Comparison between different local tests: Simes, Simes with Storey and Wilcoxon-Mann-Whitney using the natural outliers distribution

2023-08-04

The aim is to compare on real datasets the performance of three closed testing procedures, which respectively use Simes local test with and without Storey estimator for the proportion of true null hypotheses and Wilcoxon-Mann-Whitney local test. We will consider outlier population to be the set of observations tagged as "outlier" in the dataset of interest.

R functions and libraries

```
library(nout)
library(R.matlab)
library(isotree)
library(readr)
library(foreign)
library(tictoc)
library(tidyverse)
library(doSNOW)
library(ggplot2)

compact_results = function(res){
  resT=as.data.frame(t(res))

  results = list()
  for(j in 1:length(n1s)){
    lb.d = as.data.frame(
      cbind("d_BH"=unlist(res[[j]][rownames(res[[j]])=="d_BH",]),
            "d_StoBH"=unlist(res[[j]][rownames(res[[j]])=="d_StoBH",]),
            "d_Sim"=unlist(res[[j]][rownames(res[[j]])=="d_Sim",]),
            "d_StoSimes"=unlist(res[[j]][rownames(res[[j]])=="d_StoSimes",]),
            "d_WMW"=unlist(res[[j]][rownames(res[[j]])=="d_WMW",])
            )
    )
    mean.lb.d = apply(lb.d, MARGIN = 2, FUN = mean)

    power.GlobalNull = as.data.frame(lb.d>0)
    mean.powerGlobalNull = apply(power.GlobalNull, MARGIN = 2, FUN = mean)

    # n.disc = as.data.frame(
    #   cbind("n.disc.Simes" = unlist(res[[j]][rownames(res[[j]])=="n.disc.Simes",]),
    #         "n.disc.Simes2" = unlist(res[[j]][rownames(res[[j]])=="n.disc.Simes2",]),
    #          "n.disc.StoSimes" = unlist(res[[j]][rownames(res[[j]])=="n.disc.StoSimes",]),
    #         "n.disc.WMW" = unlist(res[[j]][rownames(res[[j]])=="n.disc.WMW",]),
    #         "n.disc.WMW.cpp" = unlist(res[[j]][rownames(res[[j]])=="n.disc.WMW.cpp",])
    #         )
    # )
```

```
    # mean.n.disc = apply(n.disc, MARGIN = 2, FUN = mean)
    # #mean.n.disc_pos = apply(n.disc>0, MARGIN = 2, FUN = mean)

    results[[j]] = list("lb.d" = lb.d,
                        "mean.lb.d" = mean.lb.d,
                        "power.GlobalNull" = power.GlobalNull,
                        "mean.powerGlobalNull" = mean.powerGlobalNull,
                        # "n.disc" = n.disc,
                        # "mean.n.disc" = mean.n.disc,
                        # #"mean.n.disc>0" = mean.n.disc_pos,
                        "pi.not" = res[[j]][rownames(res[[j]])=="pi.not",],
