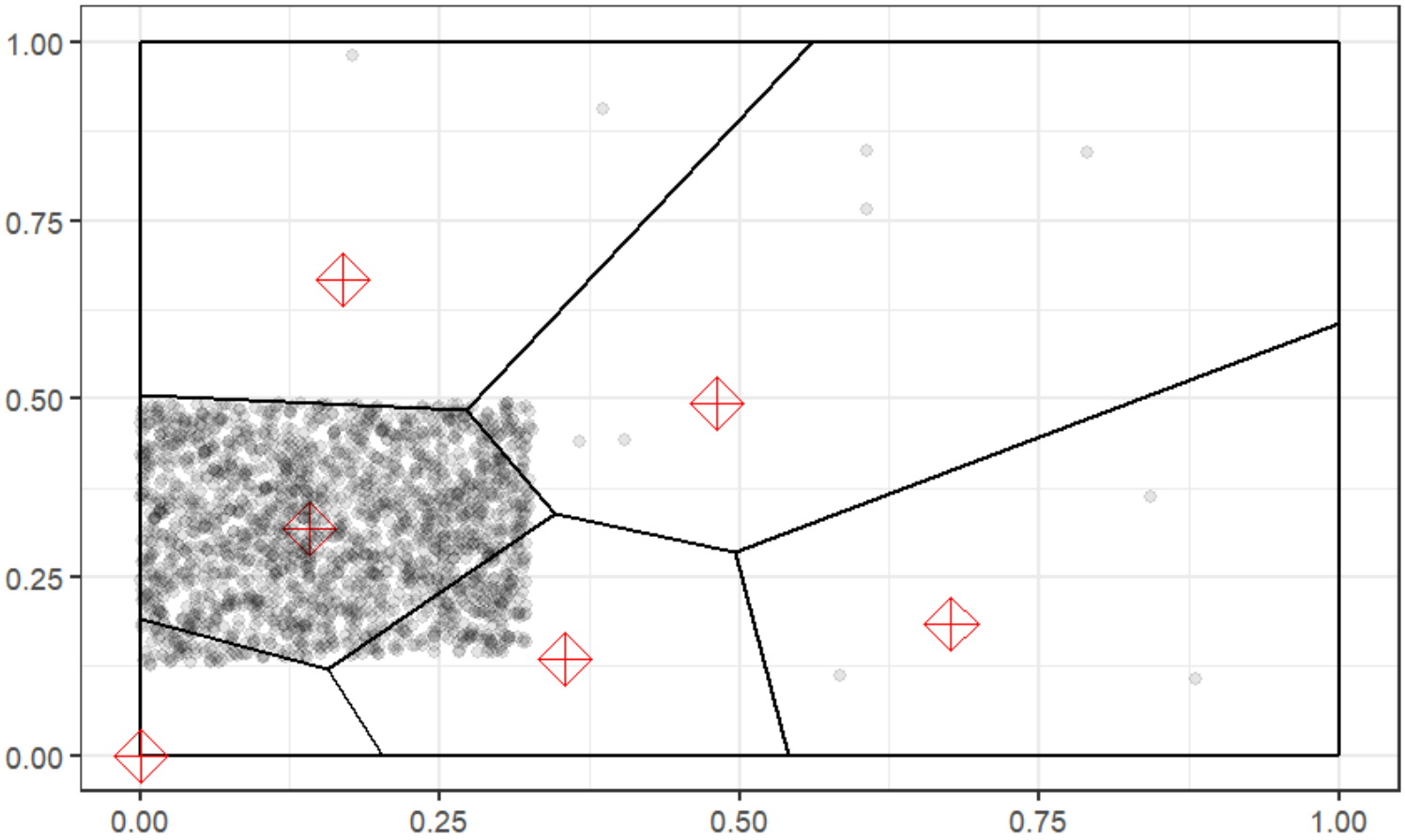
```
density_biased_function = lapply(1:6,
function(i){
function(x, cell){g_adapt(x, cell, 10^3)}})

inputs_function = lapply(1:6,
  function(i){function(cell){
    set.seed(10)
    create_sample(2000,cell, 10^3)}})
```

Evolving sampling IV

```
res_proto_adapt = find_prototypes(
  multistart = 1,
  method_IS = "percell",
  density_ratio = density_adapt,
  inputs_ref = inputs_unif,
  data_ref = outputs_unif,
  density_function = fX,
  density_biased_function = density_biased_function,
  inputs_function = inputs_function,
  outputs_function = function(df){apply(df,1,Y)},
  print_progress = TRUE,
  threshold = 0.01,
  nb_cells = 6)
```

Voronoï cell with evolving sampling

The sampling here is the one for cell 2:



Section 4

Estimation of the variances

Estimators

The objective of the importance sampling is to reduce the variance of the estimators

- $\hat{P}_n(\Gamma, j, g) = \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}_{Y(\tilde{X}^k)\in C_j^\Gamma} \frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$
- $\hat{E}_n^{IS}(\Gamma, j, g) = \frac{\frac{1}{n}\sum_{k=1}^{n} Y(\tilde{X}^k)\mathbb{1}_{Y(\tilde{X}^k)\in C_j^\Gamma} \frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}}{\hat{P}_n(\Gamma,j,g)}$

And we have

- $\mathbb{V}(\frac{1}{n}\sum_{k=1}^{n} h(Y(\tilde{X}^k))\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}) = \frac{1}{n}\mathbb{V}(h(Y(\tilde{X}))\frac{f_X(\tilde{X})}{g(\tilde{X})})$
- $\mathbb{V}\left(\frac{A}{B}\right) \approx \frac{\mu_A^2}{\mu_B^2}\left[\frac{\mathbb{V}(A)}{\mu_A^2} - 2\frac{cov(A,B)}{\mu_A\mu_B} + \frac{\mathbb{V}(B)}{\mu_B^2}\right]$.

with μ_A, μ_B the mean of A and B.

