Comparison between different local tests: Simes, Simes with Storey and Wilcoxon-Mann-Whitney using the Lehmann alternative distribution with k=2

2023-11-28

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 simulate outliers distribution so that it will be to the Lehmann's alternative with $k = 3$. Denoting inliers distribution by F, we are going to simulate the outliers distribution corresponding to F^k with $k = 3$ in order to perform a power analysis and to show that closed testing procedure with LMPI test statistic T_3 as local test is more powerful than closed testing with Simes local test with and without Storey estimator and than closed testing with Wilcoxon-Mann-Whitney local test.

Paths

```
# pathDatasets = file.path("C:", "Users", "chiar", "Documents",
#                          "BiomJ - RankTestsForOutlierDetection", "R code", "Datasets",
#                          fsep="\\")
#
# pathResults = file.path("C:", "Users", "chiar", "Documents",
#                         "BiomJ - RankTestsForOutlierDetection", "R code", fsep="\\")

pathDatasets = file.path("G:", "Il mio Drive", "PHD", "Progetto di ricerca",
                         "RankTestsForOutlierDetection",
                         "BiomJ - RankTestsForOutlierDetection", "R code", "Datasets",
                         fsep="\\")

pathResults = file.path("G:", "Il mio Drive", "PHD", "Progetto di ricerca",
                        "RankTestsForOutlierDetection",
                        "BiomJ - RankTestsForOutlierDetection", "R code", fsep="\\")
```

R functions and libraries

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

```r
# Lehmann's outlier distribution for k=3

compact_resultsk3 = function(res){

  results = list()
  for(j in 1:length(n1s)){
    lb.d = as.data.frame(
      cbind("d_BH"=unlist(res[[j]]["d_BH",]),
            "d_StoBH"=unlist(res[[j]]["d_StoBH",]),
            "d_Sim"=unlist(res[[j]]["d_Sim",]),
            "d_StoSimes"=unlist(res[[j]]["d_StoSimes",]),
            "d_WMW"=unlist(res[[j]]["d_WMW",]),
            "d_T3"=unlist(res[[j]]["d_T3",])
            )
    )
    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)

    results[[j]] = list("lb.d" = lb.d,
                        "mean.lb.d" = mean.lb.d,
                        "power.GlobalNull" = power.GlobalNull,
                        "mean.powerGlobalNull" = mean.powerGlobalNull,
                        "pi.not" = res[[j]]["pi.not",],
                        "n1" = res[[j]]["n1",1],
                        "alpha" = 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 = in_ind[! (in_ind %in% tr_ind)]

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

}


PredictIsoForest = function(isofo, dataset){

  inliers = dataset[isofo$inlier_remaining,]
  outliers = dataset[out_ind,]
  inliers.score = predict.isolation_forest(isofo$model, inliers, type = "score")
  outliers.score = predict.isolation_forest(isofo$model, outliers, type = "score")
```

```r
  return(list("inliers.score" = inliers.score,
              "outliers.score" = outliers.score))

}

CompareMethodLehmannOutliersk3 = function(B, n1, n, k, inliers_score, isofo.model, dataset){

  n0 = n-n1
  N = n0 + m + k*n1

  foreach(b = 1:B, .combine=cbind) %dopar% {

    S_cal.te = sample(inliers_score, size = N)
    S_cal = S_cal.te[1:m]
    S_remaining = S_cal.te[(m+1):N]

    if(n1==0)
      S_te = sample(S_remaining, size = n0)
    if(n1==n)
      S_te = sapply(1:n1, FUN=function(i) max(S_remaining[(1+k*(i-1)):(i*k)]))
    if(0<n1&n1<n)
      S_te = c(S_remaining[(1+k*n1):(n0+k*n1)],
               sapply(1:n1, FUN=function(i) max(S_remaining[(1+k*(i-1)):(i*k)])))

    d_WMW = nout::d_MannWhitney(S_Y = S_te, S_X = S_cal, alpha=alpha)
    d_T3 = nout::d_MannWhitneyk3(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)

    return(list("d_BH" = d_BH,
                "d_StoBH" = d_StoBH,
                "d_Sim" = d_Sim,
                "d_StoSimes" = d_StoSimes,
                "d_WMW" = d_WMW,
                "d_T3" = d_T3,
                "n1" = n1,
                "pi.not" = pi.not,
                "alpha" = alpha))
  }
}
```

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 $B = 10^5$.