                        "uniques" = res[[j]][rownames(res[[j]])=="uniques",],
                        "n1" = res[[j]][rownames(res[[j]])=="n1",1],
                        "alpha" = res[[j]][rownames(res[[j]])=="alpha",1])
  }
  return(results)
}


TrainingIsoForest = function(l, dataset){

  tr_ind = sample(in_ind, size = l)
  tr = dataset[tr_ind,]
  isofo.model = isotree::isolation.forest(tr, ndim=ncol(dataset), ntrees=10, nthreads=1,
                            scoring_metric = "depth", output_score = TRUE)$model
  in_index2 = setdiff(in_ind, tr_ind)

  return(list("model"=isofo.model, "inlier_remaining" = in_index2))

}



CompareMethodNaturalOutliers = function(B, n1, n, out_ind, inlier_remaining, isofo.model, dataset){

  n0 = n-n1
  foreach(b = 1:B, .combine=cbind) %dopar% {
    if(n1==0){
      N = n0 + m
      in_index3 = sample(inlier_remaining, size = N)
      cal_ind = in_index3[1:m]
      te_ind = in_index3[(m+1):N]
      cal = dataset[cal_ind,]
      te = dataset[te_ind,]
      S_cal = predict.isolation_forest(isofo.model, cal, type = "score")
      S_te = predict.isolation_forest(isofo.model, te, type = "score")

      d_WMW = nout::d_MannWhitney(S_Y = S_te, S_X = S_cal, alpha=alpha)
      d_Sim = nout::d_Simes(S_X = S_cal, S_Y = S_te, alpha = alpha)
      StoSimes = nout::d_StoreySimes(S_X = S_cal, S_Y = S_te, alpha = alpha)
      d_StoSimes = StoSimes$d
      pi.not = StoSimes$pi.not
      d_BH = nout::d_benjhoch(S_X = S_cal, S_Y = S_te, alpha = alpha)
```

```
  d_StoBH = nout::d_StoreyBH(S_X = S_cal, S_Y = S_te, alpha = alpha)
  uniques = length(unique(c(S_cal, S_te)))
  return(list("d_BH" = d_BH,
              "d_StoBH" = d_StoBH,
              "d_Sim" = d_Sim,
              # "n.disc.Simes" = 0,
              # "n.disc.Simes2" = 0,
              "d_StoSimes" = d_StoSimes,
              # "n.disc.StoSimes" = 0,
              "d_WMW" = d_WMW,
              # "n.disc.WMW" = 0,
              # "n.disc.WMW.cpp" = 0,
              "uniques" = uniques,
              "n1" = n1,
              "pi.not" = pi.not,
              "alpha" = alpha))
}

else{
  N = n0 + m
  in_index3 = sample(inlier_remaining, size = N)
  cal_ind = in_index3[1:m]
  if(n0!=0)
    tein_ind = in_index3[(m+1):N]
  else
    tein_ind = NULL
  teout_ind = sample(out_ind, size = n1)
  cal = dataset[cal_ind,]
  te = dataset[c(tein_ind, teout_ind),]
  S_cal = predict.isolation_forest(isofo.model, cal, type = "score")
