gsDesign: An R Package for Designing Group Sequential Clinical Trials Version 2.0 Manual

Keaven M. Anderson
Merck Research Laboratories

August 15, 2009

Abstract

The gsDesign package supports group sequential clinical trial design. While there is a strong focus on designs using α- and β-spending functions, Wang-Tsiatis designs, including O'Brien-Fleming and Pocock designs, are also available. The ability to design with non-binding futility rules is an important feature to control Type I error in a manner acceptable to regulatory authorities.

The routines are designed to provide simple access to commonly used designs using default arguments. Standard, published spending functions are supported as well as the ability to write custom spending functions. A `gsDesign` class is defined and returned by the `gsDesign()` function. A plot function for this class provides a wide variety of plots: boundaries, power, estimated treatment effect at boundaries, conditional power at boundaries, spending function plots, expected sample size plot, and B-values at boundaries. Using function calls to access the package routines provides a powerful capability to derive designs or output formatting that could not be anticipated through a gui interface. This enables the user to easily create designs with features they desire, such as designs with minimum expected sample size.

In addition to straightforward group sequential design, the gsDesign package provides tools to effectively adapt clinical trials during execution. First, the spending function approach to design allows altering timing of analyses during the course of the trial. Information-based timing of analyses allows adaptation of sample size or number of events to ensure adequate power for a trial. Finally, gsDesign provides a routine that enable design adaptation using conditional power.

In summary, the intent of the gsDesign package is to easily create, fully characterize, and even optimize routine group sequential trial designs, as well as to provide a tool to derive and evaluate innovative designs.

Contents

1 Introduction

1.1 Overview

The gsDesign package is intended to provide a flexible set of tools for designing and analyzing group sequential trials. There are other adaptive methods that can also be supported using this underlying toolset. This manual is intended as an introduction to gsDesign. Many users may just want to apply basic, standard design methods. Others will be interested in applying the toolset to very ambitious adaptive designs. We try to give some orientation to each of these sets of users, and to distinguish between the material needed by each.

The remainder of this overview provides a quick review of topics covered in this manual. The introduction continues with some basic theory behind group sequential design to provide background for the routines. There is no attempt to fully develop the theory for statistical tests or for group sequential design in general since many statisticians will already be familiar with these and there are excellent texts available such as Jennison and Turnbull [9] and Proschan, Lan and Wittes [18].

This section continues with a simple outline of the main routines provided in the gsDesign package followed by motivational examples that will be used later in the manual. Basic sample size calculations for 2-arm binomial outcome trials using the `nBinomial()` function and 2-arm time-to-event endpoint trials using `nSurvival()` are shown, including an example of a non-inferiority trial. Both superiority and noninferiority trials are considered.

Further material is arranged by topic in subsequent sections. Section 2 provides a minimal background in asymptotic probability theory for group sequential testing. The basic calculations involve computing boundary crossing probabilities for correlated normal random variables. We demonstrate the `gsProbability()` routine to compute boundary crossing probabilities and expected sample size for group sequential designs

Setting boundaries for group sequential designs, particularly using spending functions is the main point of emphasis in the gsDesign package. Sections 3 through 7 of the manual present the design and evaluation of designs for group sequential trials using the `gsDesign()` routine.

Default parameters for `gsDesign()` are demonstrated for the motivational examples in Section 3. Basic computations for group sequential designs using boundary families and error spending are provided in Section 4. The primary discussion of Wang-Tsiatis [23] boundary families (*e.g.*, O'Brien-Fleming [15] and Pocock [17] designs) is provided here in Section 4.2.

Next we proceed to a short discussion in Section 5 of `gsDesign()` parameters for setting Type I and II error rates and the number and timing of analyses. This section also explains how to use a measure of treatment effect to size trials, with specific discussion of event-based computations for trials with time-to-event analyses.

The basics of standard spending functions are provided in Section 6. Subsections defining spending functions and spending function families are followed by a description of how to use built-in standard Hwang-Shih-DeCani [8] and power [10] spending functions in `gsDesign()`. Section 6.4 shows how to reset timing of interim analyses using `gsDesign()`.

The final section on spending functions is Section 7 which presents details of how spending functions are defined for `gsDesign()` and other advanced topics that will probably not be needed by many users. The section will be of use to those interested in investigational spending functions and optimized spending function choice. Recently published spending function families by Anderson and Clark [2] providing additional flexibility to standard one-parameter spending functions are detailed as part of a comprehensive list of built-in spending functions. This is followed by examples of how to derive optimal designs and how to implement new spending functions.

Next comes Section 8 on the basic analysis of group sequential trials. This includes computing stagewise and repeated p-values as well as repeated confidence intervals.

Conditional power and B-values are presented in Section 9. These are methods used for evaluating interim trends in a group sequential design, but may also be used to adapt a trial design at an interim analysis using the methods of Muller and Schaffer [14]. The routine `gsCP()` provides the basis for applying these adaptive methods.

We end with a discussion of Bayesian compution in Section 10. The gsDesign package can quite simply be used with decision theoretic methods to derive optimal designs. We also apply Bayesian computations to update the probability of succes a trial based on knowing a bound has not been crossed, but without knowledge of unblinded treatment results.

Future extenstions of the manual could further discuss implementation of information-based designs and additional adaptive design topics.

1.2 Quick start: installation and online help

This brief section is meant to get you up and going. Those who really do not like manuals may read just this section and then use the online help files for futher instruction.

The package comes in a binary format for a Windows platform in the file gsDesign-2.0.zip (may be updated to fix bugs in a file such as gsDesign-2.0.01.zip). This file includes a copy of this manual in the file gsDesignManual.pdf. Binaries are also available for OS/X. For other platforms the source code is in the file gsDesign-2.0.tar.gz (may be updated to fix bugs in a file such as gsDesign-2.0.01.tar.gz).

