Zoning simulated data

B. Charnomordic

2018-02-06

Contents

```
library(geozoning)
library(sp)
library(fields)
```

This vignette illustrates the zoning with corrections procedure on simulated data.

Generate a map

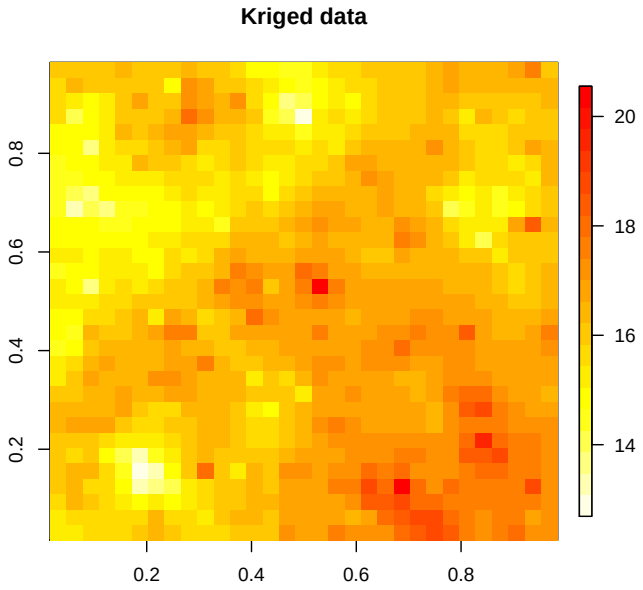
A map object is simulated with an exponential field and a variogram model. 450 points (default) are randomly allocated on a square field of size 1. Then 1936 points are kriged on a regular grid using inverse distance weighted interpolation. A Delaunay tesselation yields point neighborhood in the sense of Voronoi. For this purpose, we use the *genMap* function which is a wrapper of the *randKmap* function (see the documentation).

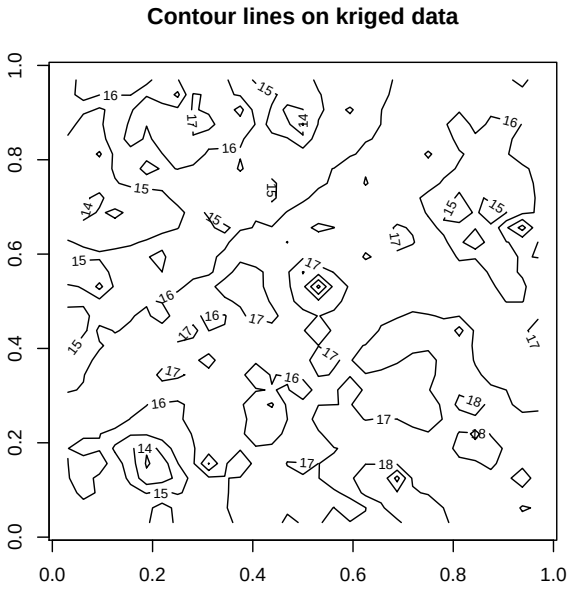
```
seed=80
map=genMap(DataObj=NULL,seed=seed,disp=FALSE,krig=2,Vmean=15,typeMod="Exp")
```

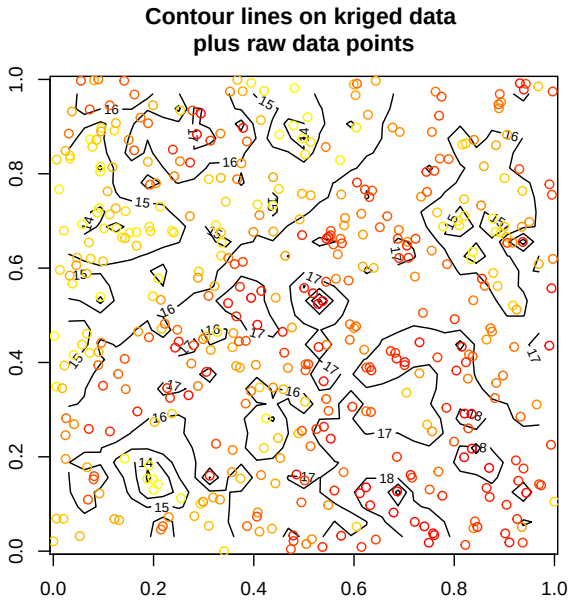
```
## [1] "DataObj=NULL, generating DataObj-seed= 80"
## [inverse distance weighted interpolation]
```

Display 2D map with three different views: first one=kriged data, second one=contour lines, third one=raw data.

```
plotMap(map)
```







The map object contains all components required for zoning generation, evaluation and visualization. The data-related components include the kriged data, grid resolution, size in each dimension and map boundary. Kriged data are available as a **SpatialPointsDataFrame** object, as well as a matrix object directly usable by image functions. rawData are stored as well in the map object, for traceability purposes. The neighborhood-related components include the list of neighbor point indices for each kriged data point, as well as the areas of Voronoi polygons associated to all points. Finally the variogram-related components include the VGM model used to simulate the field, as well as the equivalent RandomFields model.