Digits dataset

The dataset is available at http://odds.cs.stonybrook.edu/pendigits-dataset.

```r
set.seed(321)
```

```r
# Initializing parameters
B = 10^4
l = 1999
m = 1999
n = 200
alpha = n/(m+1)
n1s = seq(from=0, to=n, by=1)


data = readMat(paste0(pathDatasets,"\\pendigits.mat"))
dataset = cbind(data$X, data$y); colnames(dataset)[ncol(dataset)] = "y"
in_ind = which(dataset[,ncol(dataset)]==0)
out_ind = which(dataset[,ncol(dataset)]==1)
theta = length(out_ind)/nrow(dataset) # proportion of outliers in the entire dataset
```

```r
#,eval = FALSE
cluster <- makeCluster(parallel::detectCores()-1)
registerDoSNOW(cluster)
clusterEvalQ(cluster, {list(library(isotree), library(nout))})
```

```
## [[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"
##
##
## [[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"
##
##
## [[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"
```

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

modeltrain = TrainingIsoForest(l=l, dataset=dataset)
scores = PredictIsoForest(isofo=modeltrain, dataset=dataset)
```

```
stopCluster(cluster)
scores_1999_v2 = scores
save(scores_1999_v2, file="~/nout/Examples/Digits/Lehmannk2/scores_1999_v2")
```

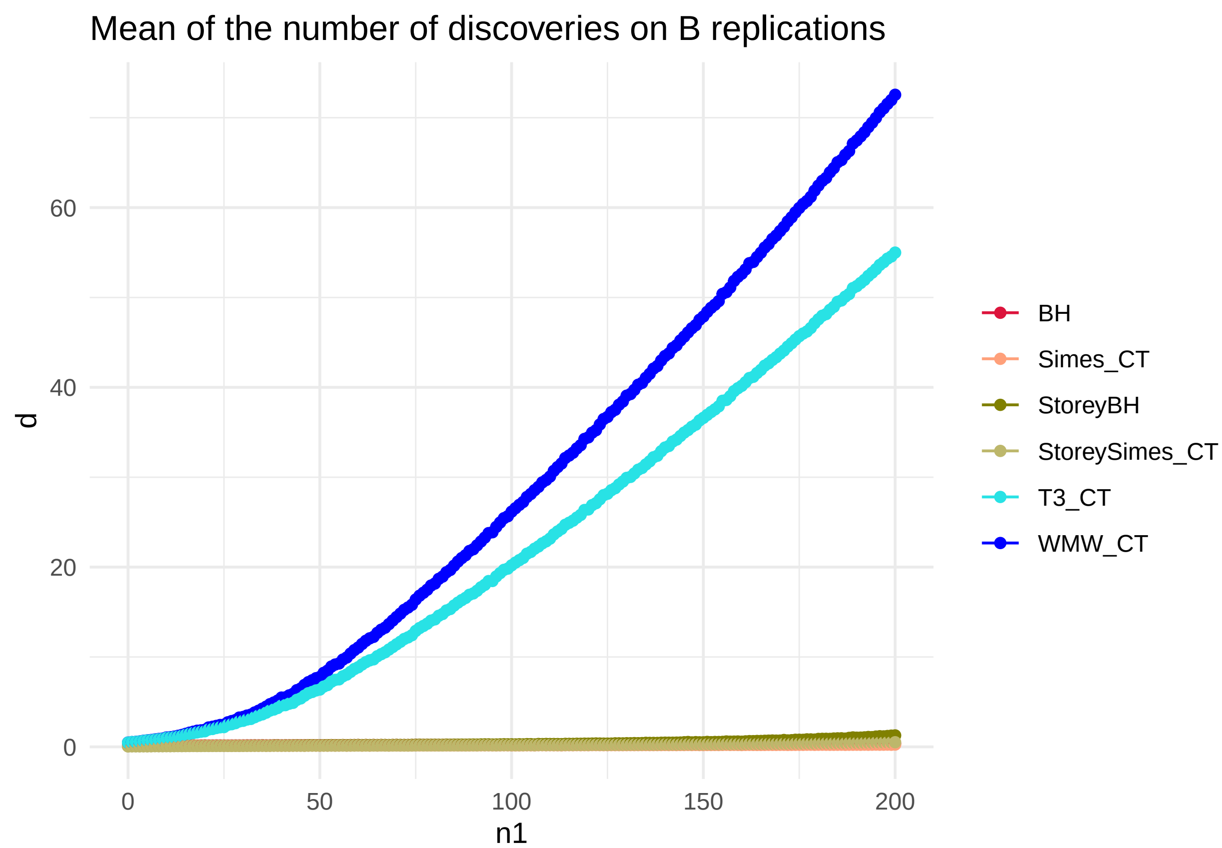
```
cluster <- makeCluster(parallel::detectCores()-1)
registerDoSNOW(cluster)
clusterEvalQ(cluster, {list(library(isotree), library(nout))})
```