Following are basic instructions for installing the binary version on a Windows machine. It is assumed that a 'recent' version of R is installed. From the Windows interface of R, select the

Packages menu line and from this menu select Install packages from local zip files.... Browse to select gsDesign-2.0.zip. Once installed, you need to load the package by selecting the Packages menu line, selecting Load package... from this menu, and then selecting gsDesign. You are now ready to use the routines in the package. The most up-to-date version of this manual and the code is also available at `http://r-forge.r-project.org`.

Online help can be obtained by entering the following on the command line:

```
> help(gsDesign)
```

There are many help topics covered there which should be sufficient information to keep you from needing to use this document for day-to-day use or if you just generally prefer not using a manual. In the Window version of R, this brings up a "Contents" window and a documentation window. In the contents window, open the branch of documentation headed by "Package gsDesign: Titles." The help files are organized sequentially under this heading.

1.3 Installation qualification

This brief note is for more advanced users. Installation qualification routines are included in the package subdirectory `inst/unitTests`. While these tests were originally intended to run at the time the package is checked for an operating system, the initiating file `doRUnit.R` was removed from the `tests` directory so that the tests would not run automatically at CRAN where the duration of package checking is an issue due to the large number of packages there that are recompiled frequently. If the file `doRUnit.R` is be moved to the `tests` directory for the package prior to running `R CMD check gsDesign`, the tests run automatically.

1.4 The primary routines in the gsDesign package

As an overview to the R package, 3 R functions are supplied to provide basic computations related to designing and evaluating group sequential clinical trials:

1. The `gsDesign()` function provides sample size and boundaries for a group sequential design based on treatment effect, spending functions for boundary crossing probabilities, and relative timing of each analysis. Standard and user-specified spending functions may be used. In addition to spending function designs, the family of Wang-Tsiatis designs—including O'Brien-Fleming and Pocock designs—are also available.

2. The `gsProbability()` function computes boundary crossing probabilities and expected sample size of a design for arbitrary user-specified treatment effects, bounds, and interim analysis sample sizes.

3. The `gsCP()` function computes the conditional probability of future boundary crossing given a result at an interim analysis. The `gsCP()` function returns a value of the same type as `gsProbability()`.

The package design strategy should make its tools useful both as an everyday tool for simple group sequential design as well as a research tool for a wide variety of group sequential design problems. Both `print()` and `plot()` functions are available for both `gsDesign()` and `gsProbability()`. This should make it easy to incorporate design specification and properties into documents, as required.

The most extensive set of supportive routines enables design and evaluation of binomial trials. We use the Farrington and Manning [6] method for sample size estimation in `nBinomial()` and the

Table 1: Fixed design sample size possibilities for the CAPTURE trial by control group event rate and relative treatment effect.

Control	Event rate reduction		
rate	1/3	1/2	2/3
7.5%	2942	1184	594
10%	2158	870	438
15%	1372	556	282
20%	980	398	202

80% power, $\alpha = .05$, 2-sided

corresponding Miettinen and Nurminen [13] method for testing, confidence intervals and simulation. We also provide a basic Lachin and Foulkes [11] for sample size for survival studies. The examples we present apply these methods to group sequential trial design for binomial and time-to-event endpoints.

Functions are set up to be called directly from the R command line. Default arguments and output for `gsDesign()` are included to make initial use simple. Sufficient options are available, however, to make the routine very flexible.

Simple examples provide the best overall motivation for group sequential design. This manual does not attempt to comprehensively delineate all that the gsDesign package may accomplish. The intent is to include enough detail to demonstrate a variety of approaches to group sequential design that provide the user with a useful tool and the understanding of ways that it may be applied and extended. Examples that will reappear throughout the manual are introduced here.

1.5 The CAPTURE trial: binary endpoint example

The CAPTURE investigators [3] presented the results of a randomized trial in patients with unstable angina who required treatment with angioplasty, an invasive procedure where a balloon is inflated in one or more coronary arteries to reduce blockages. In the process of opening a coronary artery, the balloon can injure the artery which may lead to thrombotic complications. Standard treatment at the time the trial was run included treatment with heparin and aspirin before and during angioplasty to reduce the thrombotic complications such as the primary composite endpoint comprising myocardial infarction, recurrent urgent coronary intervention and death over the course of 30 days. This trial compared this standard therapy to the same therapy plus abciximab, a platelet inhibitor. While the original primary analysis used a logrank statistic to compare treatment groups, for this presentation we will consider the outcome binary. Approximately 15% of patients in the control group were expected to experience a primary endpoint, but rates from 7.5% to 20% could not be ruled out. There was an expectation that the experimental treatment would reduce incidence of the primary endpoint by at least 1/3, but possibly by as much as 1/2 or 2/3. Since a 1/3 reduction was felt to be conservative, the trial was planned to have 80% power. Given these various possibilities, the desirable sample size for a trial with a fixed design had over a 10-fold range from 202 to 2942; see Table 1.

The third line in the above table can be generated using the call

```
nBinomial(p1=.15, p2=.15 * c(2/3, 1/2, 1/3), beta=.2)
```

and rounding the results up to the nearest even number. The function `nBinomial()` in the `gsDesign` package is designed to be a flexible tool for deriving sample size for two-arm binomial trials for both superiority and non-inferiority. Type `help(nBinomial)` at the command prompt to see background

on sample size, simulation, testing and confidence interval routines for fixed (non-group sequential) binomial trials. These routines will be used with this and other examples throughout the manual.

1.6 A time to event endpoint in a cancer trial

As a second example we consider comparing a new treatment to a standard treatment for a cancer trial. Lachin and Foulkes [11] provide a method of computing sample size assuming the following distributions are known:

- the time to a primary endpoint in each treatment group,
- the time until dropout in each group,
- enrollment over time.