Estimation with bootstrap

If a sample $(\tilde{X}^k)_{k=1}^n$ is provided, then bootstrap approach is recommended

The estimators can be seen as $\hat{\theta}_n = T(\tilde{X}^1, \ldots, \tilde{X}^n)$. Then,

- Sample with replacement leading to a bootstrap sample $\tilde{X}_{(1)}^{\star(1)}, \ldots, \tilde{X}_{(1)}^{\star(n)}$. Compute $\hat{\theta}_{n,1} = T(\tilde{X}_{(1)}^{\star(1)}, \ldots, \tilde{X}_{(1)}^{\star(n)})$
- Repeat the previous step B times, providing $\hat{\theta}_{n,1}, \ldots, \hat{\theta}_{n,B}$
- Compute $\hat{s} = \sqrt{\frac{1}{B}\sum_{j=1}^{B}(\hat{\theta}_{n,j} - \bar{\theta})^2}$ with $\bar{\theta} = \frac{1}{B}\sum_{j=1}^{B}\hat{\theta}_{n,j}$

Estimation without bootstrap

If a sample $(\tilde{X}^k)_{k=1}^{\tilde{n}}$ is provided with $\tilde{n} >> n$, then

$$\hat{\mathbb{V}}\left(h(Y(\tilde{X}))\frac{f_X(\tilde{X})}{g(\tilde{X})}\right) = \frac{1}{\tilde{n}-1}\sum_{k=1}^{\tilde{n}}\left(h(Y(\tilde{X}^k))\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)} - \bar{h}\right)^2$$

with $\bar{h} = \frac{1}{\tilde{n}}\sum_{k=1}^{\tilde{n}} h(Y(\tilde{X}^k))\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$

And then the variance $\mathbb{V}(\frac{1}{n}\sum_{k=1}^{n} h(Y(\tilde{X}^k))\frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)})$ is estimated by $\hat{\mathbb{V}}\left(h(Y(\tilde{X}))\frac{f_X(\tilde{X})}{g(\tilde{X})}\right)$ this variance by n.

std_proba

```
std_proba_fX = std_proba(
data = outputs_fX,
prototypes_list = list(res_proto_fX$prototypes),
cells = 1:6,
density_ratio = density_ratio_fX,
bootstrap = 3000)
```

Comparison of the std_proba

std_proba_fX

```
#> [[1]]
[1] 0.003048712 0.249471763 0.232306413 0.226953128 0.312199864
0.246070898
```

std_proba_unif

```
#> [[1]]
[1] 0.20723311 0.04856455 0.05941715 0.05071932 0.06369044 0.04815171
```

std_proba_adapt

```
#> [[1]]
[1] 0.04713507 0.01216419 0.02047372 0.01248061 0.02060620 0.02508885
```

std_centroid

```
std_centroid_fX = std_centroid(
data = outputs_fX,
prototypes_list = list(res_proto_fX$prototypes),
cells = 1:6,
density_ratio = density_ratio_fX,
bootstrap = 1000)
```

Output of the std_centroid

std_centroid_fX

```
## [[1]]
## [[1]][[1]]
## [1] 5.868107e-05 3.603997e-05
##
## [[1]][[2]]
## [1] 0.01482003 0.01726276
##
## [[1]][[3]]
## [1] 0.02049966 0.01710263
##
## [[1]][[4]]
## [1] 0.017598540 0.008361999
##
## [[1]][[5]]
## [1] 0.03671068 0.04440973
##
## [[1]][[6]]
## [1] 0.02299448 0.03268645
```

Section 5

Estimation of the quantization error

Two approaches

$$e(\Gamma)^2 = \mathbb{E}(\|Y(X) - q_\Gamma(Y(X))\|^2)$$
$$= \sum_{j=1}^{\ell} \mathbb{E}(\|Y(X) - q_\Gamma(Y(X))\|^2 | \, q_\Gamma(Y(X)) = \gamma_j)\mathbb{P}(q_\Gamma(Y(X)) = \gamma_j)$$

Two approaches are proposed by FunQuant:

- Unique sampling density:
  $\hat{e}(\Gamma)^2 = \frac{1}{n}\sum_{k=1}^{n} \|Y(\tilde{X}^k) - q_\Gamma(Y(\tilde{X}^k))\|_{\mathcal{Y}}^2 \frac{f_X(\tilde{X}^k)}{g(\tilde{X}^k)}$
- Multiple sampling density:
  $\hat{e}(\Gamma) = \sum_{j=1}^{\ell} \frac{1}{n_j}\sum_{k=1}^{n_j} \|Y(\tilde{X}^k) - q_\Gamma(Y(\tilde{X}_j^k))\|_{\mathcal{Y}}^2 \mathbb{1}_{Y(\tilde{X}_j^k)\in C_j^\Gamma} \frac{f_X(\tilde{X}_j^k)}{g_j(\tilde{X}_j^k)}$

with $(\tilde{X}_j^k)_{k=1}^{n_j}$ i.i.d. of density g_j.

quanti_error

```
quanti_error(
prototypes = res_proto_fX$prototypes,
data = outputs_fX,
density_ratio = density_ratio_fX,
method_IS = "unique")

quanti_error(
prototypes = res_proto_adapt$prototypes,
data = outputs_adapt,
density_ratio = density_ratio_adapt),
method_IS = "percell"
```