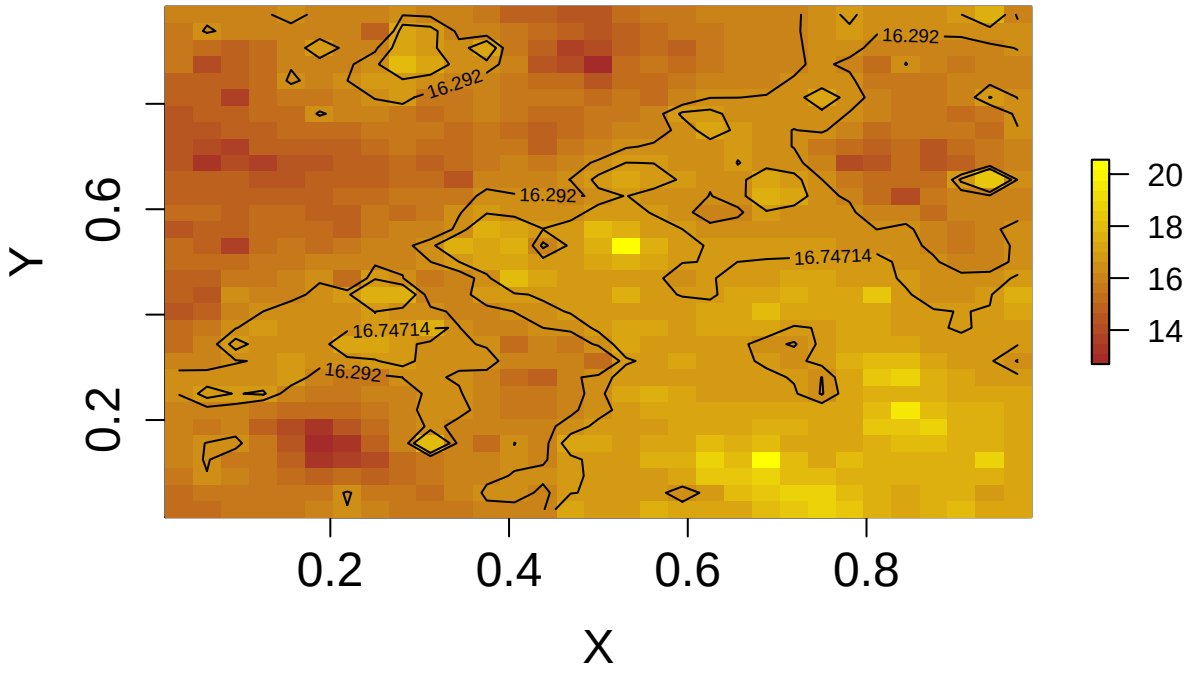
```
# Check the mean and standard deviation of generated data
meanvarSimu(map)
```

```
##    raw mean kriged mean      raw sd   kriged sd
##  16.2167192  16.2537869   1.6219676   0.9622766
```

Generate zoning from map for a given probability vector

Given a probability vector, a vector of values is obtained using the *quantile* function. Display map image with contour levels corresponding to *qq* values

```
qq=quantile(map$krigGrid,na.rm=TRUE,prob=c(0.5,0.7))
dispZ(map$step,map$krigGrid,valQ=qq)
```