Statistical testing is performed using the logrank test statistic. The methods allow different assumptions in different strata. Enrollment time and total study duration are assumed fixed, and the sample size and number of events required during those periods, respectively, to achieve a desired power and Type I error are computed. Here we apply the simplest form of this method, assuming an expontential distribution in each case with no stratification. The routine `nSurvival` can be used to derive the sample size and number of events required. This routine works with failure rates rather than distribution medians or dropout rates per year. An exponential distribution with failure rate λ has cumulative probability of failure at or before time t of

$$F(t) = 1 - e^{-\lambda t}. \tag{1}$$

If the cumulative failure rate is known to be p_0 at time t_0 then the value of λ is

$$\lambda = -\ln(1 - p_0)/t_0. \tag{2}$$

We assume for the trial of interest that the primary endpoint is the time from randomization until the first of disease progression or death (progression free survival or PFS). Patients on the standard treatment are assumed to have an exponential failure rate with a median PFS of 6 months, yielding $\lambda_C = -\ln(.5)/6 = .1155$ with t measured in months. The trial is to be powered at 90% to detect a reduction in the hazard rate for PFS of 30% in the experimental group compared to standard treatment. This yields an experimental group failure rate of $\lambda_E = .7\lambda_C = .0809$. Patients are assumed to drop out at a rate of 5% per year of follow-up which implies an exponential rate $\eta = -\ln(.95)/12 = .00427$. Enrollment is assumed to be uniform over 30 months with patients followed for a minimum of 6 months, yielding a total study time of 36 months.

The function `nSurvival()` is included in the package to compute sample size using the Lachin and Foulkes [11] method. The code

```
x <- nSurvival(lambda.0=-log(.5) / 6, lambda.1=-log(.5) / 6 * .7,
        eta=-log(.95)/12, Tr=30 ,Ts=36, type="rr", entry="unif")
x$Sample.size
x$Num.events
```

shows that 418 patients and 330 events are required to obtain 90% power with a 2.5% one-sided Type I error. A major issue with this type of study is that many experimental cancer therapies have toxic side-effects and, at the same time, do not provide benefit. For such drugs, it is desirable to minimize the number of patients exposed to the experimental regimen and further to minimize the duration of exposure for those who are exposed. Thus, it is highly desirable to do an early evaluation of data to stop the trial if no treatment benefit is emerging during the course of the trial. Such an evaluation must be carefully planned to 1) avoid an unplanned impact on the power of the study, and 2) to allow a realistic assessment of the emerging treatment effect.

1.7 A non-inferiority study for a new drug

The `nBinomial()` function presented above was specifically designed to work for noninferiority trial design as well as superiority designs. We consider a new treatment that is to be compared to a standard that has a successful treatment rate of 67.7%. An absolute margin of 7% is considered an acceptable noninferiority margin. The trial is to be powered at 90% with 2.5% Type I error (one-sided) using methods presented by Farrington and Manning [6]. The function call `nBinomial(p1=.677, p2=.677, delta0=.07)` shows that a fixed sample size of 1874 is adequate for this purpose. There are some concerns about these assumptions, however. First, the control group event rate may be incorrect. As the following code using event rates from .55 to .75 demonstrates, the required sample size may range from 1600 to over 2100.

```
> p <- c(.55, .6, .65, .7, .75)
> ceiling(nBinomial(p1=p,p2=p,delta0=.07))
[1] 2117 2054 1948 1800 1611
```

More importantly, if the experimental group therapy does not work quite as well as control, there is a considerable dropoff in power to demonstrate non-inferiority. Thus, there may be value in planning an interim futility analysis to stop the trial if the success rate with experimental therapy is trending substantially worse than with control.

1.8 A diabetes outcomes trial example

Current regulatory standards for chronic therapies of diabetes require ensuring that a new drug in a treatment class does not have substantially inferior cardiovascular outcomes compared to an approved treatment or treatments [7]. While we do not claim the designs for this example presented here would be acceptable to regulators, the specifics of the guidance provide a nice background for the use of the `gsDesign` package to derive group sequential designs that fit a given problem. The initial reason for presenting this example is that there is likely to be a genuine public health interest in showing any of the following for the two treatment arms compared:

- The two treatment arms are similar (equivalence).
- One arm is similar to or better than the other (non-inferiority).
- Either arm is superior to the other (2-sided testing of no difference).

The example is somewhat simplified here. We assume patients with diabetes have a risk of a cardiovascular event of about 1.5% per year and a 15% dropout rate per year. If each arm has the same cardiovascular risk as the other, we would like to have 90% power to rule out a hazard ratio of 1.3 in either direction. Type I error if one arm has an elevated hazard ratio of 1.3 compared to the other should be 2.5%. The trial is to enroll in 2 years and have a minimum follow-up of 4 years, leading to a total study time of 6 years. The sample size routine `nSurvival()` is currently set up to have no treatment difference as the null hypothesis, whereas here we wish to use this as the alternative hypothesis. Thus, in using `nSurvival()` we switch the role of Type I error `alpha` and Type II error `beta`

```
x <- nSurvival(lambda.0=-log(.985), lambda.1=-log(.985) * 1.3, eta=-log(.85),
        alpha=.2, beta=.025, Tr=2, Ts=6, type="rr", entry="unif")
x$Sample.size
x$Num.events
```

Since the default in `nSurvival()` is 2-sided testing we have set `alpha=.2` to ensure there is a 10% probability of rejecting no difference when there is a hazard ratio of 1.3 or 1/1.3 for control versus experimental treatment. The above code suggests 10,800 patients should be enrolled and final

analysis conducted when 617 cardiovascular events have been observed. Generally, a confidence interval for the hazard ratio of experimental to control is used to express treatment differences at the end of this type of trial. A confidence interval will rule out the specified treatment differences consistently with testing if, for example, the same proportional hazards regression model is used for both the a Wald test and the corresponding confidence interval. The terminology of "control" and "experimental" is generally inappropriate when both therapies are approved. However, for this example it is generally the case that a new therapy is being compared to an established one and there may be some asymmetry when considering the direction of inference. Various questions arise concerning early stopping in a trial of this nature:

- While it would be desirable to stop early if the new therapy has a significantly lower cardiovascular event rate, a minimum amount of follow-up may be valuable to ensure longer-term safety and general acceptance of the results.
- If a trend emerges in favor of the experimental treatment, it will likely be possible to demonstrate non-inferiority prior to being able to demonstrate superiority. If the trial remains blinded until superiority is demonstrated or until the final planned analysis, full acceptance of a useful new therapy may be delayed. As noted above, the value of long-term safety data may be more important than an early stop based on "short-term" endpoint.
- From a sponsor's standpoint, it may be desirable to stop the trial if it becomes futile to demonstrate the experimental therapy is non-inferior to control; that is, there is an interim trend favoring control. However, if both treatment groups represent marketed products then from a public health standpoint it may be desirable to continue the trial to demonstrate a statistically significant advantage for the control treatment.

