CTGT Shortcut - Permutations

Anna Vesely

November 2019

1 Introduction and Notation

Permutation test with centered statistics. Let $F = \{1, \dots, f\}$ be the full model, while $S \subseteq F$ is the subset under test by closed testing: H_S is rejected if and only if H_V is rejected for all V such that $S \subseteq V \subseteq F$.

Assume that T_i is the standardized test statistic for the univariate hypothesis H_i and that, for each $V \subseteq F$, the test statistic for H_V can be written as the sum

$$T_V = \sum_{i \in V} T_i.$$

Given a set $\mathcal{P}$ of random permutations (including the identity $\pi_1 = \text{id}$) with $|\mathcal{P}| = B$, consider the matrix

$$T = \begin{pmatrix} T_1 & \dots & T_f \\ T_1^2 & \dots & T_f^2 \\ \vdots & & \vdots \\ T_1^B & \dots & T_f^B \end{pmatrix}$$

Assume, for simplicity, that the columns are already sorted so that $T_1 \geq \dots \geq T_f$. Subsequently, define the centered test statistics

$$D_i^\pi = T_i^\pi - T_i,$$

so that the observed values are all $D_i = 0$, and the variability due to T_i is excluded. The p-value corresponding to H_V is

$$\text{p-value}(V) = \frac{\#\{\pi : T_V^\pi \geq T_V\}}{B} = \frac{\#\{\pi : D_V^\pi \geq 0\}}{B}.$$

Test for a fixed superset size. Let $m = f - s$ be the number of indices in $F \setminus S$. If V is a superset of S, then its size is $|V| = s + v$, with $v \in \{0, 1, \dots, m\}$.

Fix a value $v \in \{0, 1, \dots, m\}$, and consider the set

$$\mathcal{V}_v = \{V : S \subseteq V \subseteq F,\ |V| = s + v\},$$

as well as

$$D_S = \begin{pmatrix} 0 \\ D_S^2 \\ \vdots \\ D_S^B \end{pmatrix} \qquad D_c = \begin{pmatrix} 0 & \dots & 0 \\ D_{c,1}^2 & \dots & D_{c,m}^2 \\ \vdots & & \vdots \\ D_{c,1}^B & \dots & D_{c,m}^B \end{pmatrix}$$

where D_c is the matrix of the m individual statistics D_j^π corresponding to the indices $j \notin S$, sorted in decreasing order within each row:

$$D_{c,1}^\pi \geq \ldots \geq D_{c,m}^\pi \quad \forall\, \pi \in \mathcal{P}.$$

The lower and upper bounds for the distribution of D_V^π are given by the sum of D_S^π and the v smaller and higher remaining test statistics:

$$D_{L(v)}^\pi = D_S^\pi + \sum_{j=m-v+1}^{m} D_{c,j}^\pi \qquad\qquad D_{U(v)}^\pi = D_S^\pi + \sum_{j=1}^{v} D_{c,j}^\pi.$$

Let $c_{L(v)}$ and $c_{U(v)}$ be the $(1-\alpha)$-quantiles of the distribution of the bounds $D_{L(v)}^\pi$ and $D_{U(v)}^\pi$. The bounds for the fixed size v may be checked in the following way:

- if $0 = D_V \leq c_{L(v)}$, then H_S is not rejected;
- if $0 = D_V > c_{U(v)}$, then H_V is rejected for all $V \in \mathcal{V}_v$, and a different size can be checked;
- otherwise, the output is indecisive.

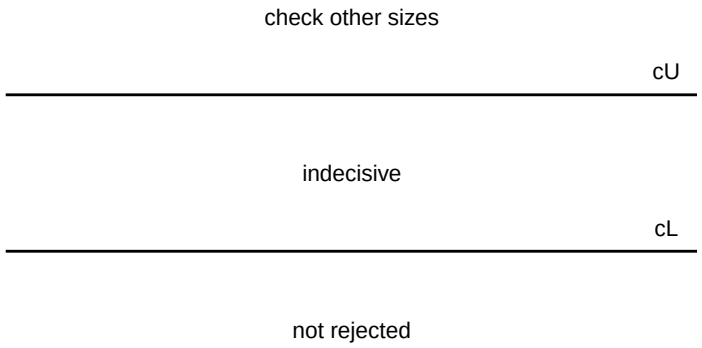

Figure 1: Outcomes of the bounds check according to different positions of the point $D_V = 0$.