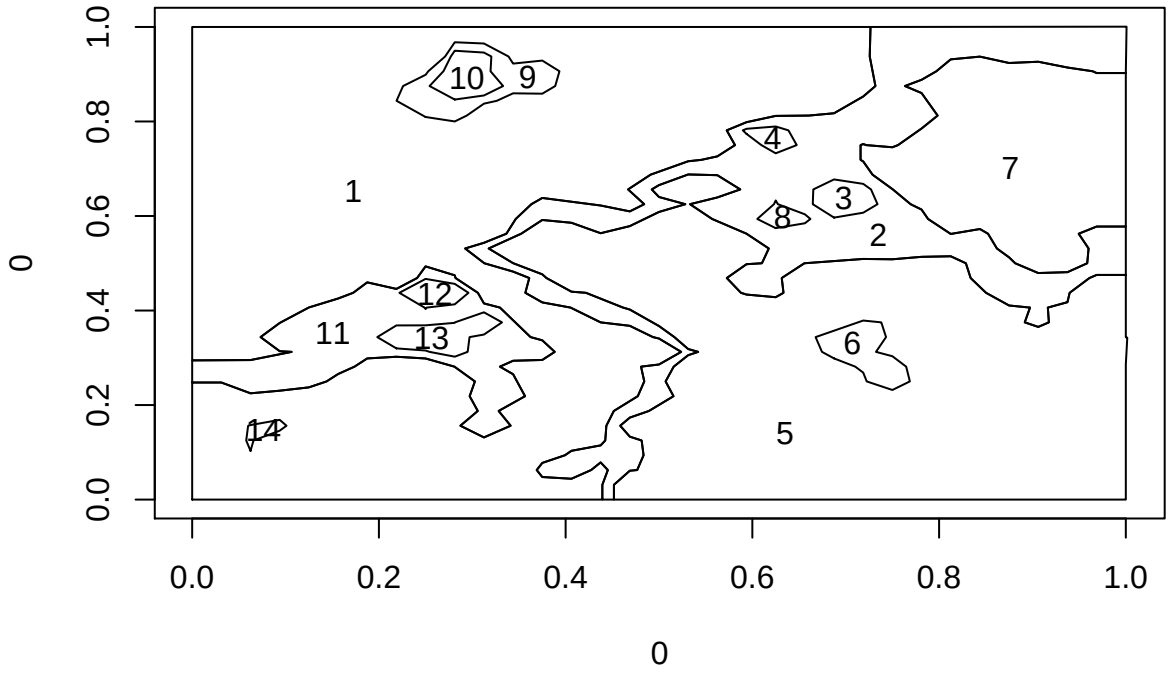

```
## NULL
```

A zoning is done on the kriged data, by computing the contour lines corresponding to the vector of values given by the probability vector, trimming them to the map boundary and defining zones corresponding to the closed contour lines. A zoning is a list of **SpatialPolygons** objects. It does not contain data, only polygon geometry.

```
# names(ZK):  "resCrit"  "resDist" "resZ" "cL" "qProb"
ZK=initialZoning(qProb=c(0.5,0.7),map=map)
```