```
## [[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"
##
##
## [[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"
##
##
## [[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"
```

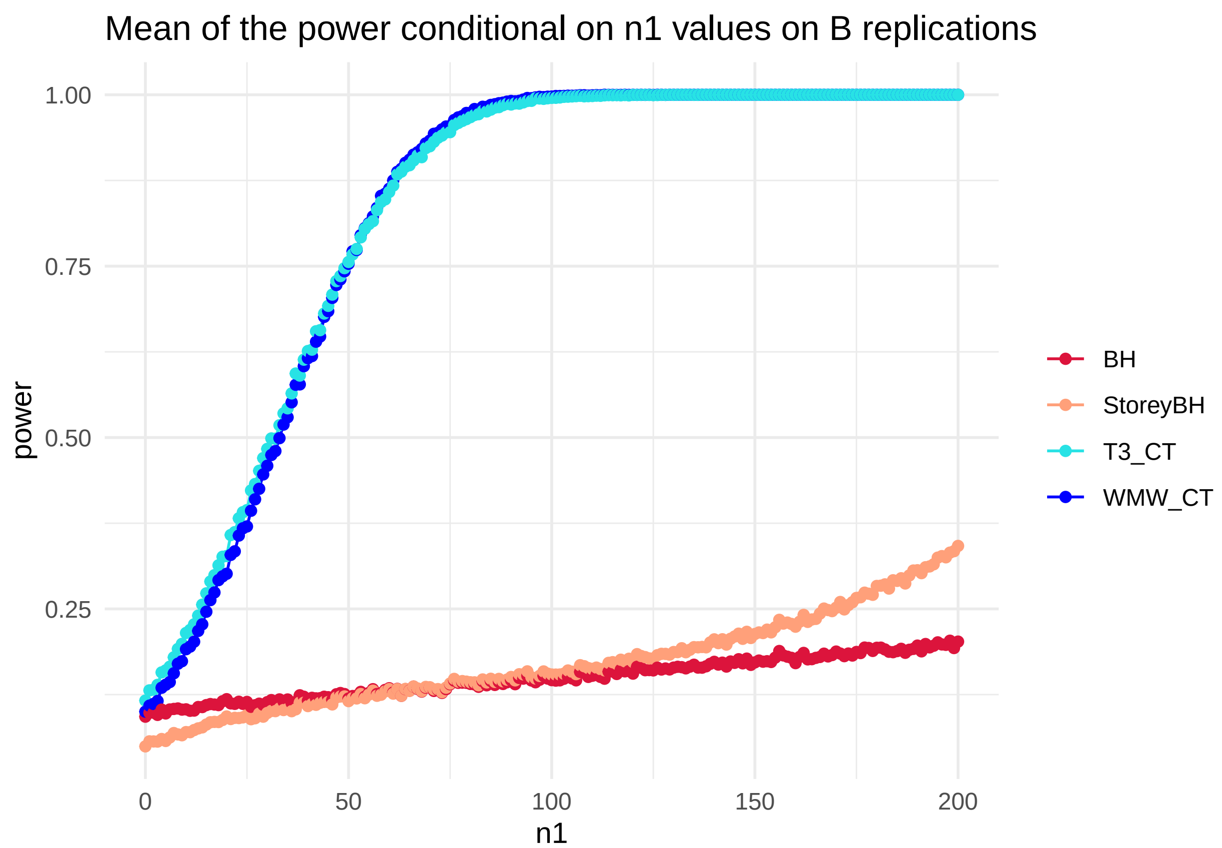
```
clusterExport(cluster, list("n", "m", "l", "in_ind", "out_ind", "dataset", "alpha"))

