Comparison between different local tests: Simes, Simes with Storey and Wilcoxon-Mann-Whitney using the Lehmann alternative distribution with k=3

2023-11-16

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){
  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",]),
            "d_T3"=unlist(res[[j]][rownames(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)

    # 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",]),
    #         "n.disc.T3" = unlist(res[[j]][rownames(res[[j]])=="n.disc.T3",])
    #         )
    # )
    # 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",],
                        "S_cal" = (res[[j]][rownames(res[[j]])=="S_cal",]),
                        "S_te" = (res[[j]][rownames(res[[j]])=="S_te",]),
                        "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))

}




CompareMethodLehmannOutliersk3 = function(B, n1, n, k, out_ind, inlier_remaining, isofo.model, dataset)

  n0 = n-n1
  foreach(b = 1:B, .combine=cbind) %dopar% {

    N = n0 + m + k*n1
    in_index3 = sample(inlier_remaining, size = N)
    cal_ind = in_index3[1:m]
    te_ind.augmented = in_index3[(m+1):N]
    cal = dataset[cal_ind,]
    te = dataset[te_ind.augmented,]
    S_cal = predict.isolation_forest(isofo.model, cal, type = "score")
    augmented.S_te = predict.isolation_forest(isofo.model, te, type = "score")

    if(n1==0)
      S_te = augmented.S_te
    if(n1==n)
      S_te = sapply(1:n1, FUN=function(i) max(augmented.S_te[(1+k*(i-1)):(i*k)]))
    if(0<n1&n1<n)
      S_te = c(augmented.S_te[(1+k*n1):(n0+k*n1)],
               sapply(1:n1, FUN=function(i) max(augmented.S_te[(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)
      uniques = length(unique(c(S_cal, S_te)))

      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,
                  "S_cal" = S_cal,
                  "S_te" = S_te,
```

```
                   "uniques" = uniques,
                   "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.

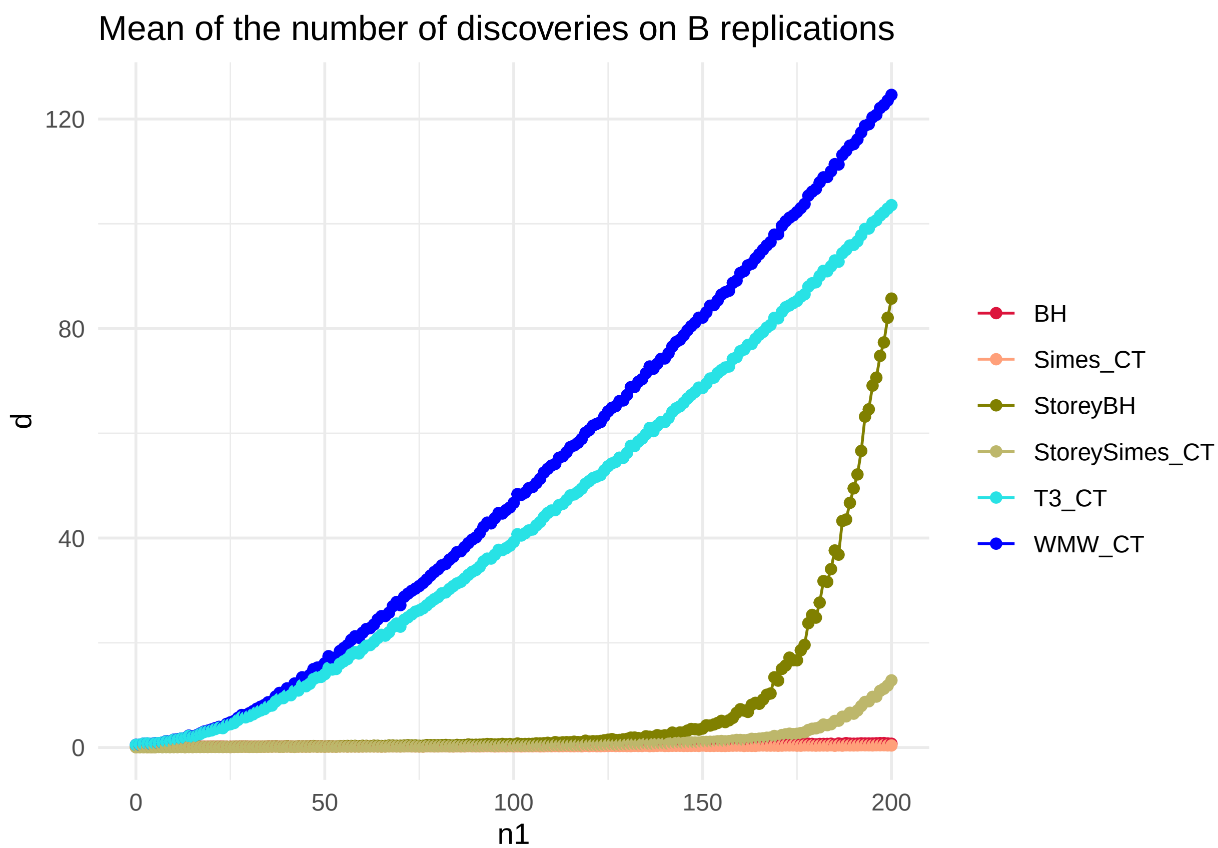
```
set.seed(321)

# Initializing parameters
B = 10^3
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

cluster <- makeCluster(parallel::detectCores())
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"
##
##
```