  S_te = predict.isolation_forest(isofo.model, te, type = "score")

  d_WMW = nout::d_MannWhitney(S_Y = S_te, S_X = S_cal, alpha=alpha)
  d_Sim = nout::d_Simes(S_X = S_cal, S_Y = S_te, alpha = alpha)
  StoSimes = nout::d_StoreySimes(S_X = S_cal, S_Y = S_te, alpha = alpha)
  d_StoSimes = StoSimes$d
  pi.not = StoSimes$pi.not
  d_BH = nout::d_benjhoch(S_X = S_cal, S_Y = S_te, alpha = alpha)
  d_StoBH = nout::d_StoreyBH(S_X = S_cal, S_Y = S_te, alpha = alpha)
  uniques = length(unique(c(S_cal, S_te)))

  # # outlier identification with WMW
  # conf.pval = sapply(1:n, function(j) (1+sum(S_cal >= S_te[j]))/(m+1))
  # confvalid.pval = conf.pval<alpha
  # confvalid.index = which(conf.pval<alpha)
  #
  # n.disc.WMW.cpp=0
  # n.disc.WMW=0
  # if(d_WMW>0 & length(confvalid.index)!=0){
  #   outlierTF.WMW.cpp = sapply(confvalid.index, function(h)
  #     nout::dselection_MannWhitney(S_Y = S_te, S_X = S_cal, S = h, alpha=alpha))
  #   #outlier.identified_MannWhitney = confvalid.index[as.logical(outlierTF)]
  #   n.disc.WMW.cpp = sum(outlierTF.WMW.cpp)
```

```
      #   outlierTF.WMW = sapply(confvalid.index, function(h)
      #     nout::dselection.prova_MannWhitney(S_Y = S_te, S_X = S_cal, S = h, alpha=alpha))
      #   n.disc.WMW = sum(outlierTF.WMW)
      # }
      #
      # # outlier identification with Simes
      # n.disc.Simes=0
      # n.disc.Simes2=0
      # if(d_Sim>0 & length(confvalid.index)!=0){
      #   outlierTF.Simes = sapply(confvalid.index, function(h)
      #       nout::dselection_Simes(S_Y = S_te, S_X = S_cal, S = h, alpha=alpha))
      #   #outlier.identified_Simes = confvalid.index[as.logical(outlierTF)]
      #   n.disc.Simes = sum(outlierTF.Simes)
      #   p = hommel(conf.pval)
      #   n.disc.Simes2 = sum(p@adjusted <= alpha)
      # }
      #
      # # outlier identification with StoreySimes
      # n.disc.StoSimes=0
      # if(d_StoSimes>0 & length(confvalid.index)!=0){
      #   outlierTF.StoSim = sapply(confvalid.index, function(h)
      #     nout::dselection_StoreySimes(S_Y = S_te, S_X = S_cal, S = h, alpha=alpha))
      #   #outlier.identified_StoSimes = confvalid.index[as.logical(outlierTFStoSim)]
      #   n.disc.StoSimes = sum(outlierTF.StoSim)
      # }

      return(list("d_BH" = d_BH,
                  "d_StoBH" = d_StoBH,
                  "d_Sim" = d_Sim,
                  # "n.disc.Simes" = n.disc.Simes,
                  # "n.disc.Simes2" = n.disc.Simes2,