res = lapply(1:length(n1s),
             function(j) CompareMethodLehmannOutliersk3(B=B, k=2, n1=n1s[j], n=n,
                                                        dataset=dataset,
                                                        isofo.model=modeltrain$model,
                                                        inliers_score=scores$inliers.score))
stopCluster(cluster)

resDigits0.1k2_1999_v2 = list("raw.res"=res)
save(resDigits0.1k2_1999_v2,
     file="~/nout/Examples/Digits/Lehmannk2/resDigits0.1k2_1999_v2")
```

```
results = compact_resultsk3(res)

# load(file="~/nout/Examples/Digits/Lehmannk2/resDigits0.1k2_1999_v2")
# results = compact_resultsk3(resDigits0.1k2_1999_v2$raw.res)
```

```r
d_BH = vector()
d_StoBH = vector()
d_Sim = vector()
d_StoSimes = vector()
d_WMW = vector()
d_T3 = vector()

pow_BH = vector()
pow_StoBH = vector()
pow_Sim = vector()
pow_StoSimes = vector()
pow_WMW = vector()
pow_T3 = 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]]$mean.lb.d[5]
  d_T3[j] = results[[j]]$mean.lb.d[6]

  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]]$mean.powerGlobalNull[5]
  pow_T3[j] = results[[j]]$mean.powerGlobalNull[6]
}

# Plot discoveries conditional on n1
df <- data.frame(
  x = n1s,
  BH = d_BH,
  StoreyBH = d_StoBH,
  Simes_CT = d_Sim,
  StoreySimes_CT = d_StoSimes,
  WMW_CT = d_WMW,
  T3_CT = d_T3
)
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, "blue")) +
  labs(x = "n1", y = "d", title = "Mean of the number of discoveries on B replications") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
# Plot power conditional on n1
dfpower <- data.frame(
  x = n1s,
  BH = pow_BH,
  StoreyBH = pow_StoBH,
  WMW_CT = pow_WMW,
  T3_CT = pow_T3
)
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, "blue")) +
  labs(x = "n1", y = "power", title = "Mean of the power conditional on n1 values on B replications") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
# Table unconditional power
thetas = seq(from = 0, to = 1, by = 0.02)
probsn1 = sapply(thetas,
                 function(theta) sapply(0:n,
                                        function(k) choose(n,k)*(1-theta)^(n-k)*theta^(k)))
colnames(probsn1) = as.character(thetas)
rownames(probsn1) = as.character(0:n)


