Phase II-III Seamless Design Software

Tze Leung Lai Philip Lavori Mei-Chiung Shih Pei He
Balasubramanian Narasimhan
Stanford University
Stanford, CA 94305

June 25, 2014

Contents

1 Introduction

This is a companion program to the paper[1] that describes a seamless Phase II-III design. It describes the actual implementation of the code that is used in the R package.

The following notation is used throughout.

- Y_i will be an indicator of response ($Y_i = 1$) or no response ($Y_i = 0$) for the i-th patient

- Z_i will be a (random) assignment of patient i to treatment ($Z_i = 1$) or control ($Z_i = 0$)
- $\pi_0 = P(Y_i = 1|Z_i = 0)$ is the probability of a response in the control arm and $\pi_1 = P(Y_i = 1|Z_i = 1)$ is the probability of a response in the treatment arm; $\boldsymbol{\pi} = (\pi_0, \pi_1)$.
- $\hat{\pi}_{t,0}$ and $\hat{\pi}_{t,1}$ are the estimates of π_0 and π_1 at interim time t respectively
- The null hypothesis is decomposed as

$$H_0^R : \pi_0 \geq \pi_1, \quad H_0^S : \pi_0 < \pi_1 \text{ and } d(\boldsymbol{\pi}, e^\alpha, e^\beta, e^\gamma) \leq 0 \tag{1}$$

  where $d(\boldsymbol{\pi}, e^\alpha, e^\beta, e^\gamma) = \{\pi_0 e^\alpha + (1 - \pi_0)\} - \{\pi_1 e^{\alpha+\beta+\gamma} + (1 - \pi_1)e^\beta\}$
- $\boldsymbol{\theta} = (\alpha, \beta, \gamma)$, $a = e^\alpha$, $b = e^\beta$ and c $= e^\gamma$, $\boldsymbol{\xi} = (a, b, c)$
- $t_j, j = 1, \ldots, k$ are the calendar times of the interim looks
- b_R, $\tilde{b_R}$, c_R, $\tilde{c_R}$ are the boundaries for stopping regions for testing H_0^R and b_S, $\tilde{b_S}$, c_S, $\tilde{c_S}$ are the boundaries of the stopping regions for testing H_0^S. The tilde versions are the futility boundaries and the others are efficacy boundaries

Some preliminaries for better output:

2 Utility Functions

We describe some utility functions that are useful for simulations. All of our simulations generate the entire trial data in one fell swoop. Given this entire data set, we also need the ability to generate an interim dataset at any given point in time. Hence the functions below.

2.1 Generating Clinical Trial Data

We assume there are two arms in the trial, say, *control* and *treatment*. We need to generate data both for the phase II short-term endpoint, response, and the phase III longer term endpoint, survival. The former is merely a coin flip with specified probabilities but the latter is a generated via a proportional hazard rate model described in [1]:

$$\lambda(t \mid Y, Z) = \lambda_0(t) \exp(\alpha Y + \beta Z + \gamma Y Z). \tag{2}$$

Here, Y is the response and Z is the indicator of treatment as above, λ_0 the baseline hazard-rate, and $\boldsymbol{\theta} = (\alpha, \beta, \gamma)$ are the regression parameters.

In addition to π_0, π_1, we'll need several other input parameters need to simulate data from a typical clinical trial that is the subject of the paper:

- the number of years of recruitment, and the number recruited in each of those years. The total number of patients in the trial would be the sum of these numbers
- the number of years of follow-up
- the baseline hazard rate λ_0

- the model parameter values $\boldsymbol{\theta} = (\alpha, \beta, \gamma)$; we'll use an *exponential* distribution for now
- a block size, for randomizing subjects to control or treatment using block randomization

Below is *one* implementation that will generate data with the arrival or entry times of the patients *uniformly* distributed in each year.

```
generateClinicalTrialData <- function(nRec, nFUp, pi0, pi1, theta,
    lambda0, blockSize = 10) {
    ## nRec is the number of patients recruited every year.
    ## Length(nRec) is the number of years of recruitment nFUp is
    ## the number of additional years of followup pi0 the
    ## probability of response under control arm pi1 the
    ## probability of response under treatment arm theta the three
    ## dimensional parameter (alpha, beta, gamma) of the joint
    ## response/survival model lambda0 the baseline hazard rate
    ## blockSize the size of the blocks for randomization of the
    ## treatment/control Returns a data frame with rows in order
    ## of patient entry

    N <- sum(nRec)  # Total number recruited
    ## Check that blockSize is even and that blockSize divides N
    nBlocks <- N/blockSize
    if (trunc(blockSize/2) * 2 != blockSize || trunc(nBlocks) *
        blockSize != N) {
        stop("Improper blocksize or number of subjects recruited")
    }

    ## Generate uniform entry times in each recruitment year
    entryTime <- sort(unlist(lapply(1:length(nRec), function(j) runif(nRec[j],
        j - 1, j))))

    ## Generate treatment indicators (1 = treatment, 0 = control)
    treatmentIndicator <- unlist(lapply(1:nBlocks, function(x) sample(c(rep(0,
        blockSize/2), rep(1, blockSize/2)))))

    ## Generate Bernoulli responses with appropriate probabilities
    responseIndicator <- integer(N)
    responseIndicator[treatmentIndicator == 1] <- rbinom(sum(treatmentIndicator),
        1, pi1)
    responseIndicator[treatmentIndicator == 0] <- rbinom(N -
        sum(treatmentIndicator), 1, pi0)

    ## Compute hazard rate per model
    rate = lambda0 * exp(theta$alpha * responseIndicator + theta$beta *
```

```
        treatmentIndicator + theta$gamma * responseIndicator *
        treatmentIndicator)
    ## Generate time to event
    timeToEvent <- rexp(N, rate = rate)

    ## Construct data frame of result, name variables
    ## appropriately
    result <- data.frame(entryTime = entryTime, responseIndicator = responseIndicator,
        treatmentIndicator = treatmentIndicator, timeToEvent = timeToEvent)
    rownames(result) <- paste("Pat", 1:N, sep = ".")
    result
}
```

The ouptut of this function is a a data frame with variables named `entryTime` (η_i), `responseIndicator` (Y_i), `treatmentIndicator` (Z_i) , and `timeToEvent` (T_i). The rows of the data frame will be labelled by the patient identifier.

2.2 Generating Interim Data

To perform interim looks, we need to be able to generate interim data at any given point in time. In the field, this is the data we have at a particular point in time, but here we are assuming we know the state of nature and that the entire data for the clinical trial is known to us. So we have to be able to generate the interim data from the full data given some parameters. The following inputs are needed for this function:

- the complete dataset, from which the interim dataset will be computed
- The time at which the interim dataset is to be computed
- The administrative censoring time; this allows us to ensure that the interim dataset can be computed for any time correctly

The observed time for subject i is $T_i(t) = \min\{T_i, \xi_i, (t-\eta_i)^+\}$ where the administrative censoring part is $(t-\eta_i)^+$, the difference between calendar time t of patient at interim analysis and the entering time η_i. The patient may also withdraw at time ξ_i, although we ignore this in the current version of this program. The response and treatment censoring are denoted in the paper as $Y_i I_{\{\eta_i \le t\}}, Z_i I_{\{\eta_i \le t\}}$. The indicator of censoring $\delta_i(t)$ is equal to $I_{\{T_i(t)=T_i\}}$.

```
generateInterimData <- function(clinicalTrialDF, interimTime,
    administrativeCensoringTime) {
    ## Given a clinical trial data frame (with variables
    ## entryTime, responseIndicator, treatmentIndicator,
    ## timeToEvent) and a calendar interimTime, generate a data
    ## frame that represents the interim data at the interimTime.
    ## administrativeCensoringTime is the time when the study ends
```

```
    ## Return data frame with timeToEvent calculated as if we at
    ## interimTime, and add a variable called eventTime, which is
    ## calendar time of event, that is, timeToEvent + entryTime
    ## Narrow to data of interest d <-
    ## clinicalTrialDF[clinicalTrialDF£entryTime <= interimTime, ]
    ## N <- nrow(d)
    d <- clinicalTrialDF
    ## perform a parallel minimum
    interimTimeToEvent <- pmin(d$timeToEvent, administrativeCensoringTime,
        pmax(0, interimTime - d$entryTime))  ## T_i(t)
    d$delta <- ifelse(interimTimeToEvent == d$timeToEvent, 1,
        0)  ## delta_i(t)
    d$timeToEvent <- interimTimeToEvent
    d$eventTime <- d$timeToEvent + d$entryTime  ## calendar event time
    d[d$timeToEvent > 0, ]  ##restrict to patients seen up to time t
}
```

Thus, the result will be another data frame with a subset of all the patients and the appropriate times to event, i.e., `interimTimeToEvent` ($T_i(t)$), `responseIndicator` (Y_i), `treatmentIndicator` (Z_i) and `delta` ($\delta_i(t)$).