Test. All the possible superset sizes $v \in \{0, 1, \ldots, m\}$ are checked with the above-mentioned procedure:

- whenever a size v^* leading to a non-rejection ($c_{L(v^*)} \geq 0$) is found, the algorithm stops and H_S is not rejected;
- if all sizes v are such that the corresponding hypotheses are rejected with certainty ($c_{U(v)} < 0$), H_S is rejected;
- otherwise, the sizes v leading to an indecisive output ($c_{L(v)} < 0 \leq c_{U(v)}$) are examined with a step-by-step method.

The analysis starts with the extremes $v = 0$ and $v = m$, corresponding to S and F. In both cases the output cannot be indecisive, as the lower and upper bounds coincide: $c_{L(0)} = c_{U(0)} = c_0$ and $c_{L(m)} = c_{U(m)} = c_m$. If neither case leads to a non-rejection, the other sizes are checked in

- increasing order $(1, \dots, m-1)$ if $c_m \le c_0 < 0$;
- decreasing order $(m-1, \dots, 1)$ if $c_0 < c_m < 0$.

Indeed, in the first case the rejection for $v = 0$ is less extreme than that for $v = m$; hence we expect that an eventual non-rejection will likely be found with small sizes, when adding one or more indices to S. On the contrary, in the second case we expect that an eventual non-rejection will likely be found when removing one or more indices from F.

Step-by-step method. Let i_1 be the index of the highest observed statistic in $F \setminus S$,

$$T_{i_1} = \max\{T_j \,:\, j \in F \setminus S\},$$

corresponding to the centered statistic $D_{c,1}$. Similarly, let i_n be the index of the n-th highest observed statistic, corresponding to $D_{c,n}$.

For each size v leading to an indecisive outcome, the set $\mathcal{V}_v$ is partitioned in

- $\mathcal{V}_v(-i_1) = \{V \in \mathcal{V}_v \,:\, i_1 \notin V\} = \{V \,:\, S \subseteq V \subseteq F \setminus \{i_1\},\ |V| = v + s\}$
- $\mathcal{V}_v(+i_1) = \{V \in \mathcal{V}_v \,:\, i_1 \in V\} = \{V \,:\, S \cup \{i_1\} \subseteq V \subseteq F,\ |V| = v + s\}$.

We expect that the quantities $D^{\pi}_{i_1} = T^{\pi}_{i_1} - T_{i_1}$ will likely be small, and thus in the analysis of $(-i_1)$ the lower bound will likely increase. In both cases, however, the bounds will never become wider.

The step-by-step method is a depth-first search. First, it explores the cases $(-i_1)$ and $(+i_1)$, and enumerates the eventual indecisive sizes. If a non-rejection is found, it stops and H_S is not rejected. Otherwise it further divides the first case into $(-i_1, -i_2)$ and $(-i_1, +i_2)$. Such procedure is iterated until $(-i_1, \dots, -i_h)$ produces no indecisive sizes; then the algorithm analyzes $(-i_1, \dots, -i_{h-1}, +i_h)$.

The cases to be examined are stacked into a last-in-first-out pile. At each step, the first element is removed, and the two subcases, if having some indecisive sizes, are added to the list:

step 1	$(-i_1)$	$(+i_1)$		
step 2	$(-i_1, -i_2)$	$(-i_1, +i_2)$	$(+i_1)$	
step 3	$(-i_1, -i_2, -i_3)$	$(-i_1, -i_2, +i_3)$	$(-i_1, +i_2)$	$(+i_1)$
...				

2 Algorithm

The algorithm requires

- a $B \times f$ matrix T where the columns correspond to individual hypotheses and are not necessarily ordered, while the rows correspond to random permutations (with $\pi_1 = \text{id}$);
- a subset S of indices under test;
- a significance level α (by default 0.05).