                  "d_StoSimes" = d_StoSimes,
                  # "n.disc.StoSimes" = n.disc.StoSimes,
                  "d_WMW" = d_WMW,
                  # "n.disc.WMW" = n.disc.WMW,
                  # "n.disc.WMW.cpp" = n.disc.WMW.cpp,
                  "uniques" = uniques,
                  "n1" = n1,
                  "pi.not" = pi.not,
                  "alpha" = alpha))
    }
  }
}


estimatek = function(B, inlier_remaining, out_ind, isofo.model, dataset){
  ress = foreach(b = 1:B, .combine=c) %dopar% {
   inlier_ind = sample(inlier_remaining, size = 1)
   outlier_ind = sample(out_ind, size = 1)
   inlier = dataset[inlier_ind,]
   outlier = dataset[outlier_ind,]
   S_inlier = predict.isolation_forest(isofo.model, inlier, type = "score")
   S_outlier = predict.isolation_forest(isofo.model, outlier, type = "score")
```

```
   greater.logi = S_inlier<S_outlier

   return(greater.logi)
  }

  greater.prob = mean(ress)
  k=greater.prob/(1-greater.prob)
  return(k)
}
```

In the following we set the calibration set and the test set size, respectively l and m, so that the nominal level α is proportional to $\frac{m}{l+1}$. The train set size is equal to n and the number of iterations is $B = 10^4$.

ALOI dataset

The dataset is available at https://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI/literature/ALOI.

```
set.seed(321)

# Initializing parameters
B = 10^4
m = 199
l = 199
n = 20
alpha = n/(m+1)
n1s = seq(from=0, to=n, by=1)

dataset = read.arff("~/nout/trials/RealData/Datasets/Dataset ALOI/ALOI_withoutdupl.arff")
out_ind = which(dataset$outlier=="yes")
in_ind = which(dataset$outlier=="no")

cluster <- makeCluster(parallel::detectCores())
registerDoSNOW(cluster)
clusterEvalQ(cluster, {list(library(isotree), library(nout), library(hommel))})
```

```
## [[1]]
## [[1]][[1]]
## [1] "isotree"   "snow"      "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
##
## [[1]][[2]]
##  [1] "nout"      "isotree"   "snow"      "stats"     "graphics"  "grDevices"
##  [7] "utils"     "datasets"  "methods"   "base"
##
## [[1]][[3]]
##  [1] "hommel"    "nout"      "isotree"   "snow"      "stats"     "graphics"
##  [7] "grDevices" "utils"     "datasets"  "methods"   "base"
##
##
## [[2]]
## [[2]][[1]]
## [1] "isotree"   "snow"      "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
##
```

```