2.3 Auxiliary Functions

In what follows, it is efficient to pre-compute some quantities from interim data that are needed throughout. Here's a function that computes some summary quantities for the response end-point; this is heavily used by the likelihood functions. The following function returns

`m0` number on control arm

`m1` number on treatment arm

`y0` number of responses in control

`y1` number of responses in treatment

`numberOfTotalResponses` number of total responses in both arms

`controlRespProp` the proportion of responders in control arm

`treatmentRespProp` the proportion of responders in the treatment arm

`pooledProp` the pooled response proportion

```
computeResponseSummary <- function(interimData) {
    m = nrow(interimData)
    interimNumberOfEvents = sum(interimData$delta)
    control <- which(interimData$treatmentIndicator == 0)  ## indices of those on contr
    treatment <- which(interimData$treatmentIndicator == 1)  ## indices of those on tre
    m0 <- length(control)  ## number on control
    m1 <- length(treatment)  ## number on treatment
    y0 <- sum(interimData$responseIndicator[control])  ## number of responses in contro
    y1 <- sum(interimData$responseIndicator[treatment])  ## number of responses in trea
    c(m = m, m0 = m0, y0 = y0, m1 = m1, y1 = y1, numberOfTotalResponses = y0 +
        y1, controlRespProp = y0/m0, treatmentRespProp = y1/m1,
        pooledProp = (y0 + y1)/(m0 + m1))
}
```

We'll need to optimize some likelihood functions under constraints. For example, equation 6 in the paper is a partial likelihood, constrained to the manifold (equation 6 of the paper)

$$d(\boldsymbol{\pi}, \boldsymbol{\xi}) = \{\pi_0 a + (1 - \pi_0)\} - \{\pi_1 abc + (1 - \pi_1)b\} = \eta. \tag{3}$$

with $\eta \geq 0$ is a specified constant value for d. We can reduce the dimensionality of the functio by solving for c given the other two parameters:

$$c = \frac{\pi_0 a + (1 - \pi_0) - \eta - (1 - \pi_1)b}{\pi_1 ab} \tag{4}$$

Two useful functions, one for solving for c and the other for computing d.

```
solveForCGivenABD <- function(piVec, a, b, d) {
    pi0 <- piVec[1]
    pi1 <- piVec[2]
    ((pi0 * a + 1 - pi0) - (1 - pi1) * b - d)/(pi1 * a * b)
}
```

```
computeDGivenXi <- function(piVec, xiVec) {
    pi0 <- piVec[1]
    pi1 <- piVec[2]
    (pi0 * xiVec[1] + 1 - pi0) - (pi1 * xiVec[1] * xiVec[3] +
        1 - pi1) * xiVec[2]
}
```

3 Likelihood Functions

There are two components of the null hypothesis:

$$H_0^R : \pi_0 \geq \pi_1, \quad \text{or} \quad H_0^S : \pi_0 < \pi_1 \text{ and } d(\boldsymbol{\pi}, e^\alpha, e^\beta, e^\gamma) \leq 0 \tag{5}$$

In all these tests, the likelihood ratio statistics of Lai and Shih[2] are used in the testing. The GLR statistics defined in equation 15 of the paper[1], reproduced here, are

$$\Lambda_t^{(1)} = l_t^{(1)}(\hat{\boldsymbol{\pi}}_t) - \sup_{\boldsymbol{\pi}:\pi_0=\pi_1} l_t^{(1)}(\boldsymbol{\pi}), \quad \Lambda_{t,\delta}^{(1)} = l_t^{(1)}(\hat{\boldsymbol{\pi}}_t) - \sup_{\boldsymbol{\pi}:\pi_1-\pi_0=\delta} l_t^{(1)}(\boldsymbol{\pi}), \tag{6}$$

We need to define these likelihood functions for both components of H_0. The log likelihood of the observed responses Y_i is

$$l_t^{(1)}(\boldsymbol{\pi}) = \sum_{i=1}^{n} 1_{\{\eta_i \le t\}} \Big\{ Y_i \log \pi_{Z_i} + (1 - Y_i) \log(1 - \pi_{Z_i}) \Big\}. \tag{7}$$

A function to implement equation 7, where we assume `piVec` holds $\boldsymbol{\pi}$ and `respSummary` holds

- `m0, m1` the number of patients on control and treatment arms respectively
- `y0, y1` the number of responses seen in the control and treatment arms respectively.

We guard against some boundary conditions in the implementation below (thanks to Richard M. Heiberger).

```
loglik1 <- function(piVec, respSummary) {
    y0 <- respSummary["y0"]
    y1 <- respSummary["y1"]
    m0 <- respSummary["m0"]
    m1 <- respSummary["m1"]
    pi0 <- piVec[1]
    pi1 <- piVec[2]

    if (((pi0 <= 0) && (y0 == 0)) || ((m0 == y0) && (pi0 >= 1))) {
        term1PlusTerm2 <- 0
    } else {
        term1PlusTerm2 <- y0 * log(pi0) + (m0 - y0) * log(1 -
            pi0)
    }

    if (((pi1 <= 0) && (y1 == 0)) || ((m1 == y1) && (pi1 >= 1))) {
        term3PlusTerm4 <- 0
    } else {
        term3PlusTerm4 <- y1 * log(pi1) + (m1 - y1) * log(1 -
            pi1)
    }
    term1PlusTerm2 + term3PlusTerm4
}
```

A single function covers both the cases of where the likelihoods are computed over the path $\pi_0 = \pi_1 + \delta$, $\delta \ge 0$. Note that we flip the sign below because we want do a maximization.

```
loglik1GivenDelta <- function(p, respSummary, delta = 0) {
    y0 <- respSummary["y0"]
    y1 <- respSummary["y1"]
    m0 <- respSummary["m0"]
    m1 <- respSummary["m1"]
    p1 <- p + delta

    if (((p <= 0) && (y0 == 0)) || ((m0 == y0) && (p >= 1))) {
        term1PlusTerm2 <- 0
    } else {
        term1PlusTerm2 <- -y0 * log(p) - (m0 - y0) * log(1 -
            p)
    }

    if (((p1 <= 0) && (y1 == 0)) || ((m1 == y1) && (p1 >= 1))) {
        term3PlusTerm4 <- 0
    } else {
        term3PlusTerm4 <- -y1 * log(p1) - (m1 - y1) * log(1 -
            p1)
    }
    term1PlusTerm2 + term3PlusTerm4
}
```

For testing the second component H_0^S, we have $\Lambda_t^{(2)} = l_t^{(2)}(\hat{\boldsymbol{\theta}}_t) - \sup_{\boldsymbol{\theta}: d(\hat{\boldsymbol{\pi}}_t, e^\alpha, e^\beta, e^\gamma)=0} l_t^{(2)}(\boldsymbol{\theta})$. The first component is the (conditional) log partial likelihood of the observed failure-time data given the observed responses:

$$l_t^{(2)}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \delta_i(t) \Big\{ \boldsymbol{\theta}^T \mathbf{W}_i - \log \Big(\sum_{j \in R_i(t)} e^{\boldsymbol{\theta}^T \mathbf{W}_j} \Big) \Big\}, \tag{8}$$

in which $R_i(t) = \{j : T_j(t) \geq T_i(t)\}$ is the "risk set" consisting of subjects still "at risk" (i.e., not having failed nor been censored) at calendar time t. The following function defines this partial likelihood given the parameter `aa` (i.e. $\theta = (\alpha, \beta, \gamma)$) and the the data frame `interimData`, *ordered by the time to event.*

```
loglik2 <- function(theta, interimData) {
    w <- theta[1] * interimData$responseIndicator + theta[2] *
        interimData$treatmentIndicator + theta[3] * interimData$responseIndicator *
        interimData$treatmentIndicator  ## alpha*Y + beta * Z + gamma * Y * Z
    w1 = rev(cumsum(rev(exp(w))))
    sum(interimData$delta * (w - log(w1)))
}
```

The second is the constrained maximized partial likelihood, that is the same likelihood just above but constrained to the manifold specified in equation 3 above, with $\eta = 0$ here.

So we can code this constrained partial likelihood, setting $\eta = 0$ as the default value, as follows:

```
loglik2.repar0 <- function(xi, interimData, pi0, pi1, eta.hyp = 0) {
    theta <- c(log(xi[1]), log(xi[2]), log((pi0 * xi[1] + 1 -
        pi0 - (1 - pi1) * xi[2] - eta.hyp)/(pi1 * xi[1] * xi[2])))
    w <- theta[1] * interimData$responseIndicator + theta[2] *
        interimData$treatmentIndicator + theta[3] * interimData$responseIndicator *
        interimData$treatmentIndicator  ## alpha*Y + beta * Z + gamma * Y * Z
    w1 = rev(cumsum(rev(exp(w))))
    sum(interimData$delta * (w - log(w1)))
}
```

During the interim look, we need to also check whether to stop for futility or not. The futility stopping can occur at calendar time $t = t_j$ $(j < k)$ if

$$\hat{\pi}_{t,1} < \hat{\pi}_{t,0} + \delta \quad \text{and} \quad \Lambda^{(1)}_{t,\delta} \geq \tilde{b}_R, \tag{9}$$

$$\text{or} \quad d(\hat{\boldsymbol{\pi}}_t, e^{\hat{\alpha}_t}, e^{\hat{\beta}_t}, e^{\hat{\gamma}_t}) < \eta \quad \text{and} \quad \Lambda^{(2)}_{t,\eta} \geq \tilde{b}_S. \tag{10}$$

where $\Lambda^{(1)}_{t,\delta} = l^{(1)}_t(\hat{\boldsymbol{\pi}}_t) - \sup_{\boldsymbol{\pi}:\pi_1-\pi_0=\delta} l^{(1)}_t(\boldsymbol{\pi})$, and $\Lambda^{(2)}_{t,\eta} = l^{(2)}_t(\hat{\boldsymbol{\theta}}_t) - \sup_{\boldsymbol{\theta}:\, d(\hat{\boldsymbol{\pi}}_t, e^{\alpha}, e^{\beta}, e^{\gamma})=\eta} l^{(2)}_t(\boldsymbol{\theta})$.

Note that functions for computing the likelihoods in $\Lambda^{(1)}_{t,\delta}$ are already available (`loglik1` and `loglik1GivenDelta`). Also, $\Lambda^{(2)}_{t,\eta}$ can be computed using `loglik2.repar0` by specifying a non-zero `eta.hyp`.

4 Stopping Boundaries

We allow for early stopping due to futility or efficacy. Specifically, to test H^R_0, we use the group sequential GLR tests introduced by Lai and Shih [2], who call these tests "modified Haybittle-Peto tests" and have established their asymptotic optimality. Let $l^{(1)}_t(\boldsymbol{\pi})$ be as defined in (7) and $\hat{\boldsymbol{\pi}}_t$ be the maximum likelihood estimator of $\boldsymbol{\pi}$ at calendar time t. The GLR statistic for testing $\pi_0 = \pi_1$ is $\Lambda^{(1)}_t$, but $\Lambda^{(1)}_{t,\delta}$ is the GLR statistic for testing the alternative hypothesis $\pi_1 = \pi_0 + \delta$, with $\delta > 0$ chosen to denote clinically significant alternatives, which will be used to guide futility stopping for the response endpoint.