unconditional.power = cbind("uncond.pow_BH" = apply(pow_BH*probsn1, MARGIN = 2, sum),
                            "uncond.pow_StoreyBH" = apply(pow_StoBH*probsn1, MARGIN = 2, sum),
                            "uncond.pow_WMW" = apply(pow_WMW*probsn1, MARGIN = 2, sum),
                            "uncond.pow_T3" = apply(pow_T3*probsn1, MARGIN = 2, sum))
print(unconditional.power)
```

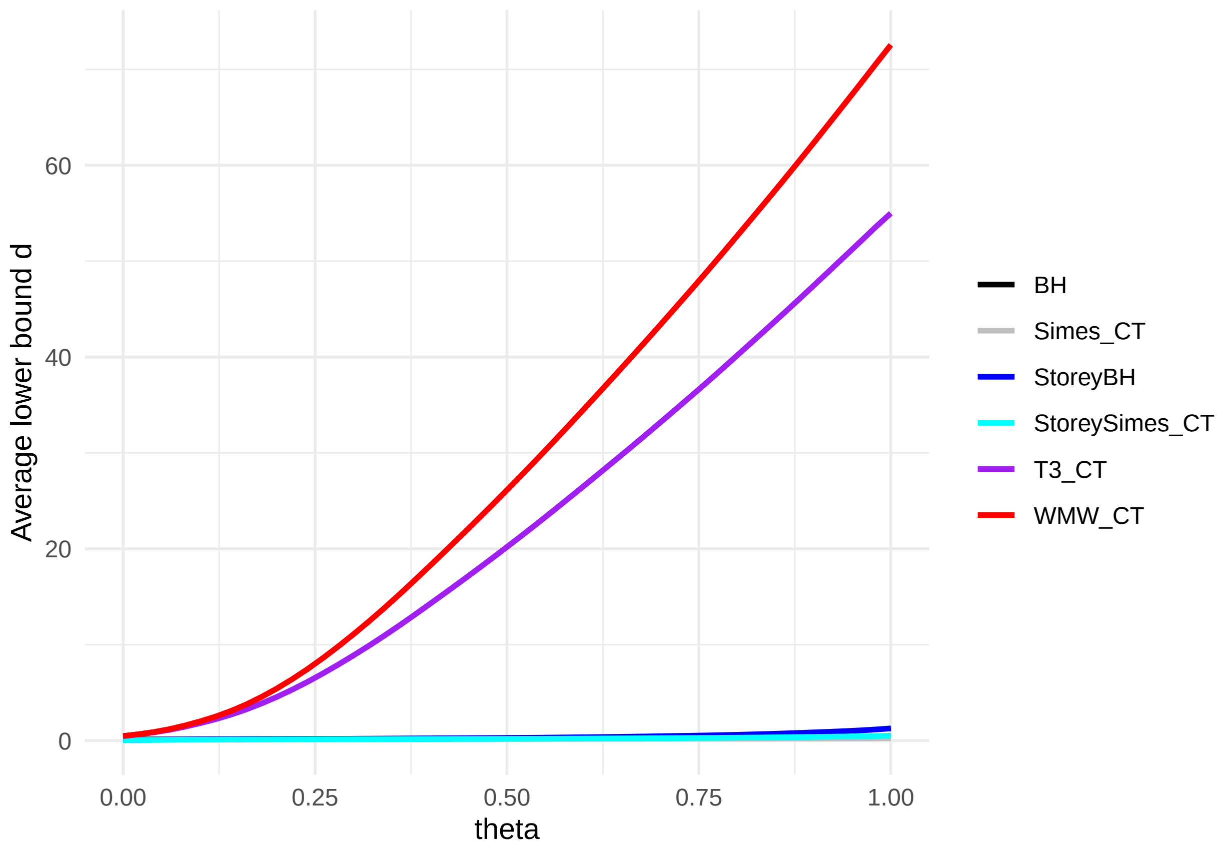
```
##      uncond.pow_BH uncond.pow_StoreyBH uncond.pow_WMW uncond.pow_T3
## 0       0.09280000          0.04950000      0.1002000     0.1163000
## 0.02    0.09966552          0.05918873      0.1300500     0.1521419
## 0.04    0.10286391          0.06639852      0.1674083     0.1906311
## 0.06    0.10537352          0.07394281      0.2101212     0.2348630
## 0.08    0.10928017          0.08184120      0.2596908     0.2854155
## 0.1     0.11182575          0.08781611      0.3124935     0.3384440
## 0.12    0.11244751          0.09184807      0.3679759     0.3931850
## 0.14    0.11318198          0.09579187      0.4267870     0.4498391
## 0.16    0.11490953          0.10039414      0.4878027     0.5072626
## 0.18    0.11703679          0.10515287      0.5493819     0.5646938
```