## [[2]][[2]]
##  [1] "nout"      "isotree"   "snow"      "stats"     "graphics"  "grDevices"
##  [7] "utils"     "datasets"  "methods"   "base"
##
## [[2]][[3]]
##  [1] "hommel"    "nout"      "isotree"   "snow"      "stats"     "graphics"
##  [7] "grDevices" "utils"     "datasets"  "methods"   "base"
##
##
## [[3]]
## [[3]][[1]]
## [1] "isotree"   "snow"      "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
##
## [[3]][[2]]
##  [1] "nout"      "isotree"   "snow"      "stats"     "graphics"  "grDevices"
##  [7] "utils"     "datasets"  "methods"   "base"
##
## [[3]][[3]]
##  [1] "hommel"    "nout"      "isotree"   "snow"      "stats"     "graphics"
##  [7] "grDevices" "utils"     "datasets"  "methods"   "base"
##
##
## [[4]]
## [[4]][[1]]
## [1] "isotree"   "snow"      "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
##
## [[4]][[2]]
##  [1] "nout"      "isotree"   "snow"      "stats"     "graphics"  "grDevices"
##  [7] "utils"     "datasets"  "methods"   "base"
##
## [[4]][[3]]
##  [1] "hommel"    "nout"      "isotree"   "snow"      "stats"     "graphics"
##  [7] "grDevices" "utils"     "datasets"  "methods"   "base"
```

```
clusterExport(cluster, list("n", "m", "l", "in_ind", "out_ind", "dataset", "alpha"))

tic()
modeltrain = TrainingIsoForest(l=l, dataset=dataset)
kest = estimatek(B=B, inlier_remaining=modeltrain$inlier_remaining,
          out_ind=out_ind, isofo.model=modeltrain$model, dataset=dataset)
res = lapply(1:length(n1s),
             function(j) CompareMethodNaturalOutliers(B=B, n1=n1s[j], n=n,
                                                      dataset=dataset,
                                                      isofo.model=modeltrain$model,
                                                      out_ind=out_ind,
                                                      inlier_remaining=modeltrain$inlier_remaining))
toc()
```

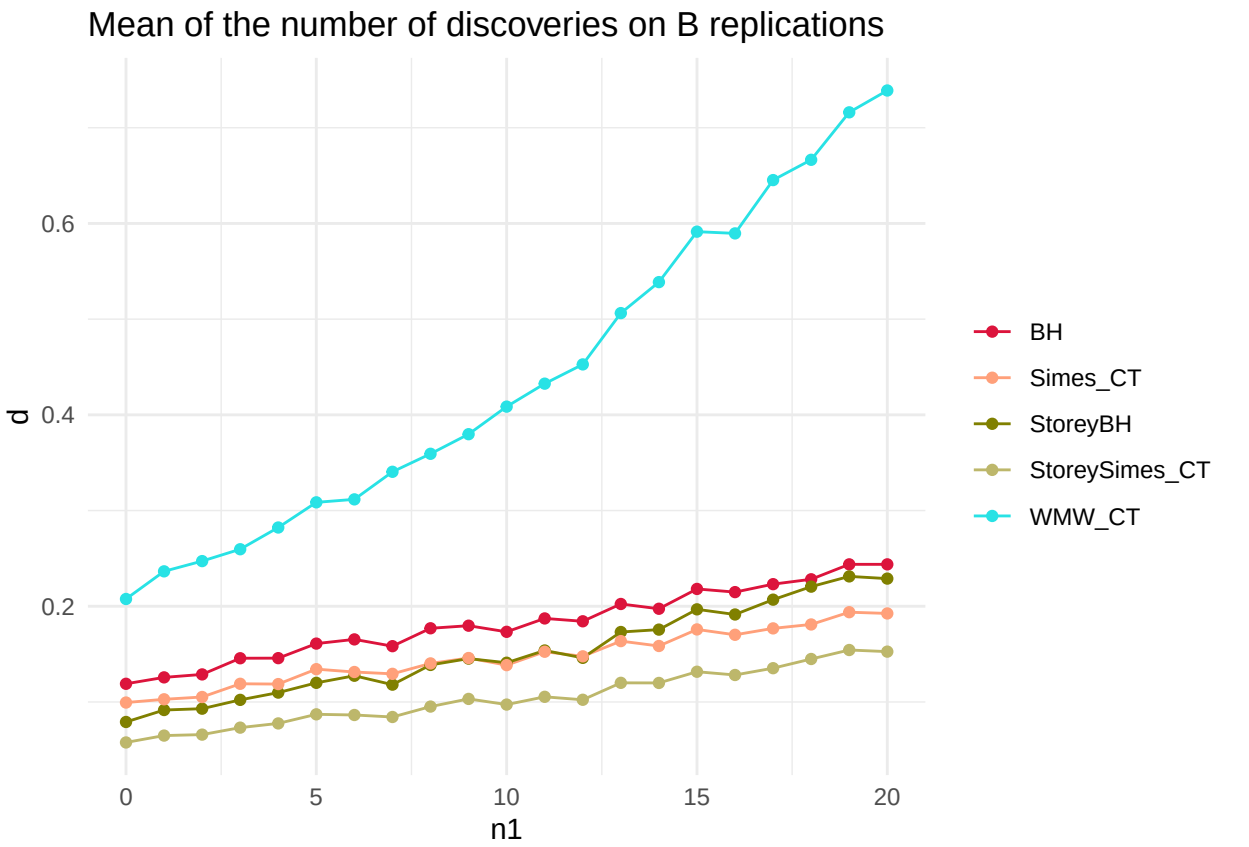
```
## 9146.08 sec elapsed
```

```
stopCluster(cluster)

kest
```

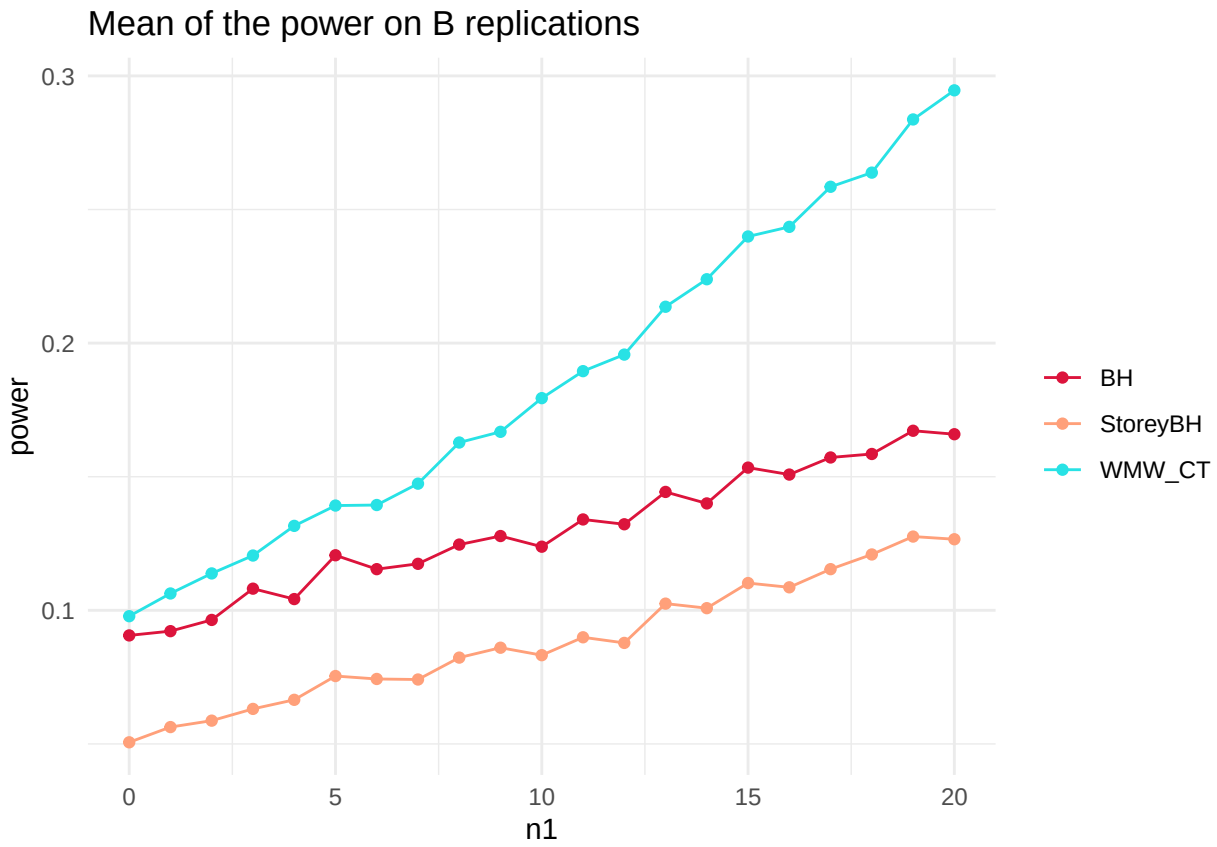
```
## [1] 1.205072
```

```
results = compact_results(res)

d_BH = vector()
d_StoBH = vector()
d_Sim = vector()
d_StoSimes = vector()
d_WMW = vector()

pow_BH = vector()
pow_StoBH = vector()
pow_Sim = vector()
pow_StoSimes = vector()
pow_WMW = vector()

# disc_Sim = vector()
# disc_StoSimes = vector()
# disc_WMW = vector()
# disc_WMW.cpp = vector()

for(j in 1:length(n1s)){
  d_BH[j] = results[[j]]$mean.lb.d[1]
  d_StoBH[j] = results[[j]]$mean.lb.d[2]
  d_Sim[j] = results[[j]]$mean.lb.d[3]