2.1 Main Body

Setting of the Input. The function `perm.setting` sorts the columns of T in increasing order, and then defines the matrix D of centered test statistics, sorted within each row. Another matrix I keeps track of the original indices: if D^{π}_j corresponds to the individual test statistic on index i in the input matrix, then $I_{\pi j} = i$. This first step will be kept apart from the following function, so that the matrices D and I can be defined only once even when different subsets S are checked.

```
# Setting of the input
# It returns the matrix D of the centered statistics (ordered within
# each row), and the matrix I of the indices
perm.setting <- function(T){
        IF NOT(check.matrix(T)){
                STOP with error message
        }

        i.vec <- c(1,...,f)
        T <- T[columns sorted so that T[1,] is in incr. order]
        i.vec <- i.vec[reordered as T[1,]]
        # note: the columns can be sorted with other criteria

        I <- matrix(B rows, each equal to i.vec)
        D <- T[each column is centered with respect to its first el.]
        D <- D[el. are sorted in incr. order within each row
           except the first]
        I <- I[reordered as D]
        # B times f matrices

        RETURN list("D"=D, "I"=I)
}
```

Main Function for Hypothesis Testing. The function `perm.test` checks the null hypothesis on the set of indices S. The inner functions are displayed in the following section.

```
# Hypothesis testing on S
# It returns rejected = TRUE if Hs is rejected
# as well as the number of steps used in the step-by-step method
perm.test(D, I, S, alpha=0.05){
        IF NOT(check.matrix(D) AND check.matrix(I)){
```

```
        STOP with error message
}
B <- nrow(D)
f <- ncol(D)

IF NOT(check.indices(I, B, f)){
        STOP with error message
}

IF NOT(check.hp(S, f)){
        STOP with error message
}
S <- unique(S)
s <- length(S)
m <- f-s

IF NOT(0 < alpha < 1){
        STOP with error message
}

# if m=0 (s=f), we are testing all the indices
IF (m == 0){
        rej <- above.quantile(rowSums(D), alpha)
        RETURN list("rejected"=rej, "bab.steps"=0)
}

# output in case of non-rejection
nr.output <- list("rejected"=FALSE, "bab.steps"=0)

Ds <- D[elements corresponding in I to indices in S]
# D_j^pi is removed if I[pi,j] is in S
Ds <- rowSums(Ds) # vector of length B

Dfull <- rowSums(D) # sum of all test statistics

# v=0 and v=m
extremes <- extremes.check(Ds, Dfull, alpha)
IF NOT(extremes){
        RETURN nr.output
}

compl.indices <- I[1,][- indices in S]
# indices not in S, in order
Dc <- D[-elements corresponding in I to indices in S]
# B times m matrix

middle <- IFELSE(incr.order, incr.check(Ds, Dc, m, alpha),
   decr.check(Dfull, Dc, m, alpha))
IF NOT(middle){
        RETURN nr.output
}


# STEP-BY-STEP METHOD
```

```
        rej <- TRUE
        bab.s <- 0 # number of steps

        list.ind <<- list(indecisive) # indecisive sizes ind.free
        list.level <<- list(0) # numbers n of considered indices
        list.fixed <<- list(Ds) # fixed quantities Dfixed

        WHILE (rej == TRUE AND length(list.ind) > 0){
                rej <- children.nodes(n=list.level[1],
                   Dfixed=list.fixed[1], ind.free=list.ind[1],
                   Dc, compl.indices, m, B, alpha)
                bab.s <- bab.s+1
        }

        RETURN list("rejected"=rej, "bab.steps"=bab.s)
}
```

Condition Check. The following internal functions are used to check the input and some conditions.

```
# TRUE if an object M is a matrix of finite numbers
check.matrix <- function(M){
        out <- (M is a matrix of finite numbers)
        RETURN out
}
```

```
# TRUE if a matrix M is a well-defined matrix of indices
# with given dimensions
check.indices <- function(M, n.row, n.col){
        out <- (nrow(I) == n.row AND ncol(I) == n.col AND
           each row of I contains the integers from 1 to n.col)
        RETURN out
}
```

```
# TRUE if an object X is a vector of integers
# between 1 and a given maximum
check.hp <- function(X, n.max){
        out <- (X is a numerical vector AND
           its elements are integers between 1 and n.max)
}
```

```
# TRUE if zero is greater than the sample quantile of a vector X
above.quantile <- function(X, alpha){
        c <- quantile(X, (1-alpha))
        out <- (sign(c) == -1) # 0 > c
        RETURN out
}
```