The stopping region of the group sequential GLR test of H^R_0 is the following:

$$\hat{\pi}_{t,0} < \hat{\pi}_{t,1}, \quad \text{and} \quad \Lambda^{(1)}_t \geq b_R \quad \text{for} \quad t = t_j \ (1 \leq j < k) \quad \text{or} \quad \Lambda^{(1)}_t \geq c_R \quad \text{for} \quad t = t_k. \tag{11}$$

Under H^R_0,

$$\text{sgn}(\hat{\pi_{t,1}} - \hat{\pi_{t,0}})\sqrt{2n_t\Lambda^{(1)}_t} \xrightarrow{asymptotically} N(0, n_t) \tag{12}$$

It is also the case that under H^S_0,

$$\text{sgn}(d(\hat{\boldsymbol{\pi}}_t, e^{\hat{\alpha}_t}, e^{\hat{\beta}_t}, e^{\hat{\gamma}_t}))\sqrt{2\Gamma_t\Lambda^{(2)}_t} \xrightarrow{asymptotically} N(0, \Gamma_t) \tag{13}$$

with independent increments, where Γ_t is determined from 8 as shown below.

Letting $I_{i,j}$ denote the (i,j)th entry of the Hessian matrix $-\ddot{l}_t^{(2)}$ (see equation 8) of second partial derivatives with respect to a, b, d evaluated at $(\hat{a}_t, \hat{b}_t, 0)$, define

$$\Gamma_t = I_{33} - (I_{31}\ I_{32}) \begin{pmatrix} I_{11} & I_{12} \\ I_{21} & I_{22} \end{pmatrix}^{-1} \begin{pmatrix} I_{13} \\ I_{23} \end{pmatrix}. \tag{14}$$

In the code below, we actually compute the equivalent derivatives with respect to (α, β, d).

Assuming that there is a `hessian` function we can compute Γ_t as follows.

```
computeGammaSubT <- function(thetaHat, pi, interimData) {
    hes = hessian(thetaHat, pi, interimData)
    hes[3, 3] - t(hes[3, 1:2]) %*% solve(hes[1:2, 1:2]) %*% hes[1:2,
        3]
}
```

But we really need a function to compute the hessian. Here's the code followed by some explanations as to what the quantities in the code actually represent. (We are limited here by what Sweave can do!) In the code below, we compute the derivatives with respect to (α, β, d) and then use a change of variables from (α, β, γ) to (α, β, d). Harken back to your vector calculus, as I did.

```
hessian <- function(theta, pi, interimData) {
    m = nrow(interimData)
    x <- cbind(interimData$responseIndicator, interimData$treatmentIndicator,
        interimData$responseIndicator * interimData$treatmentIndicator)

    w1 <- theta[1] * x[, 1] + theta[2] * x[, 2] + theta[3] *
        x[, 3]
    expW1 <- exp(w1)
    w2 <- rev(cumsum(rev(expW1)))
    u <- matrix(0, nrow = m, ncol = 3)
    for (jj in 1:3) for (ii in 1:m) u[ii, jj] <- sum(x[ii:m,
        jj] * expW1[ii:m])/w2[ii]

    likdot <- t(interimData$delta) %*% (x - u)

    likddot <- matrix(0, nrow = 3, ncol = 3)
    u2 <- matrix(0, nrow = 3, ncol = 3)
    for (ii in 1:m) {
        u2[1:3, 1:3] <- 0  ## zero out u2
        for (jj in ii:m) {
            u2 <- u2 + (x[jj, ] %*% t(x[jj, ])) * expW1[jj]
        }
        likddot <- likddot + interimData$delta[ii] * (u2/w2[ii] -
```

```
            u[ii, ] %*% t(u[ii, ]))
    }
    likddot = -likddot

    pi0 <- pi[1]
    pi1 <- pi[2]
    xi <- exp(theta)
    d0 <- (pi0 * xi[1] + 1 - pi0) - (pi1 * xi[1] * xi[3] + 1 -
        pi1) * xi[2]

    B <- (pi0 * xi[1] + 1 - pi0) - (1 - pi1) * xi[2] - d0

    A1 <- pi0 * xi[1]/B - 1
    A2 <- -(1 - pi1) * xi[2]/B - 1
    A3 <- -1/B
    D2 <- rbind(c(1, 0, 0), c(0, 1, 0), c(A1, A2, A3))

    D <- matrix(0, 3, 3)
    D[1, 1] <- -A1 * (A1 + 1)
    D[2, 2] <- -A2 * (A2 + 1)
    D[1, 2] <- D[2, 1] <- -(A1 + 1) * (A2 + 1)
    D[2, 3] <- D[3, 2] <- -A3 * (A2 + 1)
    D[1, 3] <- D[3, 1] <- -A3 * (A1 + 1)
    D[3, 3] <- -A3^2
    -(likdot[3] * D + t(D2) %*% likddot %*% D2)
}
```

The following formulas are used for computing the derivatives ($\ddot{l}_t^{(2)}$) with respect to α, β, γ, the quantity `likddot` in the code. For notational convenience/brevity, first set

$$
\begin{aligned}
D_i &= \sum_{j \in R_i(t)} \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (15)\\
P_i^{\alpha} &= \sum_{j \in R_i(t)} Y_j \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (16)\\
P_i^{\beta} &= \sum_{j \in R_i(t)} Z_j \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (17)\\
P_i^{\gamma} &= \sum_{j \in R_i(t)} Y_j Z_j \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (18)\\
Q_i^{\alpha} &= \sum_{j \in R_i(t)} Y_j^2 \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (19)\\
Q_i^{\beta} &= \sum_{j \in R_i(t)} Z_j^2 \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (20)\\
Q_i^{\gamma} &= \sum_{j \in R_i(t)} Y_j^2 Z_j^2 \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (21)\\
Q_i^{\alpha\beta} &= \sum_{j \in R_i(t)} Y_j Z_j \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (22)\\
Q_i^{\alpha\gamma} &= \sum_{j \in R_i(t)} Y_j^2 Z_j \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j) && (23)\\
Q_i^{\beta\gamma} &= \sum_{j \in R_i(t)} Y_i Z_j^2 \exp(\alpha Y_j + \beta Z_j + \gamma Y_j Z_j). && (24)
\end{aligned}
$$

Then, the partial first and second derivaties have the nice forms

$$\frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\alpha} = \sum_{i=1}^{n}\delta_i(t)\Big\{Y_i - \frac{P_i^{\alpha}}{D_i}\Big\}, \tag{25}$$

$$\frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\beta} = \sum_{i=1}^{n}\delta_i(t)\Big\{Z_i - \frac{P_i^{\beta}}{D_i}\Big\}, \tag{26}$$

$$\frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\gamma} = \sum_{i=1}^{n}\delta_i(t)\Big\{Y_iZ_i - \frac{P_i^{\gamma}}{D_i}\Big\}, \tag{27}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\alpha^2} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\alpha} - (P_i^{\alpha})^2}{D_i^2}\Big\} \tag{28}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\beta^2} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\beta} - (P_i^{\beta})^2}{D_i^2}\Big\} \tag{29}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\gamma^2} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\gamma} - (P_i^{\gamma})^2}{D_i^2}\Big\} \tag{30}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\alpha\beta} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\alpha\beta} - P_i^{\alpha}P_i^{\beta}}{D_i^2}\Big\} \tag{31}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\alpha\gamma} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\alpha\gamma} - P_i^{\alpha}P_i^{\gamma}}{D_i^2}\Big\} \tag{32}$$

$$\frac{\partial^2 l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\beta\gamma} = -\sum_{i=1}^{n}\delta_i(t)\Big\{\frac{D_iQ_i^{\beta\gamma} - P_i^{\beta}P_i^{\gamma}}{D_i^2}\Big\}, \tag{33}$$

leading to a simple matrix implementation as follows.

Let

$$X = \begin{bmatrix} Y_1 & Z_1 & Y_1Z_1 \\ Y_2 & Z_2 & Y_2Z_2 \\ \vdots & \vdots & \vdots \\ Y_m & Z_m & Y_mZ_m \end{bmatrix}, \Delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_m \end{bmatrix}$$

assuming that the rows are ordered by the time to event. (The latter enables easy calculation of "risk-set" based quantities as cumulative sums.)

Set

$$W1 = \begin{bmatrix} \alpha Y_1 + \beta Z_1 + \gamma Y_1Z_1 \\ \alpha Y_1 + \beta Z_1 + \gamma Y_1Z_1 \\ \vdots \\ \alpha Y_m + \beta Z_m + \gamma Y_mZ_m \end{bmatrix}, W2 = \begin{bmatrix} \sum_{i=1}^{m}\exp(\alpha Y_i + \beta Z_i + \gamma Y_iZ_i) \\ \sum_{i=2}^{m}\exp(\alpha Y_i + \beta Z_i + \gamma Y_iZ_i) \\ \vdots \\ \sum_{i=m}^{m}\exp(\alpha Y_i + \beta Z_i + \gamma Y_iZ_i) \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_m \end{bmatrix}.$$

In the code above,

$$U = \begin{bmatrix} P_1^\alpha/D_1 & P_1^\beta/D_1 & P_1^\gamma/D_1 \\ P_2^\alpha/D_2 & P_2^\beta/D_2 & P_2^\gamma/D_2 \\ \vdots & \vdots & \vdots \\ P_m^\alpha/D_m & P_m^\beta/D_m & P_m^\gamma/D_m \end{bmatrix}$$

and therefore,

$$\begin{bmatrix} \frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\alpha} & \frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\beta} & \frac{\partial l_t^{(2)}(\alpha,\beta,\gamma)}{\partial\gamma} \end{bmatrix} = \Delta^T(X-U).$$

The latter is precisely `likdot` in the code.