2 Group sequential testing

While the primary purpose of the `gsDesign` package is to design group sequential trials, computing boundary crossing probabilities is the essential first step. Thus, we begin a summary of theory and computation of boundary crossing probabilities.

2.1 Distributional assumptions

We illustrate the distribution theory with a sequence of normal random variates. Let X_1, X_2,... be independent and identically distributed normal random variables with mean θ and variance 1. For some positive integer k, let $n_1 < n_2 ... < n_k$ represent fixed sample sizes where data will be analyzed and inference surrounding θ will be examined. This is referred to as a group sequential design. The first $k-1$ analyses are referred to as interim analyses, while the k^{th} analysis is referred to as the final analysis. For $i = 1, 2, ...k$ consider the test statistic

$$Z_i = \sum_{j=1}^{n_i} X_j / \sqrt{n_i}.$$

The variance of $\bar{X}_i$ is $1/n_i$ and the corresponding statistical information is its reciprocal: $I_i = n_i$, $i = 1, 2, \ldots, k$. The test statistics Z_1, Z_2,...,Z_k follow a multivariate normal distribution with, for $1 \leq j \leq i \leq k$,

$$E\{Z_i\} = \theta\sqrt{I_i} \tag{3}$$

$$Cov(Z_j, Z_i) = \sqrt{I_j / I_i} \tag{4}$$

Jennison and Turnbull [9] refer to (3) and (4) as the canonical form and present several types of outcomes for which test statistics for comparing treatment groups take this form asymptotically. Specific examples in this manual consider 2-arm trials with binary or time-to-event outcomes. Note that when $\theta = 0$, the mulitivariate normal distribution expressed in (3) and (4) depends only on I_i/I_k, $i = 1, 2, \ldots, k-1$.

Computational methods for the `gsDesign` package related to the above multivariate normal distribution are described in Chapter 19 of Jennison and Turnbull [9] and are not provided here. Note that other software programs such as EAST and the University of Wisconsin software use this distributional assumption for group sequential design computations.

2.2 Hypotheses and testing

We assume that the primary test the null hypothesis H_0: $\theta = 0$ against the alternative H_1: $\theta = \delta$ for a fixed $\delta > 0$. The value of θ will be referred to as a treatment effect here since that is what clinical trials are normally set up to examine. We have arbitrarily assume that $\theta > 0$ represents a treatment benefit and will refer to this case as a positive treatment effect. We assume further that there is interest in stopping early if there is good evidence to reject one hypothesis in favor of the other. For $i = 1, 2, \ldots, k-1$, interim cutoffs $a_i < b_i$ are set; final cutoffs $a_k \leq b_k$ are also set. For $i = 1, 2, \ldots, k$, the trial is stopped at analysis i to reject H_0 if $a_j < Z_j < b_j$, $j = 1, 2, \ldots, i-1$ and $Z_i \geq b_i$. If the trial continues until stage i, H_0 is not rejected at stage i, and $Z_i \leq a_i$ then H_1 is rejected in favor of H_0, $i = 1, 2, \ldots, k$. Thus, $3k$ parameters define a group sequential design: a_i, b_i, and I_i, $i = 1, 2, \ldots, k$. Note that if $a_k < b_k$ there is the possibility of completing the trial without rejecting H_0 or H_1. We will generally restrict $a_k = b_k$ so that one hypothesis is rejected.

2.3 Boundary crossing probabilities: `gsProbability()`

2.3.1 One-sided testing

We begin with a one-sided test. In this case there is no interest in stopping early for a lower bound and thus $a_i = -\infty$, $i = 1, 2, \ldots, k$. The probability of first crossing an upper bound at analysis i, $i = 1, 2, \ldots, k$, is

$$\alpha_i^+(\theta) = P_\theta\{\{Z_i \geq b_i\} \cap_{j=1}^{i-1} \{Z_j < b_j\}\} \tag{5}$$

The Type I error is the probability of ever crossing the upper bound when $\theta = 0$. The value $\alpha_i^+(0)$ is commonly referred to as the amount of Type I error spent at analysis i, $1 \leq i \leq k$. The total upper boundary crossing probability for a trial is denoted in this one-sided scenario by

$$\alpha^+(\theta) \equiv \sum_{i=1}^{k} \alpha_i^+(\theta) \tag{6}$$

and the total Type I error by $\alpha^+(0)$. Assuming $\alpha^+(0) = \alpha$ the design will be said to provide a one-sided group sequential test at level α.

As an example, assume $k = 3$, $n_i = 100 \times i$, and $b_i = \Phi^{-1}(.975) = 1.96$, $i = 1, 2, 3$. Thus, we are testing 3 times at a nominal .025 level at three equally spaced analyses. The function `gsProbability()` is designed to compute the probability of crossing the upper boundary at each analysis as follows. `gsProbability()` requires a lower bound; we let $a_i = -20$, $i = 1, 2, 3$ to effectively make the probability of crossing a lower bound 0.

```
> x<-gsProbability(k=3, theta=0, n.I=c(100, 200, 300), a=array(-20,3),
+ b=array(qnorm(.975),3))
> x
```

```
             Lower bounds   Upper bounds
  Analysis  N  Z   Nominal p  Z   Nominal p
         1 100 -20         0 1.96     0.025
         2 200 -20         0 1.96     0.025
         3 300 -20         0 1.96     0.025

Boundary crossing probabilities and expected sample size assuming any cross
stops the trial

Upper boundary (power or Type I Error)
          Analysis
  Theta     1      2      3  Total  E{N}
      0 0.025 0.0166 0.0121 0.0536 293.3

Lower boundary (futility or Type II Error)
          Analysis
  Theta 1 2 3 Total
      0 0 0 0     0
```