Check of Superset Sizes. The following functions check different superset sizes. In particular, `incr.check` and `decr.check` exploit the fact that

$$D_{L(v)} = D_{L(v-1)} + D_{m-v+1} \qquad D_{U(v)} = D_{U(v-1)} + D_v$$
$$D_{L(v)} = D_{L(v+1)} - D_{m-v} \qquad D_{U(v)} = D_{U(v+1)} - D_{v+1}.$$

```
# Check of the cases v=0 and v=m
# FALSE in case of a non-rejection, TRUE otherwise
# It creates incr.order, TRUE if the order to be used for the middle
# sizes is increasing
extremes.check <- function(Ds, Dfull, alpha)
        # v=0
        c0 <- quantile(Ds, (1-alpha))
        # stop (not rejected) if 0 <= c0
        IF (sign(c0) > -1){
                RETURN FALSE
        }

        # v=m
        cfull <- quantile(Dfull, (1-alpha))
        # stop (not rejected) if 0 <= cfull
        IF (sign(cfull) > -1){
                RETURN FALSE
        }

        # if cfull <= c0, sizes will be explored in incr. order
        incr.order <<- IFELSE (cfull <= c0, TRUE, FALSE)
        RETURN TRUE
}
```

```
# Check of sizes v=1,...,m-1 in increasing order
# FALSE in case of a non-rejection, TRUE otherwise
# It creates the vector of indecisive sizes
incr.check <- function(Ds, Dc, m, alpha){
        v <- 1
        Dlow <- Ds
        Dup <- Ds
        indecisive <<- c() # keeps track of indecisive sizes

        WHILE (v < m){
                Dlow <- Dlow + Dc[,m-v+1]
                # stop (not rejected) if 0 <= clow
                IF NOT(above.quantile(Dlow), alpha){
                        RETURN FALSE
                }

                Dup <- Dup + Dc[,v]
                # indecisive if if clow < 0 <= cup
                IF NOT(above.quantile(Dup), alpha){
                        indecisive <<- c(indecisive, v)
                }

                v <- v+1
```

```
        }
        RETURN TRUE
}
```

```
# Check of sizes v=m-1,...,1 in decreasing order
# FALSE in case of a non-rejection, TRUE otherwise
# It creates the vector of indecisive sizes
decr.check <- function(Dfull, Dc, m, alpha){
        v <- m-1
        Dlow <- Dfull
        Dup <- Dfull
        indecisive <<- c() # keeps track of indecisive sizes

        WHILE (v > 0){
                Dlow <- Dlow - Dc[,m-v]
                # stop (not rejected) if 0 <= clow
                IF NOT(above.quantile(Dlow)){
                        RETURN FALSE
                }

                Dup <- Dup - Dc[,v+1]
                # indecisive if if clow < 0 <= cup
                IF NOT(above.quantile(Dup)){
                        indecisive <<- c(indecisive, v)
                }

                v <- v-1
        }
        RETURN TRUE
}
```

2.2 Step-by-Step Method

Enumeration of Subcases. In the list of subcases to be examined, each element is characterized by

- n, the total number of indices $i_1, \ldots, i_n$ that have been considered (removed or kept);
- D_{fixed}, the sum of D_S and the test statistics that have been kept;
- $ind.free$, the vector of the indecisive sizes minus the number of kept statistics.

At every step, the function `children.nodes` is used to delete the first element and add its two subcases derived by removing and keeping i_n, if they correspond to at least one indecisive size. Let R and K denote the distributions of the bounds when i_n is removed and kept, respectively.