  d_StoSimes[j] = results[[j]]$mean.lb.d[4]
  d_WMW[j] = results[[j]]$mean.lb.d[5]

  pow_BH[j] = results[[j]]$mean.powerGlobalNull[1]
  pow_StoBH[j] = results[[j]]$mean.powerGlobalNull[2]
  pow_Sim[j] = results[[j]]$mean.powerGlobalNull[3]
  pow_StoSimes[j] = results[[j]]$mean.powerGlobalNull[4]
  pow_WMW[j] = results[[j]]$mean.powerGlobalNull[5]

  # disc_Sim[j] = results[[j]]$mean.n.disc[1]
  # disc_StoSimes[j] = results[[j]]$mean.n.disc[3]
  # disc_WMW[j] = results[[j]]$mean.n.disc[4]
  # disc_WMW.cpp[j] = results[[j]]$mean.n.disc[5]
}

# Plot discoveries
df <- data.frame(
  x = n1s,
  BH = d_BH,
  StoreyBH = d_StoBH,
  Simes_CT = d_Sim,
  StoreySimes_CT = d_StoSimes,
  WMW_CT = d_WMW
)
df_long <- tidyr::pivot_longer(df, cols = -x, names_to = "group", values_to = "y")

ggplot(df_long, aes(x = x, y = y, color = group)) +
  geom_line() +
  geom_point()+
```

```
  scale_color_manual(values = c("#DC143C", "#FFA07A", "#808000", "#BDB76B", 5)) +
  labs(x = "n1", y = "d", title = "Mean of the number of discoveries on B replications") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
# Plot power
dfpower <- data.frame(
  x = n1s,
  BH = pow_BH,
  StoreyBH = pow_StoBH,
  WMW_CT = pow_WMW
)
df_long_power <- tidyr::pivot_longer(dfpower, cols = -x, names_to = "group", values_to = "y")

# Plot the lines with different colors and legends
ggplot(df_long_power, aes(x = x, y = y, color = group)) +
  geom_line() +
  geom_point()+
  scale_color_manual(values = c("#DC143C","#FFA07A",5)) +
  labs(x = "n1", y = "power", title = "Mean of the power on B replications") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
# n.disc.tablelist = list()
# for(i in 1:length(n1s)){
#   n.disc.tablelist[[i]] = matrix(ncol = 5, nrow = 2)
#   colnames(n.disc.tablelist[[i]]) = c("Simes", "Simes2", "StoSimes", "WMW", "WMW.cpp")
#   rownames(n.disc.tablelist[[i]]) = c("mean.n.disc", "mean.d")
#   n.disc.tablelist[[i]][1,] = apply(results[[i]][["n.disc"]], MARGIN = 2, FUN = mean)
#   n.disc.tablelist[[i]][2,] = results[[i]]$mean.lb.d[c(3,3,4,5,5)]
# }
#
# for(i in 1:length(n1s)){
#   cat("\n")
#   cat(paste("n1=", n1s[i]))
#   cat("\n")
#   print(n.disc.tablelist[[i]])
# }

resALOI0.1 = list("raw.res"=res,
                  "k.est" = kest,
                  "compact.results" = results # ,
                  # "n.disc.tablelist" = n.disc.tablelist
                  )
save(resALOI0.1, file="~/nout/trials/RealData/PowerStudy/FinalSimu/ALOI/resALOI0.1")
```