In the table at the top of the output, we see the sample size at each test along with the upper boundary of 1.96 that is used for each test and its associated nominal significance level of .025. The lower part of the output shows the actual probabilities of first crossing the upper boundary at each test. Recall first that since `theta=0` we would get the same boundary crossing probabilities whether the statistical information `n.I=c(1,2,3)` or `n.I=c(100,200,300)`. That is, the boundary crossing probabilities under the null hypothesis depend only on the correlation structure of Z_i, $i = 1, 2, \ldots, k$. When the nominal .025 upper bound was used repeatedly at 3 equally spaced intervals in group sequential testing, the first test had a probability of $\alpha_1^+(0) = .025$ of crossing the upper bound. The probability computed for crossing the upper bound at the second test excludes cases where the boundary was crossed at the first test and is thus $\alpha_2^+(0) = .0166 < .025$. The total probability of crossing the upper bound using all 3 tests is $\alpha^+(0) = .053 > .025$ which illustrates the multiplicity issue of testing multiple times at the overall significance level normally specified for a test. The expected sample size indicated in the lower part of the output will be discussed in detail in section 2.4 below.

A Bonferroni adjustment maintains a Type I error of $\leq .025$. For $i = 1, 2, 3$ this would use the upper bound $b_i = \Phi^{-1}(1 - .025/3)$. Substituting `qnorm(1-.025/3)` for `qnorm(.975)` in the above call to `gsProbability()` yields an upper bound of 2.39 and total Type I error of .0192. Thus, the Bonferroni adjustment is overly conservative for this case. We will return to this example later to show how to set equal bounds that yield a total Type I error of .025.

In the above code, the call to `gsProbability()` returned a value into `x` which was then printed. The command `class(x)` following the above code will indicate that `x` has class `gsProbability`. The elements of this class are displayed as follows:

```
> summary(x)
      Length Class  Mode
k     1      -none- numeric
theta 1      -none- numeric
n.I   3      -none- numeric
lower 2      -none- list
upper 2      -none- list
en    1      -none- numeric
r     1      -none- numeric
```

Briefly, `k` is k, `theta` a vector of θ-values, and `n.I` is a vector containing I_i, $i = 1, 2, \ldots, K$. The value in r is a positive integer input to `gsProbability` that is used to define how fine of a grid is used for numerical integration when computing boundary crossing probabilities; this is the same as input and will normally not be changed from the default of 18. The value of `en` will be discussed below in Section 2.4. This leaves the lists `lower` and `upper`, which have identical structure. We examine `upper` by

```
> x$upper
$bound
[1] 1.959964 1.959964 1.959964

$prob
           [,1]
[1,] 0.02500000
[2,] 0.01655891
[3,] 0.01207016
```

to see that it contains a vector `bound` which contains the values for b_i and upper boundary crossing probabilities in `prob[i,j]` for analysis i and the θ-value in `theta[j]` for `i=1,2,...,k` and `j` from 1 to the length of `theta`. Further documentation is in the online help file displayed using `help(gsProbability)`.

2.3.2 Two-sided testing

With both lower and upper bounds for testing and any real value θ representing treatment effect we denote the probability of crossing the upper boundary at analysis i without previously crossing a bound by

$$\alpha_i(\theta) = P_\theta\{\{Z_i \geq b_i\} \cap_{j=1}^{i-1} \{a_j < Z_j < b_j\}\}, \tag{7}$$

$i = 1, 2, \ldots, k$. The total probability of crossing an upper bound prior to crossing a lower bound is denoted by

$$\alpha(\theta) \equiv \sum_{i=1}^{k} \alpha_i(\theta). \tag{8}$$

Next, we consider analogous notation for the lower bound. For $i = 1, 2, \ldots, k$ denote the probability of crossing a lower bound at analysis i without previously crossing any bound by

$$\beta_i(\theta) = P_\theta\{\{Z_i \leq a_i\} \cap_{j=1}^{i-1} \{a_j < Z_j < b_j\}\}. \tag{9}$$

For symmetric testing for analysis i we would have $a_i = -b_i$, $\beta_i(0) = \alpha_i(0)$, $i = 1, 2, \ldots, k$. The total lower boundary crossing probability in this case is written as $\beta(\theta) = \sum_{i=1}^{k} \beta_i(\theta)$. The total lower boundary crossing probability for a trial is denoted by

$$\beta(\theta) \equiv \sum_{i=1}^{k} \beta_i(\theta). \tag{10}$$

To extend the one-sided example using repeated testing at a .025 level to two-sided testing at the .05 level, try the commands

```
b<-array(qnorm(.975),3)
x2<-gsProbability(k=3, theta=0, n.I=c(100, 200, 300), a=-b, b=b)
x2
```

The fact that a lower bound can be crossed before crossing an upper bound means that after the first interim analysis the upper boundary crossing probability here should be less than it was for the one-sided computation performed previously. To examine this further, we print the upper boundary crossing probability at each analysis for the above one-sided and two-sided examples, respectively, to see that there is a small difference:

```
> x$upper$prob
            [,1]
[1,] 0.02500000
[2,] 0.01655891
[3,] 0.01207016
> x2$upper$prob
            [,1]
[1,] 0.02500000
[2,] 0.01655890
[3,] 0.01206929
```

Group sequential designs most often employ more stringent bounds at early interim analyses than later. We modify the above example to demonstrate this.

```
> b<-qnorm(.975)/sqrt(c(1,2,3)/3)
> b
[1] 3.394757 2.400456 1.959964
> x2b<-gsProbability(k=3, theta=0, n.I=c(100, 200, 300), a=-b,b=b)
> x2b
               Lower bounds   Upper bounds
  Analysis  N    Z   Nominal p  Z   Nominal p
         1 100 -3.39    0.0003 3.39    0.0003
         2 200 -2.40    0.0082 2.40    0.0082
         3 300 -1.96    0.0250 1.96    0.0250

Boundary crossing probabilities and expected sample size assuming any cross
stops the trial

Upper boundary (power or Type I Error)
          Analysis
  Theta     1     2      3  Total  E{N}
      0 3e-04 0.008 0.0195 0.0279 298.3

Lower boundary (futility or Type II Error)
          Analysis
  Theta     1     2      3  Total
      0 3e-04 0.008 0.0195 0.0279
```

By setting the interim boundaries to be substantially higher than $\Phi^{-1}(.975) = 1.96$ we have drastically reduced the excess Type I error caused by multiple testing while still testing at the nominal .05 (2-sided) level at the final analysis. Thus, minimal adjustment to the final boundary should be required when employing such a strategy. Precise control of Type I error when using either equal bounds or adjusting relative sizes of bounds using the square root of sample size is discussed further in section 4.2.