For the second derivative `likddot`, the matrix `u2` in the code above, for each i, $1 \le i \le m$ is merely

$$U2_i = \begin{bmatrix} Q_i^\alpha & Q_i^{\alpha\beta} & Q_i^{\alpha\gamma} \\ Q_i^{\alpha\beta} & Q_i^\beta & Q_i^{\beta\gamma} \\ Q_i^{\alpha\gamma} & Q_i^{\beta\gamma} & Q_i^\gamma \end{bmatrix}.$$

To perform the change of variables, note that

$$\gamma = \log\{(\pi_0 e^\alpha + 1 - \pi_0) - (1-\pi_1)e^\beta - d\} - \log\pi_1 - \alpha - \beta. \tag{34}$$

So, letting $l = l_t^{(2)}$ for brevity,

$$\begin{aligned} \frac{\partial^2 l(\alpha,\beta,d)}{\partial(\alpha,\beta,d)^2} &= \left[\frac{\partial l(\alpha,\beta,\gamma)}{\partial(\alpha,\beta,d)}\right]_{3\times3}^T \times \left[\frac{\partial^2 l(\alpha,\beta,\gamma)}{\partial(\alpha,\beta,d)^2}\right]_{3\times3} \times \left[\frac{\partial l(\alpha,\beta,\gamma)}{\partial(\alpha,\beta,d)}\right]_{3\times3} \\ &+ \frac{\partial l}{\partial\alpha}\left[\frac{\partial^2\alpha}{\partial(\alpha,\beta,d)^2}\right]_{3\times3} + \frac{\partial l}{\partial\beta}\left[\frac{\partial^2\beta}{\partial(\alpha,\beta,d)^2}\right]_{3\times3} + \frac{\partial l}{\partial\gamma}\left[\frac{\partial^2\gamma}{\partial(\alpha,\beta,d)^2}\right]_{3\times3} \end{aligned} \tag{35}$$

The second derivatives on the last line of equation 35 except for $\frac{\partial^2\gamma}{\partial(\alpha,\beta,d)^2}$ yield zero matrices.

Now, from equation 34, letting $B = \pi_0 e^\alpha + 1 - \pi_0 - (1-\pi_1)e^\beta - d$ we calculate $\frac{\partial^2\gamma}{\partial(\alpha,\beta,d)^2}$

as follows:

$$\frac{\partial\gamma}{\partial\alpha} = \frac{\pi_0 e^{\alpha}}{B} - 1 \qquad \triangleq \texttt{A1} \tag{36}$$

$$\frac{\partial\gamma}{\partial\beta} = \frac{(\pi_1 - 1)e^{\beta}}{B} - 1 \qquad \triangleq \texttt{A2} \tag{37}$$

$$\frac{\partial\gamma}{\partial d} = \frac{-1}{B} \qquad \triangleq \texttt{A3} \tag{38}$$

$$\frac{\partial^2\gamma}{\partial\alpha^2} = \frac{\pi_0 e^{\alpha}}{B}(1 - \frac{\pi_0 e^{\alpha}}{B}) \qquad = \texttt{(A1+1)*(-A1)} \tag{39}$$

$$\frac{\partial^2\gamma}{\partial\alpha\beta} = \frac{\partial^2\gamma}{\partial\beta\alpha} = \frac{\pi_0 e^{\alpha+\beta}(1-\pi_1)}{B^2} \qquad = \texttt{-(A1+1)*(A2+1)} \tag{40}$$

$$\frac{\partial^2\gamma}{\partial\alpha d} = \frac{\partial^2\gamma}{\partial d\alpha} = \frac{\pi_0 e^{\alpha}}{B^2} \qquad = \texttt{(-A3)*(A1+1)} \tag{41}$$

$$\frac{\partial^2\gamma}{\partial\beta^2} = \frac{(\pi_1 - 1)e^{\beta}}{B}(1 + \frac{(1-\pi_1)e^{\beta}}{B}) \qquad = \texttt{(A2+1)*(-A2)} \tag{42}$$

$$\frac{\partial^2\gamma}{\partial\beta d} = \frac{\partial^2\gamma}{\partial d\beta} = \frac{1}{B^2}((\pi_1 - 1)e^{\beta}) \qquad = \texttt{(-A3)*(A2+1)} \tag{43}$$

$$\frac{\partial^2\gamma}{\partial d^2} = \frac{-1}{B^2} \qquad = \texttt{-A3**2} \tag{44}$$

That should clarify the last line of the `hessian` function.

But the interim analyses are carried out at calendar times and Γ_t or even n_t might not be known except at the interim analysis time. Therefore, at the design stage, we need to employ some approximations, to compute approximate boundaries for *all but the final* interim look. (At the last look, the boundary can be computed using actual quantities to control the Type I error after tabulating how much has actually been spent.)

We need the boundary conditions to satisfy the level α constraint such that

$$\text{pr}_{\pi_0=\pi_1}(\hat{\pi}_{t_j,0} < \hat{\pi}_{t_j,1} \text{ and } \Lambda^{(1)}_{t_j} \geq b_R \text{ for some } 1 \leq j < k, \text{ or } \hat{\pi}_{t_k,0} < \hat{\pi}_{t_k,1} \text{ and } \Lambda^{(1)}_{t_k} \geq c_R) = \alpha, \tag{45}$$

and

$$\begin{aligned}\text{pr}_{d=0}\{d(\hat{\boldsymbol{\pi}}_{t_j}, e^{\hat{\alpha}_{t_j}}, e^{\hat{\beta}_{t_j}}, e^{\hat{\gamma}_{t_j}}) > 0 \text{ and } \Lambda^{(2)}_{t_j} \geq b_S \text{ for some } k_0 \leq j < k,\\ \text{or } d(\hat{\boldsymbol{\pi}}_{t_k}, e^{\hat{\alpha}_{t_k}}, e^{\hat{\beta}_{t_k}}, e^{\hat{\gamma}_{t_k}}) > 0 \text{ and } \Lambda^{(2)}_{t_k} \geq c_S\} = \alpha.\end{aligned} \tag{46}$$

The boundary conditions, are the specific values for b_R, b_S and c_R, c_s in a one-sided or two-sided test. For given variances n_t for each interim look, we calculate b_R, b_S by

$$\begin{aligned}&\text{pr}\{(\textstyle\sum_{i=1}^{j} z_i)/\sqrt{2j} \geq \sqrt{b_R} \text{ for some } 1 \leq j \leq k-1\} = \varepsilon\alpha,\\ &\text{pr}\{(\textstyle\sum_{i=1}^{j} z_i)/\sqrt{2j} \geq \sqrt{b_S} \text{ for some } 1 \leq j \leq k-k_0\} = \varepsilon\alpha,\end{aligned} \tag{47}$$

We use the random walk approximation to the signed-root likelihood ratio statistics, with $k-1$ or $k-k_0$ standard normal increments z_i. we solve the previous equation for b_R and b_S using function as follows for a one-sided or two-sided test.

```
mHP.b <- function(mu = c(0, 0), v = c(1, 2), alpha = 0.05, eps = 1/2,
    side = c("one", "two")) {
    side <- match.arg(side)
    vsq = sqrt(v)
    v0 = diag(v)
    k = length(v)
    if (k > 1) {
        for (i in 1:(k - 1)) for (j in (i + 1):k) v0[i, j] = v0[j,
            i] = v[i]
    }

    alphaEps <- alpha * eps
    ## Calculate the prob of stopping at interim looks
    crossingProbDiff <- function(b = 3) {
        if (length(v) == 1) {
            if (side == "one")
                pnorm(-b * vsq, mean = mu, sd = vsq) - alphaEps else 2 * pnorm(-b * vsq,
        } else {
            bound <- b * vsq
            if (side == "one")
                1 - pmvnorm(lower = -Inf, upper = bound, mean = mu,
                  sigma = v0, algorithm = Miwa()) - alphaEps else 1 - pmvnorm(lower = -b
                sigma = v0, algorithm = Miwa()) - alphaEps
        }
    }

    if (k == 1) {
        if (side == "one")
            list(alpha = alpha, eps = eps, mu = mu, v = v, b = qnorm(1 -
                alphaEps), alpha.spent = alphaEps) else if (side == "two")
            list(alpha = alpha, eps = eps, mu = mu, v = v, b = qnorm(1 -
                alphaEps/2), alpha.spent = alphaEps)
    } else {
        w <- uniroot(f = crossingProbDiff, lower = qnorm(1 -
            alphaEps/2), upper = qnorm(1 - alphaEps/length(mu)/2))
        ## cat('b=',w$root,'\n')
        list(alpha = alpha, eps = eps, mu = mu, v = v, b = w$root,
            alpha.spent = w$f.root + alphaEps)
    }
}
```

Note that we use a (univariate) zero-finding function in R called `uniroot` and that the boundary above is one-sided by default.

We need a function for computing the final boundary after taking into account the α spent so far.

```
mHP.c <- function(mu = c(0, 0, 0), v = c(1, 2, 3), b = 3, alpha = 0.05,
    eps = 1/2, side = c("one", "two")) {
    side = match.arg(side)
    vsq = sqrt(v)
    bVsq = b * vsq
    k = length(v)
    v0 = diag(v)
    for (i in 1:(k - 1)) for (j in (i + 1):k) v0[i, j] = v0[j,
        i] = v[i]

    crossingProbDiff <- function(c1 = 3) {
        bound <- c(bVsq[1:(k - 1)], c1 * sqrt(v[k]))
        if (side == "one")
            1 - pmvnorm(lower = -Inf, upper = bound, mean = mu,
                sigma = v0, algorithm = Miwa()) - alpha else 1 - pmvnorm(lower = -bound,
            sigma = v0, algorithm = Miwa()) - alpha
    }

    w <- uniroot(f = crossingProbDiff, lower = qnorm(1 - alpha),
        upper = b)
    list(alpha = alpha, eps = eps, mu = mu, v = v, b = b, c = w$root,
        type1 = w$f.root + alpha)
}
```

5 Performing The Interim Looks

This is the most complicated part of the program because the null is composite (H_0^R or H_0^S) and information needs to be carried from one interim look to the next. Two items are particularly needed: the rejection status and the variance of the stochastic process defined by the GLR statistic at the interim look times.

