OPERATING-CHARACTERISTIC GUIDED DESIGN OF GROUP-SEQUENTIAL TRIALS

ANIKO SZABO

Abstract. Group-sequential designs are commonly used for clinical trials to allow early stopping for efficacy or futility. While the design of a single-stage randomized trial is guided by a target power for an alternative hypothesis of interest, the addition of interim analyses is driven by technical choices that are less understandable for clinicians. For example, the commonly used Lan-DeMets methodology requires specification of the timing of analyses and error spending functions. Since the rationale and effect of these technical choices is often unclear, the operating characteristics of the final design are explored under various values of the parameter of interest, and the design is then adjusted until desired properties are obtained.

In this work we develop methods for constructing designs that achieve the desired operating characteristics without the need to specify error spending functions or the timing of analyses. Specifically, we consider designing a study for the mean difference δ of a normally distributed outcome with known variance. The null hypothesis $H_0 : \delta = \delta_0$ is tested versus $H_a : \delta = \delta_A$, with power π at a significance level α. The interim analyses are designed so that for a pre-specified sequence δ_{Ak} the study stops for efficacy at stage k with probability π if $\delta = \delta_{Ak}$. If stopping for futility is also considered, then the requirement to stop for futility at stage k with probability π_F if $\delta = \delta_{0k}$ for pre-specified sequence δ_{0k} can also be added. We show that under some monotonicity restrictions, such designs exist for any choice of the timing of interim analyses. Specific designs can be selected by imposing additional optimality requirements, such as minimizing the expected sample size under the target alternative δ_A, or the average sample size under a weighted selection of the alternatives.

Date: July 19, 2020.

1. Main function: calculate design

"../R/gsDesignOC.R" 3≡

```
#'Find optimal group-sequential design
#'
#'The \code{gsDesignOC} function finds the analysis times, boundaries, and sample size
#'for a group-sequential trial
#'
#'@export
#'@param thA numeric; effect size under the alternative hypothesis
#'@param thA.seq monotone numeric vector of values getting closer to thA from outside the
#'th0-thA range (ie, if thA > th0, they should form a decreasing sequence with all values > thA). #'For t
#'@param th0 numeric; effect size under the null hypothesis
#'@param th0.seq monotone numeric vector of values getting closer to th0 from outside the
#'th0-thA range (ie, if thA > th0, they should form an increasing sequence with all values < th0).
#'The study will stop for futility at or before the k-th stage with probability \code{power.futility}.
#'Its length should be equal to that of \code{thA.seq}. The default value of \code{NULL} implies no
#'futility monitoring.
#'@param min.under character string, one of \code{c("alt","null","alt.mixture", "null.mixture")}
#'or their unique abbreviations. Specifies the hypothesis under which the study sample size is
#'minimized. \code{min.under="alt"} minimizes under the alternative hypothesis (theta=\code{thA}),
#'\code{min.under="null"} under the null hypothesis (theta=\code{th0}), while the \code{"mixture"}
#'option minimize the weighted average of sample sizes under theta=\code{thA.seq} or theta=\code{th0.seq}
#'The weights are specified in \code{mix.w}.
#'@param sig.level numeric; the one-sided significance level. The default is 0.025.
#'@param sig.level_final numeric; the desired nominal significance level for testing the null
#'hypothesis at the final stage. Should be between 0 and \code{sig.level}. If NULL, the value
#'will be found using the optimization criterion.
#'@param power numeric; the desired study power. The default is 0.9. This value will also be
#'used to set the probability of stopping for efficacy at stage k under \code{thA.seq}.
#'@param power.futility numeric; the probability of stopping for futility at stage k under \code{th0.seq}
#'@param futility.type character string, one of \code{c("non-binding","binding")} or their
#'unique abbreviations. Specifies whether the effect of the futility boundary should be included
#'in the efficacy power/type I error calculations ("binding"), or not ("non-binding").
#'@param mix.w numeric vector of length equal to that of \code{thA.seq}. The weights of the
#'elements of \code{thA.seq} or \code{th0.seq} in the optimality criterion when using \code{min.under="al
#'or \code{min.under="null.mixture"}, respectively. It will be normalized to a sum of 1.
#'Defaults to equal weights.
#'@param control list of parameters controlling the estimation altorithm to replace the default
#'values returned by the \code{glsControl} function.
#'@return an object of class \code{gsDesignOC}
#'@author Aniko Szabo
#'@references Szabo, A, Tarima, S (?) Operating-characteristic guided design of group-sequential trials.
#'@keywords nonparametric
#'@examples
#'
#'gsDesignOC(0.3, thA.seq = c(1, 0.5), min.under="alt")
#'
#'@name gsDesignOC

gsDesignOC <- function(thA, thA.seq, th0=0, th0.seq=NULL,
                       min.under=c("alt","null","alt.mixture", "null.mixture"),
                       sig.level = 0.025, sig.level_final=NULL,
                       power=0.9, power.futility = power,
                       futility.type=c("non-binding","binding"),
                       mix.w = rep(1, length(thA.seq)),
                       control=list()){
  ⟨ Check inputs ? ⟩
  ⟨ Define optimization function ? ⟩

  k <- length(thA.seq) + 1
  n.guess <- ztest.n(delta=c(thA.seq,thA)-th0, sd=1,
                     sig.level=sig.level, power = power)
  if (is.null(sig.level_final)){
    thA.seq <- c(thA.seq, thA)
  }
```