Plot zoning (14 zones in this case)

```
K=ZK$resZ
Z=K$zonePolygone
plotZ(Z)
```



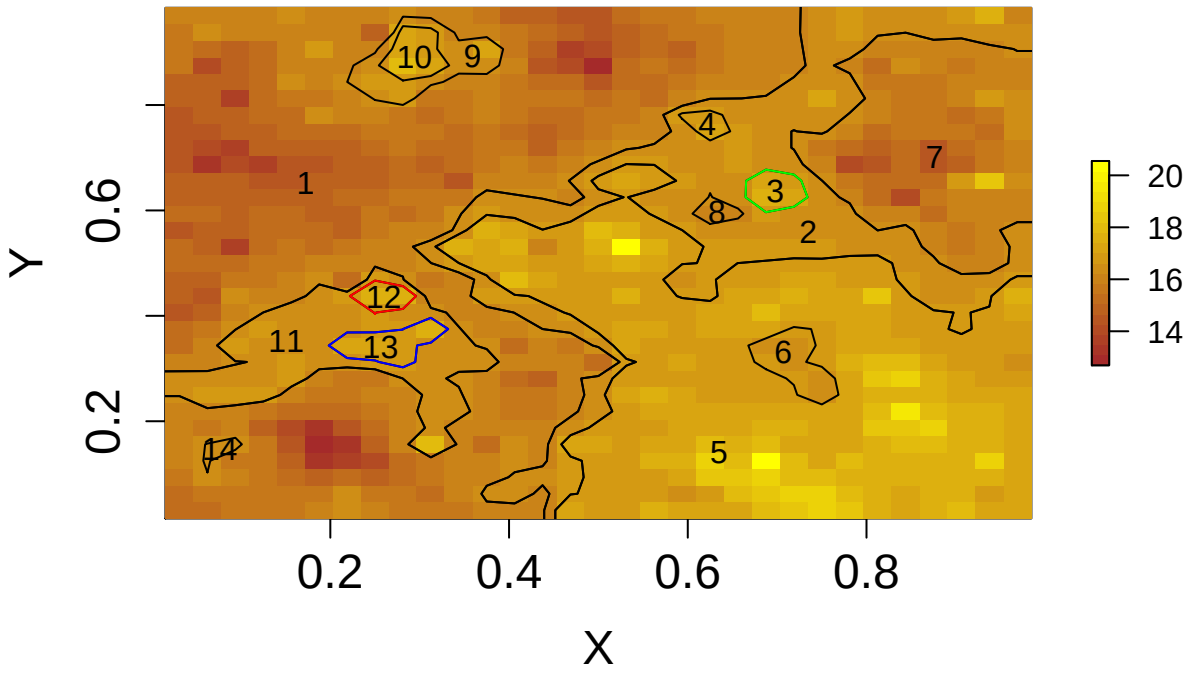

```
## NULL
```

Or a more detailed plot with data underneath - compare it to previous plot of contour lines. We see that contour lines are now extended to the map boundary in order to close zones and that contour lines impossible to close or zones with 0 or 1 point are removed.

```
# Outline boundaries of zone 12 with different colors
dispZ(map$step,map$krigGrid,zonePolygone=Z,iZ=12)
```

```
## NULL
```

```
# Outline all polygons of zone 13 with a different color
linesSp(Z[[13]],col="blue") # first one in blue
linesSp(Z[[2]],k=2,col="green") # second one in red, and so on
```

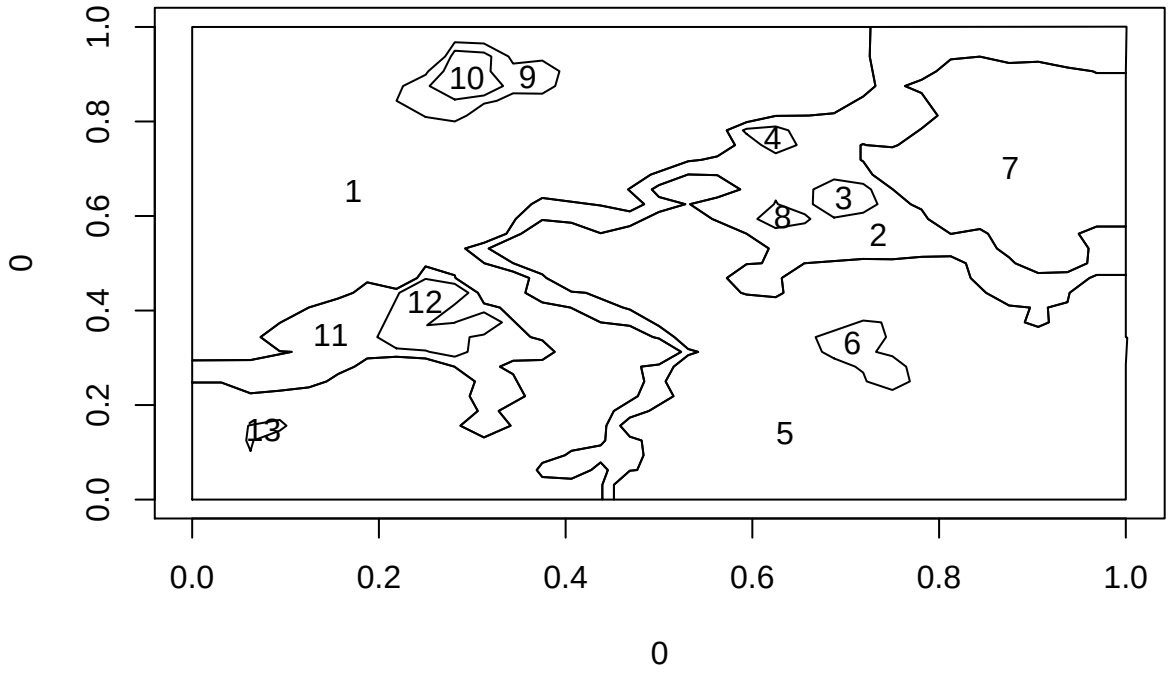



```
# A zone can have one or several holes
# and each hole is an independent zone. zone 2 has 3 holes (zones 3,8,4).
# Due to its shape and to the common borders with the map boundary,
# zone 7 is not a hole in zone 2.
holeSp(Z[[2]])
```

```
## [1] 3
```

Junction of 2 zones: Join zone 12 with another zone near by (zone 13). Both zones have the same label

```
kmi=optiRG(K,map,12,13,disp=1)
plotZ(kmi$zonePolygone)
```