The statistics corresponding to indices $i_1, \ldots, i_n$ are removed from D_c to define $\tilde{D}_c$. Moreover, $\tilde{D}^{\pi}_{\text{fixed}}$ is defined as the sum of D^{π}_{fixed} and the centered statistic corresponding to i_n. The indecisive sizes are explored following the previously-defined order.

```
# Analysis of the subcases obtained by removing/keeping i_n
# FALSE in case of a non-rejection, TRUE otherwise
```

```
children.nodes <- function(n, Dfixed, ind.free, Dc, compl.indices,
   m, B, alpha){
        delete.list() # the first element is removed from the list

        # the first n highest statistics are removed from Dc
        Dc.tilde <- Dc[-el. corresponding to indices in
           compl.indices[1:n] in I]
        Dfixed.tilde <- Dfixed + Dc[el. corresponding to index
           compl.indices[n] in I]
        m.tilde <- m - n # dim. of Dc.tilde

        L <- length(ind.free)
        i <- 1
        ind.remove <<- c() # indecisive free sizes after removing
        ind.keep <<- c() # indecisive free sizes after keeping

        IF (incr.order){
                incr.bab.set(Dfixed, Dfixed.tilde)
                WHILE (i <= L){
                        v <- ind.free[i]
                        current.step <- incr.bab.loop(z, v, Dc.tilde,
                           m.tilde, B, alpha)
                        # stop if one non-rejection is found
                        IF NOT(current.step){
                                RETURN FALSE
                        }
                        z <- v
                        i <- i+1

                }
        }
        ELSE{
                decr.bab.set(Dfixed, Dfixed.tilde)
                WHILE (i <= L){
                        v <- ind.free[i]
                        current.step <- decr.bab.loop(z, v, Dc.tilde,
                           m.tilde, B, alpha)
                        # stop if one non-rejection is found
                        IF NOT(current.step){
                                RETURN FALSE
                        }
                        z <- v
                        i <- i+1
                }
        }

        add.list(ind.keep, n+1, Dfixed.tilde)
        add.list(ind.remove, n+1, Dfixed)

        RETURN TRUE
}
```

Operations on the List. The following internal functions are used to delete or add elements from the list of cases to be examined.

```
# It removes the first element of the lists
# list.ind, list.level and list.fixed
delete.list <- function(){
        list.ind <<- list.ind[-1]
        list.level <<- list.level[-1]
        list.fixed <<- list.fixed[-1]
}
```

```
# If a vector X is non-empty
# it is added to the list list.ind
# and the quantities lev and fixed are added to the lists
# list.level and list.fixed, respectively
add.list <- function(X, lev, fixed){
        IF (length(X) > 0){
                list.ind <<- c(X, list.ind)
                list.level <<- c(lev, list.level)
                list.fixed <<- c(fixed, list.fixed)
        }
}
```

Check in Increasing Order. The following functions are used to check sizes in increasing order, using the fact that

$$R^\pi_{L(v)} = D^\pi_{\text{fixed}} + \tilde{D}^\pi_{c,m-n-v+1} + A^\pi_{L(v)} \qquad K^\pi_{L(v)} = \tilde{D}^\pi_{\text{fixed}} + A^\pi_{L(v)}$$
$$R^\pi_{U(v)} = D^\pi_{\text{fixed}} + \tilde{D}^\pi_{c,v} + A^\pi_{U(v)} \qquad K^\pi_{U(v)} = \tilde{D}^\pi_{\text{fixed}} + A^\pi_{U(v)}$$