⟨ *Check inputs ?* ⟩ ≡

```
if (any(diff(c(thA.seq, thA)) * sign(thA-th0) <= 0))
  stop("'thA.seq' should approach thA in a monotone sequence outside the th0-thA range")

if (!is.null(th0.seq)){
  if (length(th0.seq) != length(thA.seq))
    stop("If specified, 'th0.seq' should have the same length as 'thA.seq'")
  if (any(diff(c(th0.seq, th0)) * sign(th0-thA) <= 0))
    stop("'th0.seq' should approach th0 in a monotone sequence outside the th0-thA range")
}

min.under <- match.arg(min.under)
if (min.under == "alt.mixture" & length(mix.w) != length(thA.seq))
  stop("'mix.w' should have the same length as 'thA.seq' when optimizing under the mixture of alternati
if (min.under == "null.mixture"){
  if (is.null(th0.seq))
    stop("'th0.seq' has to be defined when optimizing under the mixture of nulls")
  if (length(mix.w) != length(th0.seq))
    stop("'mix.w' should have the same length as 'th0.seq' when optimizing under the mixture of nulls")
}

if (!is.null(sig.level_final) & ((sig.level_final <= 0 | sig.level_final > sig.level)))
  stop("'sig.level_final' should be between 0 and 'sig.level'")

futility.type <- match.arg(futility.type)

controlvals <- ocControl()
if (!missing(control))
  controlvals[names(control)] <- control
```

◇

Fragment referenced in 3.

⟨ *Define optimization function ?* ⟩ ≡

```
# n.vec is stage-specific sample sizes
.cp <- function(n.vec){

  n <- sum(n.vec)
  n.I <- cumsum(n.vec/n)
  if (!is.null(sig.level_final)) n.I <- head(n.I, -1)

  dp <- design.properties(n=n, n.I = n.I, thA.seq = thA.seq, th0.seq = th0.seq,
                               th0=th0, thA=thA, sig.level_final = sig.level_final,
                               power=power)
  Q <-
    optim.penalty*abs(dp$typeI - sig.level)/(sig.level*(1-sig.level)) +
    optim.penalty*abs(dp$pow - power)/(power*(1-power)) +
    dp$enA
  #n
  Q
}
```

◇

Fragment referenced in 3.

2. Utility help-functions

"../R/Utility.R" ?≡

```
#'One-sample Z-based sample size for one-sided test
#'
#'The \code{ztest.n} function finds the sample size of a one-sided Z-test with given power
#'and significance level
#'
#'@export
#'@param delta numeric; targeted effect size
#'@param sd numeric; standard deviation, assumed known
#'@param sig.level numeric; one-sided significance level
#'@param power numeric; target power

ztest.n <- function(delta, sd, sig.level, power){
  za <- qnorm(sig.level, lower=FALSE)
  zb <- qnorm(power)
  n <- (za+zb)^2 * sd^2/delta^2
  n
}
```