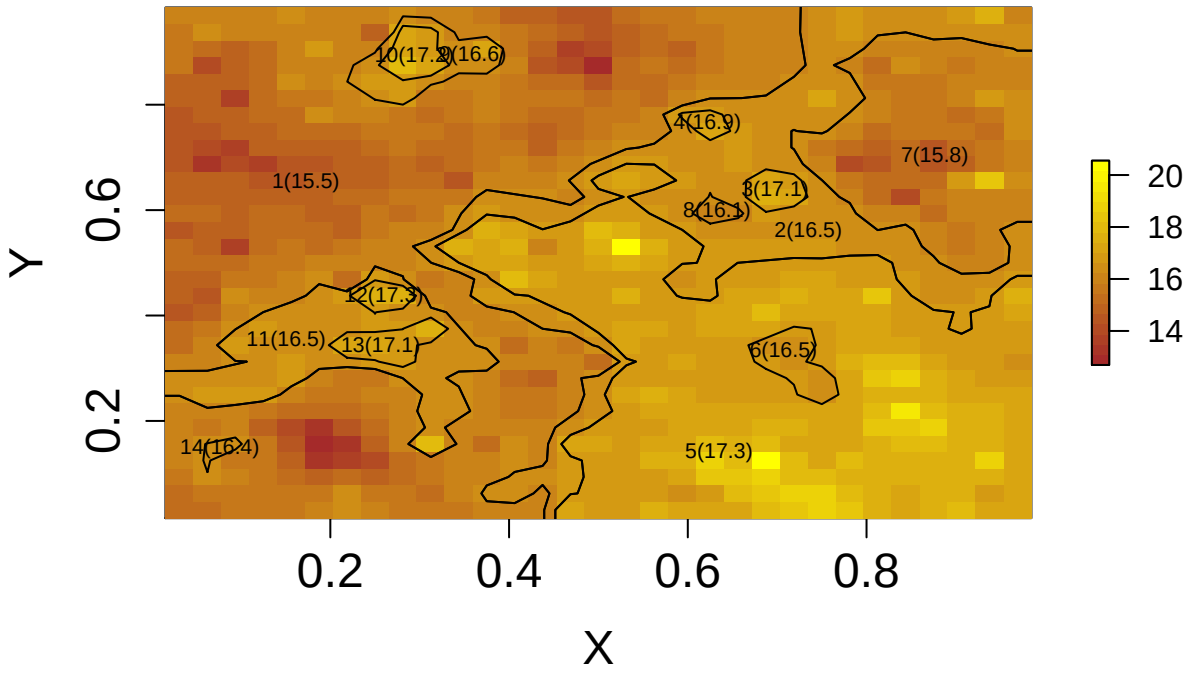

```
## NULL
```

A more detailed plot: each zone is labelled with its number and its mean value

```
dispZ(map$step,map$krigGrid,zonePolygone=Z,K=K,boundary=map$boundary,nbLvl=0,id=FALSE,mu=2,cex=0.7)
```

```
## NULL
```

```
# add quantile values and criterion value for Z
title(paste(" q=[",toString(round(qq,2)),"]  crit=",round(ZK$resCrit,2),sep=""))
```




```
  # print zone labels
  printLabZ(list(K))
```

```
## [1] "2q zone labels= c(1, 2, 3, 3, 3, 2, 1, 1, 2, 3, 2, 3, 3, 2)"

## [[1]]
##  [1] 1 2 3 3 3 2 1 1 2 3 2 3 3 2
```

```
  # print zone areas
  printZsurf(K$zonePolygone)
```

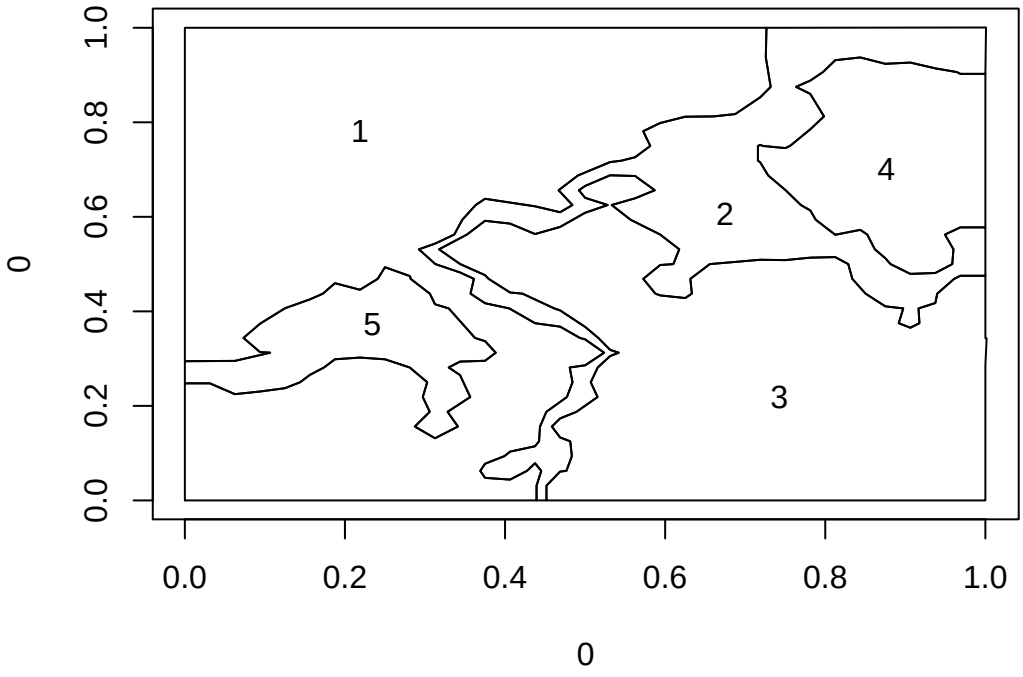
```
## [1] "iZ= 1 area= 0.42236"
## [1] "iZ= 2 area= 0.13187"
## [1] "iZ=3 area=0.00403 < minSize(0.012)"
## [1] "iZ=4 area=0.00194 < minSize(0.012)"
## [1] "iZ= 5 area= 0.26953"
## [1] "iZ=6 area=0.00756 < minSize(0.012)"
## [1] "iZ= 7 area= 0.08913"
## [1] "iZ=8 area=0.00172 < minSize(0.012)"
## [1] "iZ=9 area=0.01112 < minSize(0.012)"
## [1] "iZ=10 area=0.00521 < minSize(0.012)"
## [1] "iZ= 11 area= 0.04538"
## [1] "iZ=12 area=0.00282 < minSize(0.012)"
## [1] "iZ=13 area=0.00631 < minSize(0.012)"
## [1] "iZ=14 area=0.00119 < minSize(0.012)"

## [1] 14  8  4 12  3 10 13  6  9
```

```
  # print zone ids
  printZid(Z)
```