2.4 Expected sample size

We denote the sample size at analysis i by n_i, $i = 1, 2, \ldots, k$ and the sample size at the time a boundary is crossed by N. The average sample number (ASN) or expected sample size is defined by

$$\text{ASN}(\theta) = E_\theta\{N\} = \sum_{i=1}^{k} n_i \times (\alpha_i(\theta) + \beta_i(\theta)). \tag{11}$$

Values of ASN(θ) corresponding to θ-values input to `gsProbability` in `theta` are output in the vector `en` returned by that function. In the one- and two-sided examples above we only had a single element 0 in `theta` and the expected sample sizes rounded to 293 and 286, respectively; these were labeled `E{N}` in the printed output. Since the probability of crossing a boundary at an interim analysis was small, the trial usually proceeds to the final analysis with 300 observations. We consider an additional θ-value to demonstrate that the average sample number can be substantially smaller than the maximum sample size:

```
> x2c<-gsProbability(k=3, theta=c(0,.3), n.I=c(100, 200, 300), a=-b, b=b)
> x2c
               Lower bounds   Upper bounds
  Analysis  N    Z   Nominal p  Z   Nominal p
         1 100 -3.39    0.0003 3.39    0.0003
         2 200 -2.40    0.0082 2.40    0.0082
         3 300 -1.96    0.0250 1.96    0.0250

Boundary crossing probabilities and expected sample size assuming any cross stops the trial

Upper boundary (power or Type I Error)
          Analysis
  Theta      1      2      3  Total  E{N}
    0.0 0.0003 0.0080 0.0195 0.0279 298.3
    0.3 0.3465 0.6209 0.0320 0.9994 168.6

Lower boundary (futility or Type II Error)
          Analysis
  Theta     1     2      3  Total
    0.0 3e-04 0.008 0.0195 0.0279
    0.3 0e+00 0.000 0.0000 0.0000
```

Thus, assuming a positive treatment effect, the average sample number was approximately 169 compared to almost 300 when there was no treatment difference.

3 Applying the default group sequential design

3.1 Default parameters

We are now prepared to demonstrate derivation of group sequential designs using default parameters with the `gsDesign()` function. Along with this, we discuss the `gsDesign` class returned by `gsDesign()` and its associated standard print and plot functions. We then apply this default group sequential design to each of our motivational examples. The main parameters in `gsDesign()` will be explained in more detail in sections 4 through 6.

The main parameter defaults that you need to know about are as follows:

1. Overall Type I error $\alpha = 0.025$ (one-sided)

2. Overall Type II error $\beta = 0.1$ (Power = 90%)

3. Two interim analyses equally spaced at 1/3 and 2/3 of the way through the trial plus the final analysis (`k=3`)

4. Asymmetric boundaries, which means we may stop the trial for futility or superiority at an interim analysis.

5. β-spending is used to set the lower stopping boundary. This means that the spending function controls the incremental amount of Type II error at each analysis, $\beta_i(\delta)$, $i = 1, 2, \ldots, K$.

6. Non-binding lower bound. Lower bounds are sometimes considered as guidelines, which may be ignored during the course of the trial. Since Type I error is inflated if this is the case, regulators often demand that the lower bounds be ignored when computing Type I error. That is, Type I error is computed using $\alpha^+(\theta)$ from (5) and (6) rather than $\alpha(\theta)$ from (7) and (8).

7. A Hwang-Shih-DeCani spending functions with $\gamma = -4$ for the upper bound and $\gamma = -2$ for the lower bound. This provides a conservative, O'Brien-Fleming-like superiority bound and a less conservative lower bound. Spending functions will be discussed in detail in 6.

3.2 Sample size ratio for a group sequential design compared to a fixed design.

In section 2 and its subsections we gave distributional assumptions, defined testing procedures and denoted probabilities for boundary crossing. Consider a trial with a fixed design (no interim analyses) with power 100(1–β) and level α (1-sided). Denote the sample size as N_{fix} and statistical information for this design as I_{fix}. For a group sequential design as noted above, we denote the information ratio (inflation factor) comparing the information planned for the final analysis of a group sequential design compared to a fixed design as

$$r = I_k/I_{fix} = n_k/N_{fix}. \tag{12}$$

This ratio is independent of the θ-value δ for which the trial is powered as long as the information (sample size) available at each analysis increases proportionately with I_{fix} and the boundaries for the group sequential design remain unchanged; see, for example, Jennison and Turnbull [9]. Because of this, the default for `gsDesign()` is to print the sample size ratios $r_i = I_i/I_k$, $i = 1, 2, \ldots, k$ rather than an actual sample size at each interim analysis. This is implemented through by the default value of 1 for the input parameter `n.fix`. We demonstrate in the following subsections how to set `n.fix` to apply to our motivating examples.