where

$$A^\pi_{L(v)} = \sum_{j=m-n-v+2}^{m-n} \tilde{D}^\pi_{c,j} \qquad A^\pi_{U(v)} = \sum_{j=1}^{v-1} \tilde{D}^\pi_{c,j}.$$

```
# Initialization of the loop for the increasing case
incr.bab.set <- function(Dfixed, Dfixed.tilde){
        z <<- 1 # previous size

        # bounds when removing (R) or keeping (K)
        Rlow <<- Dfixed
        Rup <<- Rlow
        Klow <<- Dfixed.tilde
        Kup <<- Klow
}
```

```
# Check of a given size v
# when sizes are explored in increasing order
# FALSE in case of a non-rejection, TRUE otherwise
# It updates the vectors of indecisive sizes
incr.bab.loop <- function(z, v, Dc.tilde, m.tilde, B, alpha){
        cond <- (v == 1) # if v=1, then v-1 < z
```

```
        Alow <- IFELSE(cond, rep(0,B),
           rowSums(Dc.tilde[,((m.tilde-v+2):(m.tilde-z+1)])
        Rlow <<- Rlow + Dc.tilde[,m.tilde-v+1] + Alow
        IF NOT(above.quantile(Rlow), alpha){
                RETURN FALSE
        }

        Klow <<- Klow + Alow
        IF NOT(above.quantile(Klow), alpha){
                RETURN FALSE
        }

        Aup <- IFELSE(cond, rep(0,B),
           rowSums(Dc.tilde[,z:(v-1)])
        Rup <<- Rup + Dc.tilde[,v] + Aup
        IF NOT(above.quantile(Rup), alpha){
                ind.remove <<- c(ind.remove, v)
        }

        Kup <<- Kup + Aup
        IF NOT(above.quantile(Kup), alpha){
                ind.keep <<- c(ind.keep, v-1)
        }
        RETURN TRUE
}
```

Check in Decreasing Order. The following functions are used to check sizes in decreasing order, using the fact that

$$R^\pi_{L(v)} = D^\pi_{\text{fixed}} + \sum_{j=1}^{m-n} \tilde{D}^\pi_{c,j} - A^\pi_{L(v)} \qquad K^\pi_{L(v)} = \tilde{D}^\pi_{\text{fixed}} + \sum_{j=1}^{m-n} \tilde{D}^\pi_{c,j} - \tilde{D}^\pi_{c,m-n-v+1} - A^\pi_{L(v)}$$

$$R^\pi_{U(v)} = D^\pi_{\text{fixed}} + \sum_{j=1}^{m-n} \tilde{D}^\pi_{c,j} - A^\pi_{U(v)} \qquad K^\pi_{U(v)} = \tilde{D}^\pi_{\text{fixed}} + \sum_{j=1}^{m-n} \tilde{D}^\pi_{c,j} - \tilde{D}^\pi_{c,v} - A^\pi_{L(v)}$$

where

$$A^\pi_{L(v)} = \sum_{j=1}^{m-n-v} \tilde{D}^\pi_{c,j} \qquad A^\pi_{U(v)} = \sum_{j=v+1}^{m-n} \tilde{D}^\pi_{c,j}.$$

```
# Initialization of the loop for the decreasing case
decr.bab.set <- function(Dfixed, Dfixed.tilde, Dc.tilde, m.tilde){
        z <<- m.tilde # previous size
        Dtot <<- rowSums(Dc.tilde)

        # bounds when removing (R) or keeping (K)
        Rlow <<- Dfixed + Dtot
        Rup <<- Rlow
        Klow <<- Dfixed.tilde + Dtot
        Kup <<- Klow
}
```

```
# Check of a given size v
# when sizes are explored in decreasing order
# FALSE in case of a non-rejection, TRUE otherwise
# It updates the vectors of indecisive sizes
decr.bab.loop <- function(z, v, Dc.tilde, m.tilde, B, alpha){
        cond <- (v == m.tilde) # if v=m.tilde, then v+1 > z
        Alow <- IFELSE(cond, rep(0,B),
           rowSums(Dc.tilde[,((m.tilde-z+1):(m.tilde-v)])
        Rlow <<- Rlow - Alow
        IF NOT(above.quantile(Rlow), alpha){
                RETURN FALSE
        }

        Klow <<- Klow - Dc.tilde[,m.tilde-v+1] - Alow
        IF NOT(above.quantile(Klow), alpha){
                RETURN FALSE
        }

        Aup <- IFELSE(cond, rep(0,B),
           rowSums(Dc.tilde[,(v+1):z])
        Rup <<- Rup - Aup
        IF NOT(above.quantile(Rup)){
                ind.remove <<- c(ind.remove, v)
        }

        Kup <<- Kup - Dc.tilde[,v] - Aup
        IF NOT(above.quantile(Kup), alpha){
                ind.keep <<- c(ind.keep, v-1)
        }
        RETURN TRUE
}
```