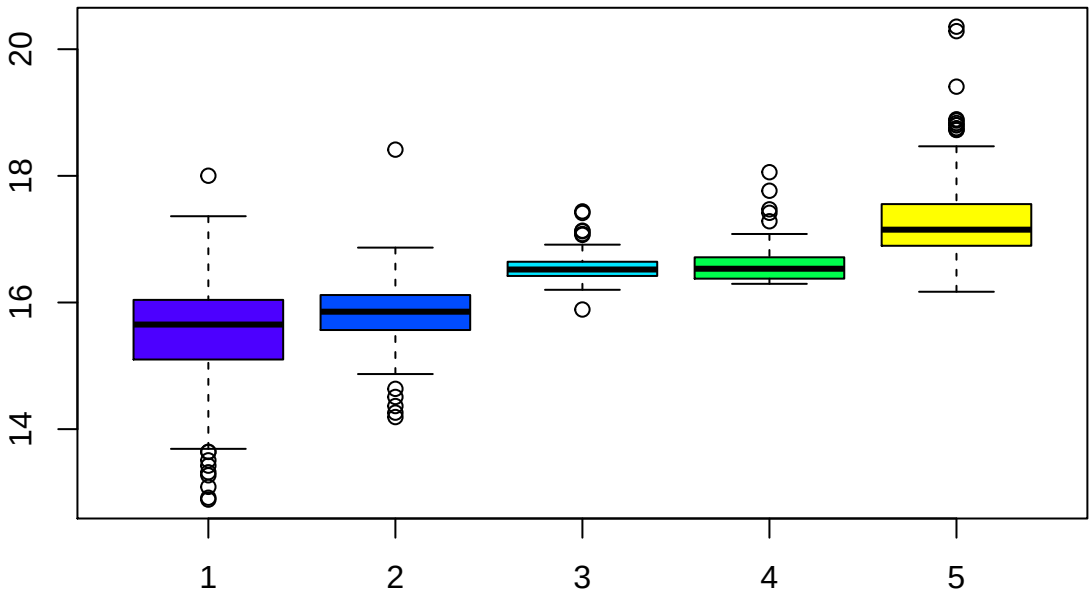
```
## [1] "ii= 1  ID= 1"
## [1] "ii= 2  ID= 2"
## [1] "ii= 3  ID= 3"
## [1] "ii= 4  ID= 4"
## [1] "ii= 5  ID= 5"
## [1] "ii= 6  ID= 6"
## [1] "ii= 7  ID= 7"
## [1] "ii= 8  ID= 8"
## [1] "ii= 9  ID= 9"
## [1] "ii= 10  ID= 10"
## [1] "ii= 11  ID= 11"
## [1] "ii= 12  ID= 12"
## [1] "ii= 13  ID= 13"
## [1] "ii= 14  ID= 14"
```

```
  # remove zones with less than 10 data points
  K=calNei(Z,map$krigData,map$krigSurfVoronoi,map$krigN,nmin=10)
  plotZ(K$zonePolygone)
```




```
## NULL
```

```
  val=valZ(map,K)$val
  boxplot(val,col=topo.colors(length(val)))
```