In the initial stages, each interim look tests H_0^R for rejection (and futility if so specified). Only when H_0^S is rejected, the testing of H_0^S proceeds. And there too futility may be tested if so specified.

Some points of detail. H_0^S cannot be tested at an interim look that has too few events. How few? This is a design parameter: `trialParameters$minimumNumberOfEvents`. Also, we do not reject for efficacy unless the variance Γ_t at the look is larger than a certain fraction of the previous variance. (Looks like we are guarding against something, what precisely???)

At each interim look, the function will return a vector of several quantities. Some of them are vectorial quantities although they will be returned as elements of a large vector.

$\hat{\boldsymbol{\pi}}_t$ the current estimate of the response probabilities

$\hat{\boldsymbol{\pi}}_{t,0}$ the estimate of the response probabilities under H_0

$\hat{\boldsymbol{\pi}}_{t,1}$ the estimate of the response probabilities under H_1

$\hat{\boldsymbol{\theta}}_t$ the current estimate of (α, β, γ)

$\hat{\boldsymbol{\theta}}_{t,0}$ the estimate under H_0

$\hat{\boldsymbol{\theta}}_{t,1}$ the estimate under H_1

$\hat{\Lambda}_t$ the current estimate of the Generalized Likelihood Ratio (GLR) statistic

$\hat{\Lambda}_{t,0}$ the estimate under H_0

$\hat{\Lambda}_{t,1}$ the estimate under H_1

$\hat{\Lambda}_{t,1}$ the estimate under H_1

Γ_t if computed

Note that programmatic convenience, another value named `usedV` is returned (if Γ_t is computed) that takes into account the minimum increase in variance of the stochastic process.

In addition to the above, four boolean quantities will be returned that are

`rejectH0R` indicating if H_0^R was rejected at the interim look

`acceptH0R` indicating if H_0^R was accepted (futility) at the interim look

`rejectH0S` indicating if H_0^S was rejected at the interim look

`acceptH0S` indicating if H_0^S was accepted (futility) at the interim look

A last quantity, the last boundary which is computed at *run time* is also returned (as a missing value if not computed).

One can now think of the trial progress as an accumulating series of rows in a matrix with the columns above. However, each look will need the value of the `rejectH0R` from the previous look in order to decide which component of the composite H_0 to test. Initially, this value is, of course, `FALSE`.

With that, we proceed, noting that a lot of statements merely store the computed quantities in the result vector.

```
performInterimLook <- function(k, trueParameters, trialParameters,
    glrBoundary, interimData, interimLookHistoryDF, argRejectH0R) {
    # cat('k = ', k, '\n')
    result <- c(rep(NA, ncol(interimLookHistoryDF) - 4), FALSE,
        FALSE, FALSE, FALSE, NA)
    names(result) <- c(names(interimLookHistoryDF), "b.metas.Last")
    numLooks <- length(trialParameters$interimLookTime)
    pdiff.hyp <- trueParameters$pdiffHyp
    m <- nrow(interimData)
    interimNumberOfEvents = sum(interimData$delta)
```

```
    ## Compute some quantities
    responseSummary <- computeResponseSummary(interimData)
    piHat <- responseSummary[c("controlRespProp", "treatmentRespProp")]
    pi0Hat <- piHat[1]
    pi1Hat <- piHat[2]
    piHatH0 <- rep(responseSummary["pooledProp"], 2)
    w1 <- loglik1(piVec = piHat, responseSummary)
    ## Compute GLR for Response
    glrRespH0 <- w1 - loglik1(piVec = piHatH0, responseSummary)
    ## Record values
    result["m0"] <- responseSummary["m0"]
    result["m1"] <- responseSummary["m1"]
    result["y0"] <- responseSummary["y0"]
    result["y1"] <- responseSummary["y1"]
    result["pi0Hat"] <- pi0Hat
    result["pi1Hat"] <- pi1Hat
    result["pi0HatH0"] <- piHatH0[1]
    result["pi1HatH0"] <- piHatH0[2]
    result["glrRespH0"] <- glrRespH0

    rejectH0R <- argRejectH0R  ## Make a current copy

    ## Skip if you have already rejected H_0^R This comes from
    ## previous interim look!
    if (!rejectH0R) {
        rejectH0R <- ((2 * glrRespH0 >= glrBoundary[k, "RespEfficacy"]^2) &&
            (pi0Hat < pi1Hat))  ## Response efficacy
        ## result['rejectH0R'] <- rejectH0R Now you have not rejected
        ## H_0^R, check for futility This comes from current interim
        ## look
        if (!rejectH0R) {
            initialPStart <- max(piHat[1] - responseSummary["m1"] *
                pdiff.hyp/responseSummary["m"], 0.01)
            test <- optim(par = initialPStart, fn = loglik1GivenDelta,
                respSummary = responseSummary, delta = pdiff.hyp,
                lower = 0.001, upper = 0.999 - pdiff.hyp, method = "L-BFGS-B")
            piHatH1 <- c(test$par, test$par + pdiff.hyp)
            glrRespH1 <- w1 - loglik1(piVec = piHatH1, responseSummary)
            ## record values
            result["pi0HatH1"] <- piHatH1[1]
            result["pi1HatH1"] <- piHatH1[2]
            result["glrRespH1"] <- glrRespH1
            ## futility test equation 19 of paper
            acceptH0R <- ((2 * glrRespH1 >= glrBoundary[k, "RespFutility"]^2) &&
```

```
                (pi1Hat < pi0Hat + pdiff.hyp))
            result["acceptH0R"] <- acceptH0R
        }
    }
    result["rejectH0R"] <- rejectH0R
    ## If, by the penultimate look H_0^R is not rejected, give up
    if ((k == (numLooks - 1)) && !rejectH0R) {
        acceptH0R <- TRUE
        result["acceptH0R"] <- acceptH0R
    } else {
        ## You might have rejected H_0^R in a previous look OR in the
        ## current look so you have test on the flag again!
        if (rejectH0R) {
            rejectH0S <- acceptH0S <- FALSE  ## initially false
            if (k < numLooks) {
                # for all but final looks
                test <- optim(par = c(0, 0, 0), fn = loglik2,
                  interimData = interimData, control = list(fnscale = -1))
                ## Store the MLE theta.hat
                thetaHat <- test$par
                xiHat <- exp(thetaHat)
                hazard = computeDGivenXi(piHat, xiHat)
                ## cat('thetaHat=', thetaHat, '\n')
                test <- constrOptim(theta = c(1.5, 0.5), f = loglik2.repar0,
                  grad = NULL, ui = rbind(c(1, 0), c(0, 1), c(pi0Hat,
                    pi1Hat - 1)), ci = c(0, 0, pi0Hat - 1 + 0),
                  control = list(fnscale = -1), interimData = interimData,
                  pi0 = pi0Hat, pi1 = pi1Hat)
                ## Now compute the c back (= w)
                w <- solveForCGivenABD(piHat, test$par[1], test$par[2],
                  d = 0)
                thetaHatH0 <- log(c(test$par, w))
                w1 = loglik2(theta = thetaHat, interimData)
                glrSurvH0 = w1 - loglik2(theta = thetaHatH0,
                  interimData)
                if (hazard < trueParameters$etaHyp) {
                  test <- constrOptim(theta = c(3, 0.2), f = loglik2.repar0,
                    grad = NULL, ui = rbind(c(1, 0), c(0, 1),
                      c(pi0Hat, pi1Hat - 1)), ci = c(0, 0, trueParameters$etaHyp -
                      (1 - pi0Hat) + 0), control = list(fnscale = -1),
                    interimData = interimData, pi0 = pi0Hat,
                    pi1 = pi1Hat, eta.hyp = trueParameters$etaHyp)
                  w = solveForCGivenABD(piHat, test$par[1], test$par[2],
                    trueParameters$etaHyp)
```

```
    thetaHatH1 = log(c(test$par, w))
    glrSurvH1 = w1 - loglik2(theta = thetaHatH1,
      interimData)
  } else {
    thetaHatH1 <- rep(NA, 3)
    glrSurvH1 = NA
  }
  ## Record quantities
  result["glrSurvH0"] <- glrSurvH0
  result["glrSurvH1"] <- glrSurvH1
  result["alphaHat"] <- thetaHat[1]
  result["betaHat"] <- thetaHat[2]
  result["gammaHat"] <- thetaHat[3]
  result["hazard"] <- hazard
  result["alphaHatH0"] <- thetaHatH0[1]
  result["betaHatH0"] <- thetaHatH0[2]
  result["gammaHatH0"] <- thetaHatH0[3]
  # print('before')
  result["alphaHatH1"] <- thetaHatH1[1]
  result["betaHatH1"] <- thetaHatH1[2]
  result["gammaHatH1"] <- thetaHatH1[3]
  # print('after')

  ## Start test for efficacy Now have to take into account that
  ## we may have too few events THIS MEANS: ENSURE THAT NO ONE
  ## SPECIFIES TRIAL MIN NO OF EVENTS TO BE > any interim # of
  ## events
  if (interimNumberOfEvents >= trialParameters$minimumNumberOfEvents) {
    ## CHECK CHECK CHECK for proper args theta and pi
    v <- computeGammaSubT(thetaHat = thetaHatH0,
      pi = piHat, interimData)
    result["v"] <- v
    if (v > 0) {
      usedVIndices <- which(interimLookHistoryDF$usedV >
        0)
      nUV <- length(usedVIndices)
      if (nUV > 0) {
        ## Only record if it is large enough compared to the previous
        ## recorded value
        if (v >= interimLookHistoryDF$usedV[usedVIndices[nUV]] *
          (1 + trialParameters$minimumIncreaseInV))
          result["usedV"] <- v
      } else {
        result["usedV"] <- v
```

```
            }
          }
          rejectH0S <- ((2 * glrSurvH0 >= glrBoundary[k,
            "SurvEfficacy"]^2) && (hazard > 0))
          result["rejectH0S"] <- rejectH0S
        }
        ## End test for efficacy Start test for futility
        acceptH0S <- ((2 * glrSurvH1 >= glrBoundary[k,
          "SurvFutility"]^2) && (hazard < trueParameters$etaHyp))
        result["acceptH0S"] <- acceptH0S
        ## End test for futility final look
      } else {
        test <- optim(par = c(0, 0, 0), fn = loglik2,
          interimData = interimData, control = list(fnscale = -1))
        ## Store the MLE theta.hat
        thetaHat <- test$par
        xiHat <- exp(thetaHat)
        hazard = computeDGivenXi(piHat, xiHat)
        ## cat('thetaHat=', thetaHat, '\n')
        test <- constrOptim(theta = c(1.5, 0.5), f = loglik2.repar0,
          grad = NULL, ui = rbind(c(1, 0), c(0, 1), c(pi0Hat,
            pi1Hat - 1)), ci = c(0, 0, pi0Hat - 1 + 0),
          control = list(fnscale = -1), interimData = interimData,
          pi0 = pi0Hat, pi1 = pi1Hat)
        ## Now compute the c back (= w)
        w <- solveForCGivenABD(piHat, test$par[1], test$par[2],
          d = 0)
        thetaHatH0 <- log(c(test$par, w))
        w1 = loglik2(theta = thetaHat, interimData)
        glrSurvH0 = w1 - loglik2(theta = thetaHatH0,
          interimData)
        ## Record quantities
        result["glrSurvH0"] <- glrSurvH0
        result["alphaHat"] <- thetaHat[1]
        result["betaHat"] <- thetaHat[2]
        result["gammaHat"] <- thetaHat[3]
        result["hazard"] <- hazard
        result["alphaHatH0"] <- thetaHatH0[1]
        result["betaHatH0"] <- thetaHatH0[2]
        result["gammaHatH0"] <- thetaHatH0[3]
        v <- computeGammaSubT(thetaHat = thetaHatH0,
          pi = piHat, interimData)
        result["v"] <- result["usedV"] <- v
        usedVIndices <- which(interimLookHistoryDF$usedV >
```

```
                  0)
                nUV <- length(usedVIndices)
                ## cat('before nUV', nUV, 'nonZeroVIndices', nonZeroVIndices,
                ## '\n') cat('V So far', interimLookHistoryDF$v, '\n')
                usedV <- {
                  if (nUV > 0)
                    c(interimLookHistoryDF$usedV[usedVIndices],
                      v) else v
                }
                ## NEED TO CALCULATE C=B.METAS[K]
                nUV <- length(usedV)
                ## cat('after nUV', nUV, 'usedV', usedV, '\n')
                if (nUV == 1) {
                  b.metas.Last <- qnorm(1 - trialParameters$type1Error)
                } else if (nUV >= 2) {
                  if (usedV[nUV] < usedV[nUV - 1]) {
                    b.metas.Last <- Inf
                  } else {
                    test <- mHP.c(mu = numeric(nUV), v = usedV,
                      b = glrBoundary[1, "SurvEfficacy"], alpha = trialParameters$type1E
                      eps = trialParameters$epsType1)
                    b.metas.Last <- test$c
                  }
                }
                result["b.metas.Last"] <- b.metas.Last
                ## cat('c=', b.metas.Last,'\n') cat('k=5 (final look):
                ## UsedV=',usedV,'\n') Check boundary crossing for
                ## c=b.metas[k]
                rejectH0S <- ((2 * glrSurvH0 >= b.metas.Last^2) &&
                  (hazard > 0))
                acceptH0S <- !rejectH0S
                result["rejectH0S"] <- rejectH0S
                result["acceptH0S"] <- acceptH0S
              }
            }
        }
        ## Return result vector
        result
}
```