3.3 The default call to `gsDesign()`

We begin with the call `x <- gsDesign()` to generate a design using all default arguments. The next line prints a summary of `x`; this produces the same effect as `print(x)` or `print.gsDesign(x)`. Note that while the total Type I error is 0.025, this assumes the lower bound is ignored if it is crossed; looking lower in the output we see the total probability of crossing the upper boundary at any analysis when the lower bound stops the trial is 0.0233. Had the option `x <- gsDesign(test.type=3)` been run, both of these numbers would assume the trial stops if the lower bound stopped and thus would both be 0.025.

```
> x <- gsDesign()
> x
Asymmetric two-sided group sequential design with 90 % power and 2.5
Type I Error.
Upper bound spending computations assume trial continues if lower bound is
crossed.

          Sample
           Size    ----Lower bounds----  ----Upper bounds-----
  Analysis Ratio*   Z   Nominal p Spend+  Z   Nominal p Spend++
         1  0.357 -0.24    0.4057 0.0148 3.01    0.0013  0.0013
         2  0.713  0.94    0.8267 0.0289 2.55    0.0054  0.0049
         3  1.070  2.00    0.9772 0.0563 2.00    0.0228  0.0188
     Total                        0.1000                0.0250
+ lower bound beta spending (under H1): Hwang-Shih-DeCani spending function
  with gamma = -2
++ alpha spending: Hwang-Shih-DeCani spending function with gamma = -4
* Sample size ratio compared to fixed non-group sequential design

Boundary crossing probabilities and expected sample size assuming any cross
stops the trial

Upper boundary (power or Type I Error)
          Analysis
   Theta      1      2      3  Total   E{N}
  0.0000 0.0013 0.0049 0.0171 0.0233 0.6249
  3.2415 0.1412 0.4403 0.3185 0.9000 0.7913

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2      3  Total
  0.0000 0.4057 0.4290 0.1420 0.9767
  3.2415 0.0148 0.0289 0.0563 0.1000

> plot(x)
```

Above we have seen standard output for `gsDesign()`. To access individual items of information about what is returned from the above, use `summary(x)` to list the elements of `x`. Type `help(gsDesign)` to get full documentation of the class `gsDesign` returned by the `gsDesign()` function. To view an individual element of `x` type, for example, `x$delta`. Other elements of `x` can be accessed in the same way, and we will use these to display aspects of designs in further examples. Of particular interest are the elements `upper` and `lower`. These are both objects containing multiple variables concerning the upper and lower boundaries and boundary crossing probabilities. Type `summary(x$upper)` to show what these variables are. The upper boundary can be shown with the command `x$upper$bound`. For additional plots, enter `plot(x, plottype=2))` for a power plot. The argument `plottype` can run from 1 (the default) to 7. The options not already noted plot effect sizes at boundaries (`plottype=3`), conditional power at boundaries (`plottype=4`)), α- and β-spending functions (`plottype=5`)), expected sample size by underlying treatment difference (`plottype=6`), and B-values at boundaries (`plottype=7`).

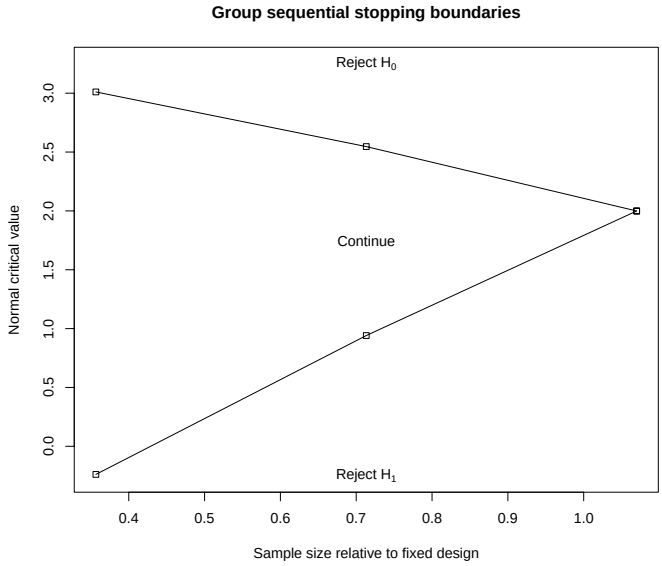

Figure 1: Default plot for gsDesign object

3.4 Applying the default design to the CAPTURE example

The sample size ratios from (12) can be obtained as follows:

```
> x <- gsDesign()
> x$n.I
[1] 0.3566277 0.7132554 1.0698832
```

These will be applied to each of our examples. Recall from the CAPTURE trial that we had a binomial outcome and wished to detect a reduction in the primary endpoint from a 15% event rate in the control group to a 10% rate in the experimental group. While we consider 80% power elsewhere, we stick with the default of 90% here. A group sequential design with 90% power and 2.5% Type I error has the same bounds as shown previously. The sample size at each analysis is obtained as follows (continuing the code just above):

```
> n.I <- nBinomial(p1 = .15, p2 = .1)
> n.I
1834.641
> n.I * x$n.I
[1]  654.2839 1308.5679 1962.8518
```

Rounding up to an even number in each case, we see from the above that a while a fixed design requires 1836 patients, employing the default group sequential design inflates the sample size requirement to 1964. Interim analyses would be peformed after approximately 654 and 1308 patients.

The group sequential design can be derived directly by replacing the input parameter `n.fix` with the sample size from a fixed design trial as follows:

```
> n.I <- nBinomial(p1 = .15, p2 = .1)
> x <- gsDesign(n.fix = n.I)
> x$n.I
[1]  654.2839 1308.5678 1962.8518
```

Printing this design now replaces the sample size ratio with the actual sample sizes at each analysis. The only other difference from the design above in Section 3 is the second value of `theta` below.

```
> x
Asymmetric two-sided group sequential design with 90 % power and
2.5 % Type I Error.
Upper bound spending computations assume trial continues if lower
bound is crossed.

                 ----Lower bounds----  ----Upper bounds-----
  Analysis   N    Z   Nominal p Spend+  Z   Nominal p Spend++
         1  655 -0.24    0.4057 0.0148 3.01    0.0013  0.0013
         2 1309  0.94    0.8267 0.0289 2.55    0.0054  0.0049
         3 1963  2.00    0.9772 0.0563 2.00    0.0228  0.0188
     Total                      0.1000                 0.0250
+ lower bound beta spending (under H1): Hwang-Shih-DeCani spending
function with gamma = -2
++ alpha spending: Hwang-Shih-DeCani spending function with gamma = -4

Boundary crossing probabilities and expected sample size assuming
any cross stops the trial

Upper boundary (power or Type I Error)
          Analysis
   Theta      1      2      3  Total   E{N}
  0.0000 0.0013 0.0049 0.0171 0.0233 1146.4
  0.0757 0.1412 0.4403 0.3185 0.9000 1451.7

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2      3  Total
  0.0000 0.4057 0.4290 0.1420 0.9767
  0.0757 0.0148 0.0289 0.0563 0.1000
```