Generate tree of possible corrections for small zones

2 operations are done for each small zone :

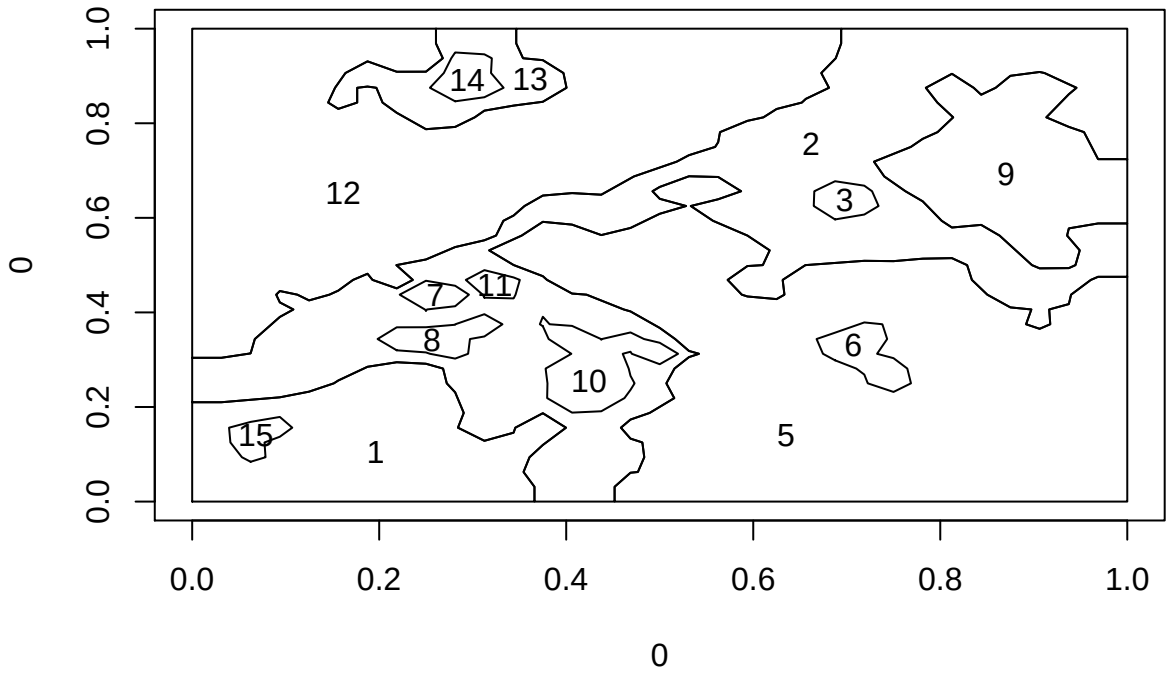
- 1- remove zone, i.e. merge into englobing zone
- 2- grow zone

Growing is done in 2 different ways depending on zone proximity to other ones. If zone is isolated (distance to other zones controlled by distIsoZ parameter), it grows bigger but remains isolated from others. Zone growing in that case is performed by finding the contour line close to the current zone contour, that maximizes the zoning quality criterion. A small value of distIsoZ ensures that a small zone have enough space to grow. If zone is non isolated, it is joined to the closest zone with the same label.

```
#save all branches resulting from correction steps
criti<-correctionTree(c(0.4,0.7),map,SAVE=TRUE,ALL=TRUE,LASTPASS=FALSE,distIsoZ=0.01)
zk=criti$zk
```