```
## 0.2      0.11910878          0.10964564      0.6099949      0.6213723
## 0.22     0.12110265          0.11382396      0.6684223      0.6760383
## 0.24     0.12313207          0.11775920      0.7233961      0.7272198
## 0.26     0.12526690          0.12147207      0.7736677      0.7738663
## 0.28     0.12742390          0.12491609      0.8184559      0.8156364
## 0.3      0.12942185          0.12802149      0.8573037      0.8523061
## 0.32     0.13123921          0.13083114      0.8900115      0.8835612
## 0.34     0.13311645          0.13357981      0.9168859      0.9095267
## 0.36     0.13528332          0.13650247      0.9386222      0.9308695
## 0.38     0.13762445          0.13954560      0.9558744      0.9482657
## 0.4      0.13980424          0.14245945      0.9691181      0.9621214
## 0.42     0.14170193          0.14517663      0.9788615      0.9727870
## 0.44     0.14345794          0.14784322      0.9857914      0.9807653
## 0.46     0.14515274          0.15053514      0.9906509      0.9866472
## 0.48     0.14674695          0.15323149      0.9940335      0.9909414
## 0.5      0.14830800          0.15602849      0.9963282      0.9940080
## 0.52     0.15004258          0.15915388      0.9978137      0.9961191
## 0.54     0.15209534          0.16278098      0.9987345      0.9975236
## 0.56     0.15444237          0.16692238      0.9992934      0.9984431
## 0.58     0.15691879          0.17141360      0.9996257      0.9990408
## 0.6      0.15927740          0.17596832      0.9998118      0.9994218
## 0.62     0.16133964          0.18040910      0.9999073      0.9996585
## 0.64     0.16315321          0.18486282      0.9999544      0.9998048
## 0.66     0.16493632          0.18961712      0.9999788      0.9998957
## 0.68     0.16686557          0.19481759      0.9999917      0.9999498
## 0.7      0.16893986          0.20036294      0.9999975      0.9999787
## 0.72     0.17104334          0.20606847      0.9999995      0.9999924
## 0.74     0.17314905          0.21192093      0.9999999      0.9999979
## 0.76     0.17532043          0.21808584      1.0000000      0.9999996
## 0.78     0.17738047          0.22460173      1.0000000      1.0000000
## 0.8      0.17902270          0.23142587      1.0000000      1.0000000
## 0.82     0.18050000          0.23889111      1.0000000      1.0000000
## 0.84     0.18240721          0.24739034      1.0000000      1.0000000
## 0.86     0.18489166          0.25690543      1.0000000      1.0000000
## 0.88     0.18765821          0.26737197      1.0000000      1.0000000
## 0.9      0.18959081          0.27817130      1.0000000      1.0000000
## 0.92     0.19000872          0.28833394      1.0000000      1.0000000
## 0.94     0.19129021          0.29866665      1.0000000      1.0000000
## 0.96     0.19484305          0.31061491      1.0000000      1.0000000
## 0.98     0.19844369          0.32475649      1.0000000      1.0000000
## 1        0.20230000          0.34190000      1.0000000      1.0000000
```

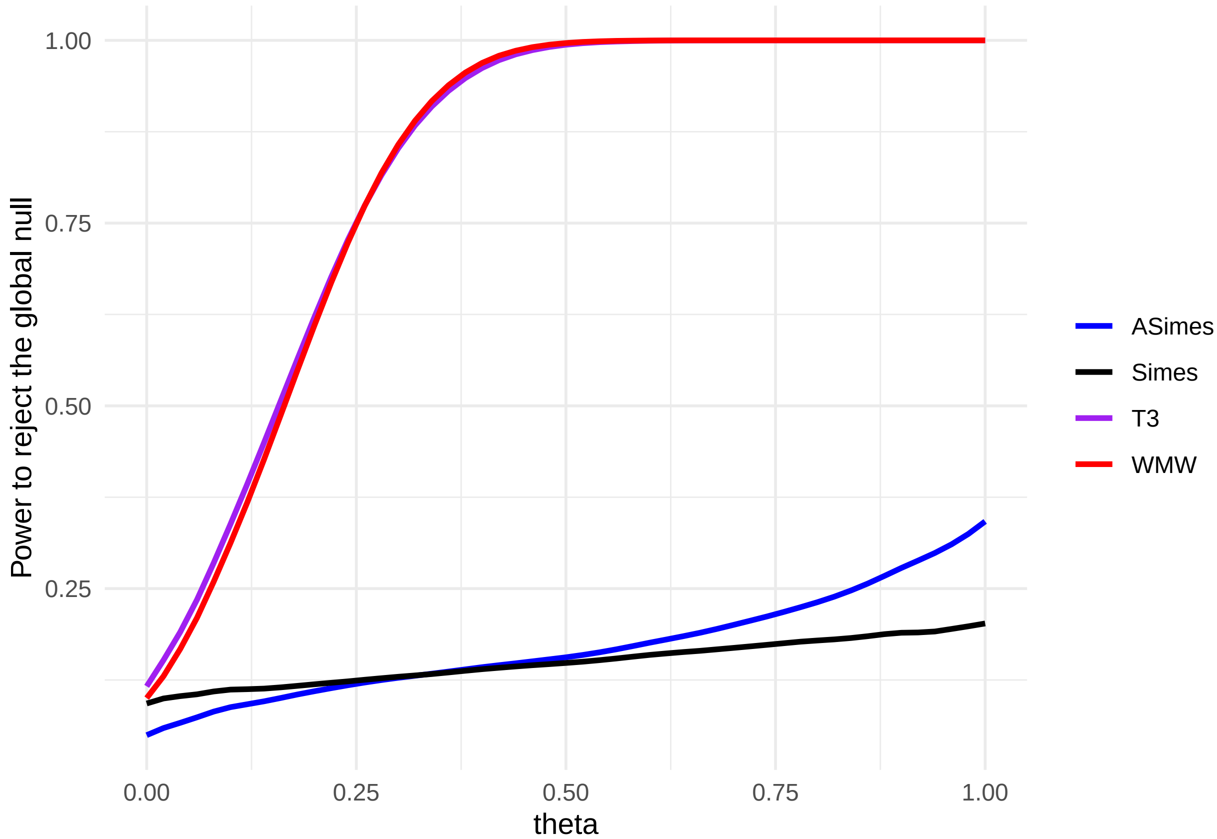
```
# load(file="~/nout/Examples/Digits/Lehmannk2/resDigits0.1k2_1999_v2")
# results = compact_resultsk3(resDigits0.1k2_1999_v2$raw.res)