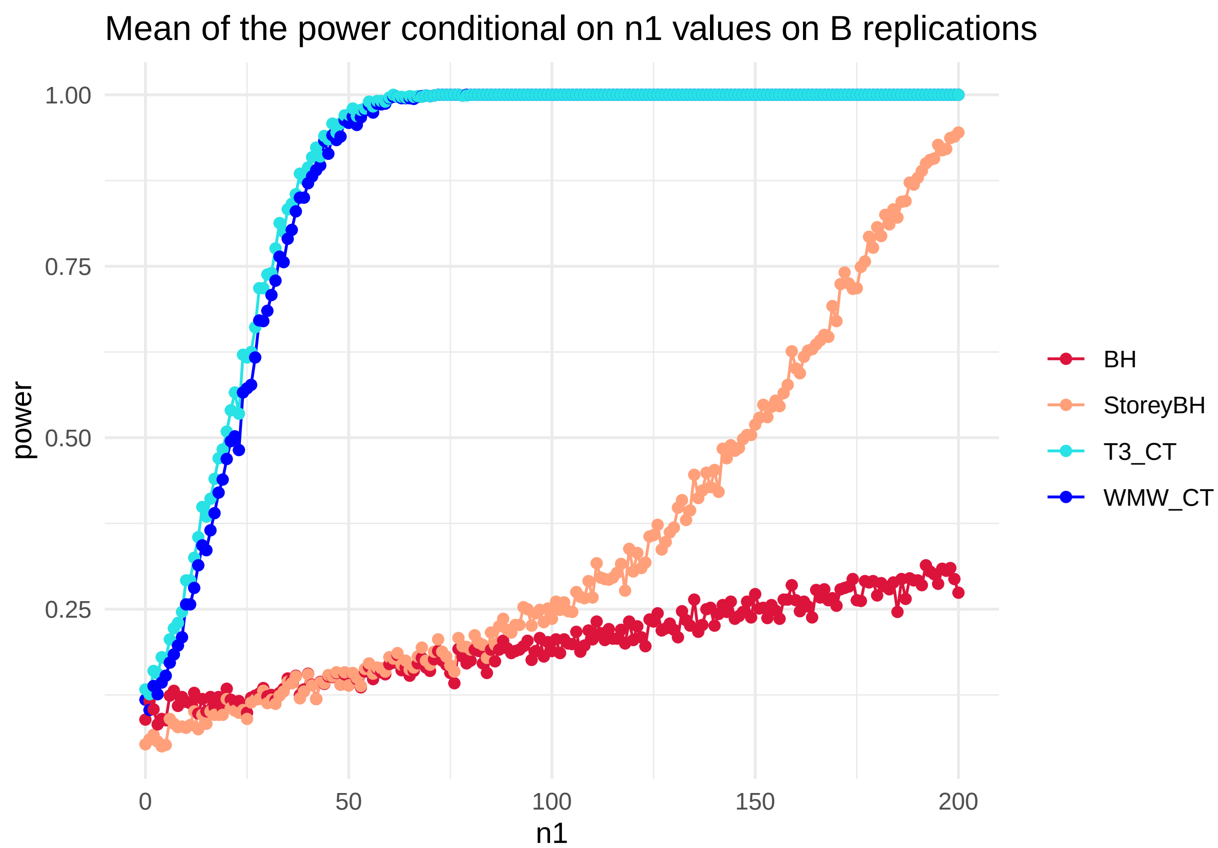
```
## [[3]]
## [[3]][[1]]
## [1] "isotree"   "snow"      "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"
##
## [[3]][[2]]
##  [1] "nout"      "isotree"   "snow"      "stats"     "graphics"  "grDevices"
##  [7] "utils"     "datasets"  "methods"   "base"
##
##
## [[4]]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

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

modeltrain = TrainingIsoForest(l=l, dataset=dataset)
res = lapply(1:length(n1s),
             function(j) CompareMethodLehmannOutliersk3(B=B, k=3, n1=n1s[j], n=n,
                                                        dataset=dataset,
                                                        isofo.model=modeltrain$model,
                                                        out_ind=out_ind,
                                                        inlier_remaining=modeltrain$inlier_remaining))
stopCluster(cluster)

results = compact_resultsk3(res)

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]
```

```r
  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())
```




```r
# 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(1:n,
                                        function(k) choose(n,k)*(1-theta)^(n-k)*theta^(k)))
colnames(probsn1) = as.character(thetas)
rownames(probsn1) = as.character(1:n)
unconditional.power = cbind("uncond.pow_BH" = apply(pow_BH[-1]*probsn1, MARGIN = 2, sum),
                            "uncond.pow_StoreyBH" = apply(pow_StoBH[-1]*probsn1, MARGIN = 2, sum),
                            "uncond.pow_WMW" = apply(pow_WMW[-1]*probsn1, MARGIN = 2, sum),
                            "uncond.pow_T3" = apply(pow_T3[-1]*probsn1, MARGIN = 2, sum))
print(unconditional.power)
```

```
##      uncond.pow_BH uncond.pow_StoreyBH uncond.pow_WMW uncond.pow_T3
## 0       0.00000000          0.00000000      0.0000000     0.0000000
## 0.02    0.09757478          0.06155762      0.1438838     0.1733931
## 0.04    0.11349128          0.07726015      0.2063804     0.2416917
## 0.06    0.11469768          0.08600940      0.2864479     0.3280272
## 0.08    0.11438750          0.09404077      0.3723709     0.4188305
## 0.1     0.11583919          0.10104485      0.4599576     0.5084081
## 0.12    0.11824754          0.10743914      0.5492642     0.5977596
## 0.14    0.12317214          0.11534618      0.6387189     0.6850096
## 0.16    0.12972939          0.12413985      0.7221881     0.7640578
## 0.18    0.13568546          0.13203506      0.7948409     0.8301154
## 0.2     0.14016278          0.13833866      0.8545091     0.8820784
## 0.22    0.14395601          0.14378094      0.9007099     0.9211392
```

```