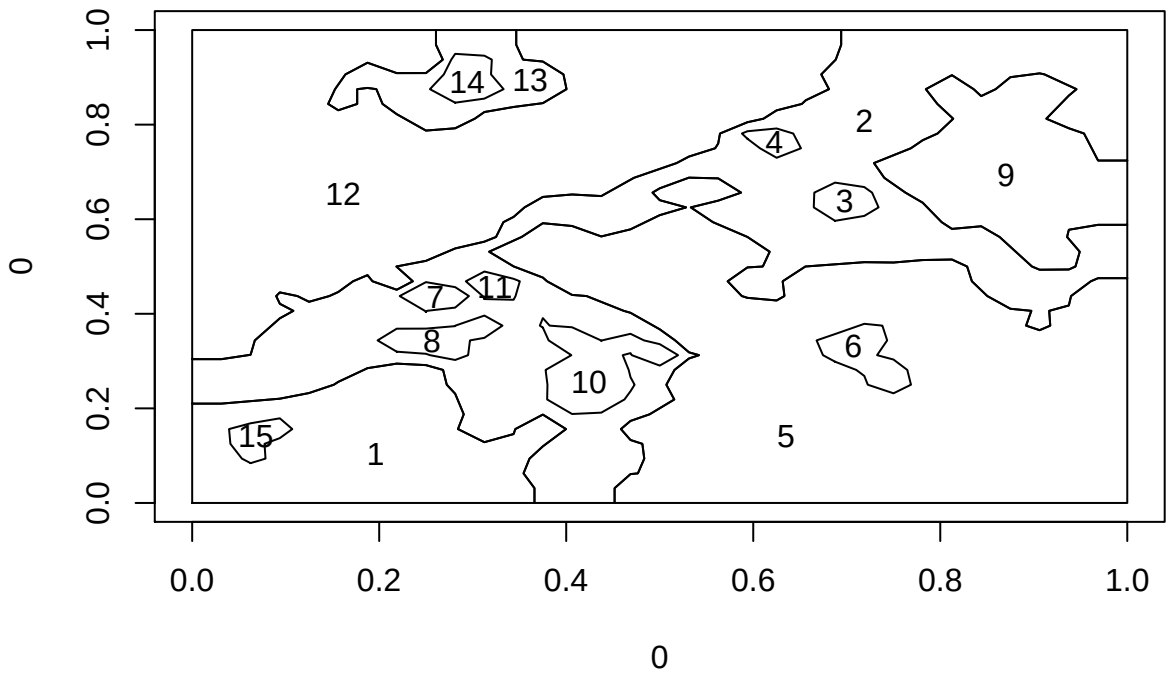
In that case we have for example #4 small zone, hence 3 levels (level 1 is initial zoning, level 2 has 2 branches, level 3 has 4 branches). For each correction step-first branch=zone removal, second-branch=zone junction. The procedure starts with the smallest zone, here zone #4.

```
Z21=zk[[2]][[1]]$zonePolygone
Z22=zk[[2]][[2]]$zonePolygone
plotZ(Z21,id=TRUE) # result of removal of zone #4
```




```
## NULL
```

```
plotZ(Z22,id=TRUE) # result of growing of zone #4
```




```
## NULL
```

```
# successively:  removal of zone#4  in Z21, growing of zone#4  in Z21
# removal of zone#4  in Z22, growing of zone#4  in Z22
# other try with LASTPASS=TRUE removes at last step the zones that are still too small
```

```r
# after all successive corrections
criti<-correctionTree(c(0.4,0.7),map,SAVE=TRUE,ALL=TRUE,LASTPASS=TRUE,distIsoZ=0.001)

# other run with ALL=FALSE saves memory by keeping only the first and the last levels
criti<-correctionTree(c(0.4,0.7),map,SAVE=TRUE,ALL=FALSE,LASTPASS=FALSE,distIsoZ=0.001)
```

Session informations

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.5 LTS
##
## locale:
##  [1] LC_CTYPE=fr_FR.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=fr_FR.UTF-8        LC_COLLATE=fr_FR.UTF-8
##  [5] LC_MONETARY=fr_FR.UTF-8    LC_MESSAGES=fr_FR.UTF-8
##  [7] LC_PAPER=fr_FR.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=fr_FR.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] ggplot2_2.2.1            rgeos_0.3-26
##  [3] raster_2.6-7             fields_9.0
##  [5] maps_3.2.0               spam_2.1-1
##  [7] dotCall64_0.9-04         deldir_0.1-14
##  [9] maptools_0.9-2           RandomFields_3.1.50
## [11] RandomFieldsUtils_0.3.25 sp_1.2-5
## [13] gstat_1.1-5              geozoning_1.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.14     pillar_1.0.1     plyr_1.8.4       xts_0.10-0
##  [5] tools_3.3.2      digest_0.6.12    evaluate_0.10.1  memoise_1.1.0
##  [9] tibble_1.4.1     gtable_0.2.0     lattice_0.20-34  rlang_0.1.6
## [13] rstudioapi_0.7   commonmark_1.4   yaml_2.1.14      knitr_1.17
## [17] withr_2.1.0      stringr_1.2.0    roxygen2_6.0.1   xml2_1.1.1
## [21] devtools_1.13.4  desc_1.1.1       rprojroot_1.2    spacetime_1.2-1
## [25] R6_2.2.2         rmarkdown_1.8    foreign_0.8-67   magrittr_1.5
## [29] htmltools_0.3.6  backports_1.1.1  scales_0.5.0     intervals_0.15.1
## [33] assertthat_0.2.0 colorspace_1.3-2 stringi_1.1.6    lazyeval_0.2.1
## [37] munsell_0.4.3    crayon_1.3.4     FNN_1.1          zoo_1.8-0
```