# Compacting intermediate results in a matrix

d_BH = vector()
d_StoBH = vector()
d_Sim = vector()
d_StoSimes = vector()
d_WMW = vector()
d_T3 = vector()
```

```
pow.rejGlob_BH = vector()
pow.rejGlob_StoBH = vector()
pow.rejGlob_Sim = vector()
pow.rejGlob_StoSimes = vector()
pow.rejGlob_WMW = vector()
pow.rejGlob_T3 = 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]]$mean.lb.d[5]
  d_T3[j] = results[[j]]$mean.lb.d[6]

  pow.rejGlob_BH[j] = results[[j]]$mean.powerGlobalNull[1]
  pow.rejGlob_StoBH[j] = results[[j]]$mean.powerGlobalNull[2]
  pow.rejGlob_Sim[j] = results[[j]]$mean.powerGlobalNull[3]
  pow.rejGlob_StoSimes[j] = results[[j]]$mean.powerGlobalNull[4]
  pow.rejGlob_WMW[j] = results[[j]]$mean.powerGlobalNull[5]
  pow.rejGlob_T3[j] = results[[j]]$mean.powerGlobalNull[6]

}

lb.d = matrix(nrow = (n+1), ncol = 6)
rownames(lb.d) = as.character(n1s)
colnames(lb.d) = c("FDR-BH", "FDR-Storey", "CT-Simes",
                   "CT-Storey", "CT-WMW", "CT-T3")

lb.d[,1] = d_BH
lb.d[,2] = d_StoBH
lb.d[,3] = d_Sim
lb.d[,4] = d_StoSimes
lb.d[,5] = d_WMW
lb.d[,6] = d_T3