6 Phase II-III Design Generation and Execution

The typical use for clinical design software can be broken down into two major activities:

1. A design phase where several parameters are explored, tested and tweaked to see the effect they have on output values. The output would then give some ideas to the user about what design parameters to choose for his or her own design, thus fixing parameter.

2. An execution phase where the design in the step above is actually put in practice. That is, data from a clinical trial arrives and the chosen design above is applied. If one thinks of a design as an oracle, the oracle will suggest at every stage of the clinical trial what decision needs to be taken.

In our Phase II-III design, the design phase actually uses simulation to arrive at design characteristics and therefore uses the execution phase. Therefore, we need to clearly delineate the steps.

6.1 Design Generation

We need a function that will generate a design object given the true state of nature and the trial parameters.

```
generateSP23Design <- function(trueParameters, trialParameters) {
  interimLookTime <- trialParameters$interimLookTime
  ##
  ## Some sanity checks
  ##
  ## Interim look times must be ordered
  if (sum(abs(sort(interimLookTime) - interimLookTime)) > 0) {
    stop(cat("Trial interim Look times", interimLookTime, "must be ordered\n"))
  }
  numLooks <- length(interimLookTime)
  N = sum(trialParameters$numberRecruitedEachYear) ## total no of subjects

  ## Compute the boundaries for efficacy for response
  test1 = mHP.b(mu = numeric(numLooks - 2),
    v = cumsum(trialParameters$numberRecruitedEachYear[1:(numLooks - 2)]),  ## why 1:3??
    alpha = trialParameters$type1ErrorForResponse,
    eps = trialParameters$epsType1,
    side = trialParameters$glrBoundarySidedness)

  test2 = mHP.c(mu = numeric(numLooks - 1),
    v = cumsum(trialParameters$numberRecruitedEachYear),  ## why 1:3???,
    b = test1$b,
    alpha = trialParameters$type1ErrorForResponse,
    eps = trialParameters$epsType1,
    side = trialParameters$glrBoundarySidedness)
```

```
  b.resp = c(rep(test1$b, numLooks - 2), test2$c, Inf) ## efficacy boundary

  if (is.na(trialParameters$type2ErrorForResponse)) {
    b.tilde.resp <- rep(Inf, numLooks)
  } else {
    test3 = mHP.b(mu = numeric(numLooks - 2),
      v = cumsum(trialParameters$numberRecruitedEachYear[1:(numLooks - 2)]),
      alpha = trialParameters$type2ErrorForResponse,
      eps = trialParameters$epsType2,
      side = trialParameters$glrBoundarySidedness)
    b.tilde.resp = c(rep(test3$b, numLooks - 2), Inf, Inf) ## futility boundary
  }

  test1 = mHP.b(mu = numeric(numLooks - 1),
    v = 1:(numLooks - 1),
    alpha = trialParameters$type1Error,
    eps = trialParameters$epsType1,
    side = trialParameters$glrBoundarySidedness)
  b.metas = c(rep(test1$b, numLooks - 1), Inf)  ## efficacy boundary

  if (is.na(trialParameters$type2Error)) {
    b.tilde.metas <- rep(Inf, numLooks)
  } else {
    test2 = mHP.b(mu = rep(0, 4),
      v = 1:(numLooks - 1),
      alpha = trialParameters$type2Error,
      eps = trialParameters$epsType2,
      side = trialParameters$glrBoundarySidedness)
    b.tilde.metas = c(rep(test2$b, numLooks - 1), Inf)  ## futility boundary
  }

##  cat("b.resp=",b.resp,"\n")
##  cat("b.tilde.resp=",b.tilde.resp,"\n")
##  cat("b.metas=",b.metas,"\n")
##  cat("b.tilde.metas=",b.tilde.metas,"\n\n")
  glrBoundary <- matrix(c(b.resp, b.tilde.resp, b.metas, b.tilde.metas), ncol=4)
  colnames(glrBoundary) <- c("RespEfficacy", "RespFutility", "SurvEfficacy", "SurvFutili


  naVector <- rep(NA, numLooks) # a qty we reuse
  logicalVector <- logical(numLooks) # another qty we reuse.
  interimLookHistoryDF <- data.frame(m0 = naVector,
                                     m1 = naVector,
                                     y0 = naVector,
```

```
                                        y1 = naVector,
                                        pi0Hat = naVector,
                                        pi1Hat = naVector,
                                        pi0HatH0 = naVector,
                                        pi1HatH0 = naVector,
                                        pi0HatH1 = naVector,
                                        pi1HatH1 = naVector,
                                        glrRespH0 = naVector,
                                        glrRespH1 = naVector,
                                        hazard = naVector,
                                        alphaHat = naVector,
                                        betaHat = naVector,
                                        gammaHat = naVector,
                                        alphaHatH0 = naVector,
                                        betaHatH0 = naVector,
                                        gammaHatH0 = naVector,
                                        alphaHatH1 = naVector,
                                        betaHatH1 = naVector,
                                        gammaHatH1 = naVector,
                                        glrSurvH0 = naVector,
                                        glrSurvH1 = naVector,
                                        v = naVector,
                                        usedV = naVector,
                                        rejectH0R = logicalVector,
                                        acceptH0R = logicalVector,
                                        rejectH0S = logicalVector,
                                        acceptH0S = logicalVector
                                        )
  list(trueParameters = trueParameters, trialParameters=trialParameters,
       glrBoundary=glrBoundary, interimLookHistoryDF=interimLookHistoryDF)
}
```

The function above computes the modified Haybittle-Peto boundaries for all but the last look; the latter needs to be computed at run time rather than design time.