## 0.24     0.14786308          0.14918318      0.9346013     0.9493213
## 0.26     0.15224132          0.15508636      0.9585029     0.9688038
## 0.28     0.15705559          0.16157058      0.9748914     0.9817086
## 0.3      0.16163952          0.16797321      0.9856667     0.9898360
## 0.32     0.16534052          0.17366684      0.9922708     0.9946044
## 0.34     0.16821114          0.17877540      0.9960365     0.9972067
## 0.36     0.17081188          0.18397818      0.9980890     0.9985787
## 0.38     0.17377421          0.19013809      0.9991588     0.9992980
## 0.4      0.17747641          0.19797519      0.9996681     0.9996665
## 0.42     0.18179876          0.20764120      0.9998818     0.9998496
## 0.44     0.18619726          0.21848743      0.9999618     0.9999387
## 0.46     0.19017327          0.22943188      0.9999891     0.9999791
## 0.48     0.19376060          0.23990814      0.9999974     0.9999944
## 0.5      0.19745737          0.25047971      0.9999995     0.9999988
## 0.52     0.20161534          0.26222784      0.9999999     0.9999998
## 0.54     0.20600650          0.27557204      1.0000000     1.0000000
## 0.56     0.21010636          0.29011637      1.0000000     1.0000000
## 0.58     0.21378242          0.30573027      1.0000000     1.0000000
## 0.6      0.21748054          0.32320754      1.0000000     1.0000000
## 0.62     0.22165416          0.34355113      1.0000000     1.0000000
## 0.64     0.22634428          0.36701277      1.0000000     1.0000000
## 0.66     0.23138791          0.39295532      1.0000000     1.0000000
## 0.68     0.23657258          0.42031967      1.0000000     1.0000000
## 0.7      0.24150175          0.44825081      1.0000000     1.0000000
## 0.72     0.24571717          0.47643418      1.0000000     1.0000000
## 0.74     0.24911582          0.50501347      1.0000000     1.0000000
## 0.76     0.25215686          0.53454544      1.0000000     1.0000000
## 0.78     0.25544304          0.56570636      1.0000000     1.0000000
## 0.8      0.25909796          0.59856537      1.0000000     1.0000000
## 0.82     0.26323349          0.63315074      1.0000000     1.0000000
## 0.84     0.26836467          0.67056673      1.0000000     1.0000000
## 0.86     0.27403555          0.71073700      1.0000000     1.0000000
## 0.88     0.27852747          0.75145439      1.0000000     1.0000000
## 0.9      0.28038948          0.79078013      1.0000000     1.0000000
## 0.92     0.28091023          0.82661735      1.0000000     1.0000000
## 0.94     0.28666363          0.86134193      1.0000000     1.0000000
## 0.96     0.29721511          0.89535987      1.0000000     1.0000000
## 0.98     0.30202709          0.92182341      1.0000000     1.0000000
## 1        0.27400000          0.94500000      1.0000000     1.0000000
```

```
resDigits0.1k3 = list("raw.res"=res,
                      "unconditional.power" = unconditional.power,
                      "compact.results" = results)
save(resDigits0.1k3, file="~/nout/Examples/Digits/Lehmannk3/resDigits0.1k3")
```