pow.rejGlob = matrix(nrow = (n+1), ncol = 6)
rownames(pow.rejGlob) = as.character(seq(from=0, to=n, by=1))
colnames(pow.rejGlob) = c("FDR-BH", "FDR-Storey", "CT-Simes",
                          "CT-Storey", "CT-WMW", "CT-T3")
pow.rejGlob[,1] = pow.rejGlob_BH
pow.rejGlob[,2] = pow.rejGlob_StoBH
pow.rejGlob[,3] = pow.rejGlob_Sim
pow.rejGlob[,4] = pow.rejGlob_StoSimes
pow.rejGlob[,5] = pow.rejGlob_WMW
pow.rejGlob[,6] = pow.rejGlob_T3

matrixDigits0.1k2_1999_v2 = list("lb.d.matrix" = lb.d,
                              "pow.rejGlob.matrix" = pow.rejGlob)
save(matrixDigits0.1k2_1999_v2,
     file = paste0("~/nout/Examples/Digits/Lehmannk2","/matrixDigits0.1k2_1999_v2"))
```

```r
# load(file = paste0("~/nout/Examples/Digits/Lehmannk2","/matrixDigits0.1k2_1999_v2"))

res = matrixDigits0.1k2_1999_v2

thetas = seq(0,1, length.out=51)

pow_BH = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,1])),4)
pow_StoBH = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,2])),4)
pow_Simes = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,3])),4)
pow_ASimes = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,4])),4)
pow_WMW = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,5])),4)
pow_T3 = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$pow.rejGlob.matrix[,6])),4)

lb.d.BH = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,1])),4)
lb.d.StoBH = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,2])),4)
lb.d.Simes = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,3])),4)
lb.d.ASimes = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,4])),4)
lb.d.WMW = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,5])),4)
lb.d.T3 = round(sapply(thetas, function(p)
  sum( dbinom(0:n,size=n,prob=p) * res$lb.d.matrix[,6])),4)


# Plot lower bound d
df <- data.frame(
  x = thetas,
  BH = lb.d.BH,
  StoreyBH = lb.d.StoBH,
  Simes_CT = lb.d.Simes,
  StoreySimes_CT = lb.d.ASimes,
  WMW_CT = lb.d.WMW,
  T3_CT = lb.d.T3
)
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(size=1) +
  scale_color_manual(values = c("black","gray","blue", "cyan","purple","red")) +
  labs(x = "theta", y = "Average lower bound d") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```r
# Plot power
dfpower <- data.frame(
  x = thetas,
  Simes = pow_BH,
  ASimes = pow_StoBH,
  WMW = pow_WMW,
  T3 = pow_T3
)
df_long_power <- tidyr::pivot_longer(dfpower, cols = -x, names_to = "group", values_to = "y")

ggplot(df_long_power, aes(x = x, y = y, color = group)) +
  geom_line(size=1) +
  scale_color_manual(values = c("blue","black","purple","red")) +
  labs(x = "theta", y = "Power to reject the global null") +
  theme_minimal() +
  theme(legend.title = element_blank())
```




```
pow_WMW
```

```
##  [1] 0.1002 0.1301 0.1674 0.2101 0.2597 0.3125 0.3680 0.4268 0.4878 0.5494
## [11] 0.6100 0.6684 0.7234 0.7737 0.8185 0.8573 0.8900 0.9169 0.9386 0.9559
## [21] 0.9691 0.9789 0.9858 0.9907 0.9940 0.9963 0.9978 0.9987 0.9993 0.9996
## [31] 0.9998 0.9999 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [41] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [51] 1.0000
```

```
pow_T3
```

```
##  [1] 0.1163 0.1521 0.1906 0.2349 0.2854 0.3384 0.3932 0.4498 0.5073 0.5647
## [11] 0.6214 0.6760 0.7272 0.7739 0.8156 0.8523 0.8836 0.9095 0.9309 0.9483
## [21] 0.9621 0.9728 0.9808 0.9866 0.9909 0.9940 0.9961 0.9975 0.9984 0.9990
## [31] 0.9994 0.9997 0.9998 0.9999 0.9999 1.0000 1.0000 1.0000 1.0000 1.0000
## [41] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [51] 1.0000
```