For simulations, we will typically create one such design object and reset the statistics computed during interim looks to start afresh. So another function.

```
resetSP23Design <- function(sp23Design) {
    numLooks <- length(sp23Design$trialParameters$interimLookTime)
    naVector <- rep(NA, numLooks)  # a qty we reuse
    logicalVector <- logical(numLooks)  # another qty we reuse.

    interimLookHistoryDF <- data.frame(m0 = naVector, m1 = naVector,
        y0 = naVector, y1 = naVector, pi0Hat = naVector, pi1Hat = naVector,
```

```
        pi0HatH0 = naVector, pi1HatH0 = naVector, pi0HatH1 = naVector,
        pi1HatH1 = naVector, glrRespH0 = naVector, glrRespH1 = naVector,
        hazard = naVector, alphaHat = naVector, betaHat = naVector,
        gammaHat = naVector, alphaHatH0 = naVector, betaHatH0 = naVector,
        gammaHatH0 = naVector, alphaHatH1 = naVector, betaHatH1 = naVector,
        gammaHatH1 = naVector, glrSurvH0 = naVector, glrSurvH1 = naVector,
        v = naVector, usedV = naVector, rejectH0R = logicalVector,
        acceptH0R = logicalVector, rejectH0S = logicalVector,
        acceptH0S = logicalVector)
    list(trueParameters = sp23Design$trueParameters, trialParameters = sp23Design$trialP
        glrBoundary = sp23Design$glrBoundary, interimLookHistoryDF = interimLookHistoryD
}
```

6.2 Design Execution

Next, we need a function to actually execute the design. This, incidentally, also is the function that will be used in the field to perform the actual interim look.

```
executeSP23Design <- function(sp23DesignObject, data, currentCalendarTime) {
    ## Find the largest interim look time smaller than
    ## calendarTime
    interimLookTime <- sp23DesignObject$trialParameters$interimLookTime
    l <- which(interimLookTime <= currentCalendarTime)
    llen <- length(l)
    if (llen > 0) {
        k <- l[llen]  # the last index
        trialParameters <- sp23DesignObject$trialParameters
        glrBoundary <- sp23DesignObject$glrBoundary
        interimLookHistoryDF = sp23DesignObject$interimLookHistoryDF
        interimData <- generateInterimData(data, trialParameters$interimLookTime[k],
            trialParameters$adminCensoringTime)
        ## this ordering is important for the computation of the
        ## likelihood functions. The ordering is by time to event.
        interimData <- interimData[order(interimData$timeToEvent),
            ]

        performInterimLook(k, sp23DesignObject$trueParameters,
            trialParameters, glrBoundary, interimData, interimLookHistoryDF,
            argRejectH0R = ifelse(k == 1, FALSE, interimLookHistoryDF$rejectH0R[k -
                1]))
    } else {
        stop(cat("Trial design does not permit an interim look at time",
            currentCalendarTime, "!\n"))
```

```
    }
}
```

6.3 Exploring the Design

Given a design, this function will explore the design by running a specified number of simulations and reporting on the statistics.

```
exploreSP23Design <- function(sp23Design, numberOfSimulations = 25,
    rngSeed = 12345, showProgress = TRUE) {
    set.seed(rngSeed)
    trialHistory <- vector(mode = "list", length = numberOfSimulations)
    if (showProgress)
        pb <- txtProgressBar(min = 0, max = numberOfSimulations,
            style = 3)
    for (i in 1:numberOfSimulations) {
        sp23DesignObj <- resetSP23Design(sp23Design)  ## reset statistics
        trialParameters <- sp23DesignObj$trialParameters
        trueParameters <- sp23DesignObj$trueParameters
        d <- generateClinicalTrialData(nRec = trialParameters$numberRecruitedEachYear,
            nFUp = trialParameters$followupTime, pi0 = trueParameters$p0,
            pi1 = trueParameters$p1, theta = trueParameters$theta,
            lambda0 = trueParameters$baselineLambda)
        interimLookTime <- sp23DesignObj$trialParameters$interimLookTime
        numLooks <- length(interimLookTime)
        for (k in 1:numLooks) {
            result <- executeSP23Design(sp23DesignObj, d, interimLookTime[k])
            ## We gather the data for the interim history for the next
            ## look
            for (x in names(sp23DesignObj$interimLookHistoryDF)) {
                sp23DesignObj$interimLookHistoryDF[k, x] <- result[x]
            }
            ## Need to also store the last beta if calculated, but easier
            ## to store it any way
            if (k == numLooks) {
                sp23DesignObj$glrBoundary[k, "SurvEfficacy"] <- result["b.metas.Last"]
                sp23DesignObj$glrBoundary[k, "SurvFutility"] <- result["b.metas.Last"]
            }
            ## print(result)
            if (result["acceptH0R"] || result["acceptH0S"] ||
                (result["rejectH0R"] && result["rejectH0S"])) {
                ## we accepted H_0^R, or accepted H_0^S or rejected the
                ## composite null of (H_0^R & H_0^S) So we are done
```

```
                break
            }
        }
        trialHistory[[i]] <- sp23DesignObj$interimLookHistoryDF
        if (showProgress)
            setTxtProgressBar(pb, i)
    }
    if (showProgress)
        close(pb)
    trialHistory
}
```

We also need an analysis function to analyze the results of a simulation run.

```
analyzeSP23Design <- function(sp23Design, trialHistory = NULL,
    data = NULL, col = c("red", "red", "brown", "brown"), lty = c(1,
        2, 1, 2)) {
    result <- list(responseSummary = NA, designSummary = NA)
    trialParameters <- sp23Design$trialParameters
    numLooks <- length(trialParameters$interimLookTime)
    trialExitTime <- function(historyDF) {
        j <- which(historyDF$acceptH0R > 0)
        if (length(j) <= 0) {
            ## we did not accept H_0^R
            j <- which(historyDF$rejectH0S > 0)
            if (length(j) <= 0) {
                ## H_0^S and hence null was NOT rejected
                j <- which(historyDF$acceptH0S > 0)  ## H_0^S was accepted somewhere
            }
        }
        trialParameters$interimLookTime[j]  ## there should only be 1 such entry
    }
    trialExitLook <- function(historyDF) {
        j <- which(historyDF$acceptH0R > 0)
        if (length(j) <= 0) {
            ## we did not accept H_0^R
            j <- which(historyDF$rejectH0S > 0)
            if (length(j) <= 0) {
                ## H_0^S and hence null was NOT rejected
                j <- which(historyDF$acceptH0S > 0)  ## H_0^S was accepted somewhere
            }
        }
        j
    }
```

```
    if (is.data.frame(trialHistory)) {
        if (is.null(data))
            stop("Expect clinical trial data as parameter!")
        exitTime <- trialExitTime(trialHistory)
        interimData <- generateInterimData(data, exitTime, trialParameters$adminCensorin
        m <- nrow(interimData)
        arm <- factor(ifelse(interimData$treatmentIndicator ==
            1, "treatment", "control"))
        responder <- factor(ifelse(interimData$responseIndicator ==
            1, "responder", "non-responder"))
        interimData$arm <- arm
        interimData$responder <- responder
        sFit <- survfit(Surv(timeToEvent, delta) ~ arm + responder,
            data = interimData)
        plot(sFit, col = col, lty = lty, lwd = 2, xlab = "Time to Event (yrs)",
            ylab = "Survival", xlim = c(0, trialParameters$adminCensoringTime))
        legend <- c(paste(levels(arm)[1], "-", levels(responder)),
            paste(levels(arm)[2], "-", levels(responder)))
        legend(0.5, 0.2, legend, text.col = col, pt.lwd = 2,
            lty = lty, col = col)
        result$responseSummary <- computeResponseSummary(interimData)
    } else {
        ## Report on the statistics
        numberOfTimesH0RIsRejectedAtFirstLook <- sum(sapply(trialHistory,
            function(x) x$rejectH0R[1]))
        numberOfTimesH0RIsRejected <- sum(sapply(trialHistory,
            function(x) (sum(x$rejectH0R) > 0)))
        numberOfTimesStoppedForFutility <- sum(sapply(trialHistory,
            function(x) sum(x$acceptH0R | x$acceptH0S)))
        numberOfTimesH0SIsRejected <- sum(sapply(trialHistory,
            function(x) (sum(x$rejectH0S) > 0)))
        numberOfTimesH0SIsAccepted <- sum(sapply(trialHistory,
            function(x) (sum(x$acceptH0S) > 0)))
        numberOfTimesFutilityDecidedAtLastLook <- sum(sapply(trialHistory,
            function(x) x$acceptH0S[numLooks]))
        numberOfTimesTrialEndedAtLook <- sapply(1:numLooks, function(j) sum(sapply(trial
            function(x) x$acceptH0R[j] || x$acceptH0S[j] || x$rejectH0S[j])))
        names(numberOfTimesTrialEndedAtLook) <- as.character(trialParameters$interimLook
        trialExitTimes <- sapply(trialHistory, trialExitTime)
        avgExitTime <- mean(trialExitTimes)
        designSummary <- list(numberOfTimesH0RIsRejectedAtFirstLook = numberOfTimesH0RIs
            numberOfTimesH0RIsRejected = numberOfTimesH0RIsRejected,
            numberOfTimesStoppedForFutility = numberOfTimesStoppedForFutility,
            numberOfTimesH0SIsAccepted = numberOfTimesH0SIsAccepted,
```

```
            numberOfTimesH0SIsRejected = numberOfTimesH0SIsRejected,
            numberOfTimesFutilityDecidedAtLastLook = numberOfTimesFutilityDecidedAtLastL
            numberOfTimesTrialEndedAtLook = numberOfTimesTrialEndedAtLook,
            avgExitTime = avgExitTime)
        result$designSummary <- designSummary
    }
    result
}
```

7 Usage Examples

The trial parameters are the same for all examples in the paper.

```
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
```

Here are example parameters settings that allow one to simulate results in tables 1 and 2. Of course, we will use a small number of simulations. The code for all examples in the paper is stored in the `inst` subdirectory of the installed package.

```
trueParameters <- list(p0 = 0.3, p1 = 0.3, pdiffHyp = 0.3, theta = list(alpha = 0,
    beta = 0, gamma = 0), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 9872831
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.3, pdiffHyp = 0.3, theta = list(alpha = log(0.5)
    beta = 0, gamma = 0), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 324612
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = 0,
    beta = 0, gamma = 0), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 2387487
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.2)
    beta = log(1.56), gamma = log(1)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 283119
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.2)
    beta = log(1.73), gamma = log(0.8)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 98872361
```

```
trueParameters <- list(p0 = 0.6, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(1),
    beta = log(1), gamma = log(1)), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 623312
```

```
trueParameters <- list(p0 = 0.6, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.5)
    beta = log(1), gamma = log(1)), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 6232
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(1),
    beta = log(0.75), gamma = log(1)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 5554231
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.75
    beta = log(0.79), gamma = log(1)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 125352
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.5)
    beta = log(0.86), gamma = log(1)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 8669312
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.17
    beta = log(1), gamma = log(1)), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 7325543
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.75
    beta = log(1), gamma = log(0.61)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 466553
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.6, pdiffHyp = 0.3, theta = list(alpha = log(0.5)
    beta = log(1), gamma = log(0.67)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 1875543
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.5, pdiffHyp = 0.3, theta = list(alpha = log(0.75
    beta = log(1), gamma = log(0.47)), baselineLambda = 0.35,
    etaHyp = 0.25)
rngSeed <- 1093
```

```
trueParameters <- list(p0 = 0.3, p1 = 0.5, pdiffHyp = 0.3, theta = list(alpha = 0,
    beta = log(0.75), gamma = 0), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 117
```

```
trueParameters <- list(p0 = 0.2, p1 = 0.5, pdiffHyp = 0.3, theta = list(alpha = 0,
    beta = log(0.75), gamma = 0), baselineLambda = 0.35, etaHyp = 0.25)
rngSeed <- 9872831
```

8 Unit Testing

Unit testing is useful here although time consuming. So we have to use a small number of simulations. We use the above to test all the cases. The code running each test is the same, abstracted here.

```
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations = 25,
    rngSeed = rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
```

```
test.caseA <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.3,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = 0,
                         beta = 0,
                         gamma = 0),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 9872831
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 0)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 0)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 25)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 0)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 0)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 0)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(9, 13, 3, 0, 0); names(out) <- trialParameters$interimLookTime; out}

}
```

```
test.caseB <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
```

```
                          followupTime = 3,
                          adminCensoringTime = 7,
                          interimLookTime = c(1, 2, 3, 5, 7),
                          type1ErrorForResponse = 0.05,
                          type2ErrorForResponse = 0.01,
                          glrBoundarySidedness = "one", # one sided or two-sided
                          type1Error = 0.05,
                          type2Error = 0.10,
                          epsType1 = 1/3,
                          epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.3,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(0.5),
                         beta = 0,
                         gamma = 0),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 324612
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 0)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 0)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 25)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 0)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 0)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 0)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(9, 16, 0, 0, 0); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.caseC <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                          minimumIncreaseInV = 0.2,
                          numberRecruitedEachYear = c(80, 120, 160, 160),
                          followupTime = 3,
                          adminCensoringTime = 7,
                          interimLookTime = c(1, 2, 3, 5, 7),
                          type1ErrorForResponse = 0.05,
                          type2ErrorForResponse = 0.01,
                          glrBoundarySidedness = "one", # one sided or two-sided
```

```
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = 0,
                         beta = 0,
                         gamma = 0),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 2387487
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 14)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 23)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 23)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 2)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 7)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(1, 1, 3, 12, 8); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.caseD <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
```

```
                        pdiffHyp=0.3,
                        theta = list(
                          alpha = log(0.2),
                          beta = log(1.56),
                          gamma = log(1.0)),
                        baselineLambda = 0.35,
                        etaHyp = 0.25)
rngSeed <- 283119
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 22)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 24)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 24)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 1)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 1)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 3, 10, 11, 1); names(out) <- trialParameters$interimLookTime; out
}
```

```
test.caseE <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                        p1 = 0.6,
                        pdiffHyp=0.3,
                        theta = list(
                          alpha = log(0.2),
                          beta = log(1.73),
                          gamma = log(0.8)),
                        baselineLambda = 0.35,
```

```
                        etaHyp = 0.25)
rngSeed <- 98872361
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 16)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 25)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 25)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 0)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 0)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(1, 9, 11, 4, 0); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.caseF <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.6,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(1.0),
                         beta = log(1.0),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 623312
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
```

```
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 0)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 0)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 25)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 0)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 0)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 0)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(13, 8, 4, 0, 0); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.caseG <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.6,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(0.5),
                         beta = log(1.0),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 6232
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 0)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 1)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 25)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 1)
```

```
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 0)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 0)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(15, 8, 1, 1, 0); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run1 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(1.0),
                         beta = log(.75),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 5554231
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 13)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 5)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 5)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 20)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 4)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 3, 1, 5, 16); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run2 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(.75),
                         beta = log(.79),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 125352
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 9)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 4)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 4)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 21)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 2)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 4, 2, 10, 9); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run3 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
```

```
                          adminCensoringTime = 7,
                          interimLookTime = c(1, 2, 3, 5, 7),
                          type1ErrorForResponse = 0.05,
                          type2ErrorForResponse = 0.01,
                          glrBoundarySidedness = "one", # one sided or two-sided
                          type1Error = 0.05,
                          type2Error = 0.10,
                          epsType1 = 1/3,
                          epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(.5),
                         beta = log(.86),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 8669312
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 11)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 1)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 1)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 24)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 1)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 3, 3, 9, 10); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run4 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                          minimumIncreaseInV = 0.2,
                          numberRecruitedEachYear = c(80, 120, 160, 160),
                          followupTime = 3,
                          adminCensoringTime = 7,
                          interimLookTime = c(1, 2, 3, 5, 7),
                          type1ErrorForResponse = 0.05,
                          type2ErrorForResponse = 0.01,
                          glrBoundarySidedness = "one", # one sided or two-sided
                          type1Error = 0.05,
```

```
                         type2Error = 0.10,
                         epsType1 = 1/3,
                         epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(.17),
                         beta = log(1.0),
                         gamma = log(1.0)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 7325543
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 16)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 6)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 6)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 19)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 2)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 6, 2, 9, 8); names(out) <- trialParameters$interimLookTime; out})
}
```

```
test.run5 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                         minimumIncreaseInV = 0.2,
                         numberRecruitedEachYear = c(80, 120, 160, 160),
                         followupTime = 3,
                         adminCensoringTime = 7,
                         interimLookTime = c(1, 2, 3, 5, 7),
                         type1ErrorForResponse = 0.05,
                         type2ErrorForResponse = 0.01,
                         glrBoundarySidedness = "one", # one sided or two-sided
                         type1Error = 0.05,
                         type2Error = 0.10,
                         epsType1 = 1/3,
                         epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.6,
                       pdiffHyp=0.3,
```

```
                        theta = list(
                          alpha = log(.75),
                          beta = log(1.0),
                          gamma = log(0.61)),
                        baselineLambda = 0.35,
                        etaHyp = 0.25)
rngSeed <- 466553
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 18)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 3)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 3)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 22)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 2)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 4, 4, 6, 11); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run6 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                        p1 = 0.6,
                        pdiffHyp=0.3,
                        theta = list(
                          alpha = log(.5),
                          beta = log(1.0),
                          gamma = log(0.67)),
                        baselineLambda = 0.35,
```

```
                        etaHyp = 0.25)
rngSeed <- 1875543
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 17)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 2)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 2)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 23)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 1)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 4, 5, 9, 7); names(out) <- trialParameters$interimLookTime; out})
}
```

```
test.run7 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.5,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = log(.75),
                         beta = log(1.0),
                         gamma = log(0.47)),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 1093
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
```

```
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 5)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 3)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 3)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 22)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 1)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 3, 3, 5, 14); names(out) <- trialParameters$interimLookTime; out}
}
```

```
test.run8 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.3,
                       p1 = 0.5,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = 0,
                         beta = log(.75),
                         gamma = 0),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 117
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 9)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 23)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 8)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 6)
```

```
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 17)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 3)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 5, 7, 7, 6); names(out) <- trialParameters$interimLookTime; out})
}
```

```
test.run9 <- function() {
trialParameters <- list(minimumNumberOfEvents = 20,
                        minimumIncreaseInV = 0.2,
                        numberRecruitedEachYear = c(80, 120, 160, 160),
                        followupTime = 3,
                        adminCensoringTime = 7,
                        interimLookTime = c(1, 2, 3, 5, 7),
                        type1ErrorForResponse = 0.05,
                        type2ErrorForResponse = 0.01,
                        glrBoundarySidedness = "one", # one sided or two-sided
                        type1Error = 0.05,
                        type2Error = 0.10,
                        epsType1 = 1/3,
                        epsType2 = 1/3)
trueParameters <- list(p0 = 0.2,
                       p1 = 0.5,
                       pdiffHyp=0.3,
                       theta = list(
                         alpha = 0,
                         beta = log(.75),
                         gamma = 0),
                       baselineLambda = 0.35,
                       etaHyp = 0.25)
rngSeed <- 9872831
sp23Design <- generateSP23Design(trueParameters, trialParameters)
print(sp23Design)
trialHistory <- exploreSP23Design(sp23Design, numberOfSimulations=25, rngSeed=rngSeed)
result <- analyzeSP23Design(sp23Design, trialHistory)$designSummary
  checkEquals(result[["numberOfTimesH0RIsRejectedAtFirstLook"]], 16)
  checkEquals(result[["numberOfTimesH0RIsRejected"]], 25)
  checkEquals(result[["numberOfTimesStoppedForFutility"]], 3)
  checkEquals(result[["numberOfTimesH0SIsAccepted"]], 3)
  checkEquals(result[["numberOfTimesH0SIsRejected"]], 22)
  checkEquals(result[["numberOfTimesFutilityDecidedAtLastLook"]], 3)
  checkEquals(result[["numberOfTimesTrialEndedAtLook"]],
          {out <- c(0, 1, 4, 9, 11); names(out) <- trialParameters$interimLookTime; out}
}
```

References

[1] Tze Leung Lai, Philip W. Lavori, and Mei-Chiung Shih. Sequential design of phase ii-iii cancer trials. *Statistics in Medicine*, 31(18):1944–1960, 2012.

[2] Tze Leung Lai and Mei-Chiung Shih. Power, sample size and adaptation considerations in the design of group sequential clinical trials. *Biometrika*, 91(3):507, 2004.