3.5 Applying the default design to the noninferiority example

The fixed noninferiority design for a binomial comparison is the same as above, only changing the `nBinomial()` call to

```
> n.fix <- nBinomial(p1=.677, p2=.677, delta0=.07)
> ceiling(gsDesign(n.fix = n.fix)$n.I)
[1]  668 1336 2004
>
```

Testing at each analysis can be peformed using the Miettinen and Nurminen [13] method. Simulation to verify the normal approximation is adequate for comparing binomial event rates can be performed using the functions `simBinomial` and `testBinomial`.

3.6 Applying the default design to the cancer trial example

For trials with time-to-event outcomes, the variable `n.fix` in `gsDesign()` needed is the number of events from a fixed design trial. The reader may wish to refer to Jennison and Turnbull [9] for further background; we also discuss distributional assumptions further in Section 5.3.2. We begin with the code from the fixed design trial for the cancer trial example from 1.6. Next, we call to `gsDesign()` with `n.fix` equal to the number of events for a fixed trial design. The value `ssratio`, the sample size ratio at each analysis compared to the fixed design sample size is then shown. Note that the values are the same as shown in the first output of this example above. The inflation in the total sample size is the same as for the number of events required; that is, the sample size required for a group sequential design with the default interim analysis plan is inflated to 595 (or 596 for an even number) from 557 (or 558) from that of a fixed design by multiplying by 1.07.

```
> x <- nSurvival(lambda.0=log(2)/6,lambda.1=log(2)/6*.7,eta=-log(.95),
+         Tr=30,Ts=36,type="rr",entry="unif")
> ceiling(x$Sample.size)
[1] 557
> ceiling(x$Num.events)
[1] 330
> y <- gsDesign(n.fix=x$Num.events)
> ceiling(y$n.I)
[1] 118 236 353
> ssratio <- (y$n.I / x$Num.events)
> ssratio
[1] 0.3566277 0.7132554 1.0698831
> ceiling(ssratio[3] * x$Sample.size)
[1] 595
```

3.7 Using `gsProbability()` following `gsDesign()`

We reconsider the default design and obtain the properties for a larger set of θ values than in the standard printout for `gsDesign()` shown previously. The first two lines of code below demonstrates a group sequential design generated by `gsDesign()` can be input to `gsProbability()` to obtain boundary crossing probabilities for an extended set of parameter values. The `theta` values in the output make more sense in this case when they are computed relative to the effect size `x$delta` for which the trial is powered; this is more easily seen in the plot of expected sample size shown here since the x-axis uses this scale. Note that the y-axis shows the expected sample size relative to a fixed design trial when the sample size ratio is computed; if we had input a fixed design sample size, the y-axis would show the actual expected sample size. Note further that the `plot()` function for the `gsDesign` class (used here) is an extension of the standard `plot()` function, and thus allows use of many of its parameters, such as line width (`lwd`), line type (`lty`), plot titles and axis labels. This plot demonstrates the ability of a group sequential design to appropriate adapt sample size to come to an appropriate conclusion depending on the true treatment effect. If the effect size is twice that for which the trial is powered, the expected sample size is about 40% of that for a fixed design, compared to over 80% in some cases when the true treatment effect is between the hypothesized values for the efficacy parameter θ.

```
> x <- gsDesign()
> y <- gsProbability(theta=x$delta*seq(0, 2, .25), d=x)
> y
Asymmetric two-sided group sequential design with 90 % power and
```

```
2.5\% Type I Error.
Upper bound spending computations assume trial continues if lower bound is
crossed.

           Sample
            Size    ----Lower bounds----  ----Upper bounds-----
  Analysis Ratio*   Z   Nominal p Spend+  Z   Nominal p Spend++
         1  0.357 -0.24    0.4057 0.0148 3.01    0.0013  0.0013
         2  0.713  0.94    0.8267 0.0289 2.55    0.0054  0.0049
         3  1.070  2.00    0.9772 0.0563 2.00    0.0228  0.0188
     Total                         0.1000                 0.0250
+ lower bound beta spending (under H1): Hwang-Shih-DeCani spending function
with gamma = -2
++ alpha spending: Hwang-Shih-DeCani spending function with gamma = -4
* Sample size ratio compared to fixed non-group sequential design

Boundary crossing probabilities and expected sample size assuming any cross
stops the trial

Upper boundary (power or Type I Error)
          Analysis
   Theta      1      2      3  Total   E{N}
  0.0000 0.0013 0.0049 0.0171 0.0233 0.6249
  0.8104 0.0058 0.0279 0.0872 0.1209 0.7523
  1.6208 0.0205 0.1038 0.2393 0.3636 0.8520
  2.4311 0.0595 0.2579 0.3636 0.6810 0.8668
  3.2415 0.1412 0.4403 0.3185 0.9000 0.7913
  4.0519 0.2773 0.5353 0.1684 0.9810 0.6765
  4.8623 0.4574 0.4844 0.0559 0.9976 0.5701
  5.6727 0.6469 0.3410 0.0119 0.9998 0.4868
  6.4830 0.8053 0.1930 0.0016 1.0000 0.4266

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2      3  Total
  0.0000 0.4057 0.4290 0.1420 0.9767
  0.8104 0.2349 0.3812 0.2630 0.8791
  1.6208 0.1138 0.2385 0.2841 0.6364
  2.4311 0.0455 0.1017 0.1718 0.3190
  3.2415 0.0148 0.0289 0.0563 0.1000
  4.0519 0.0039 0.0054 0.0097 0.0190
  4.8623 0.0008 0.0006 0.0009 0.0024
  5.6727 0.0001 0.0001 0.0000 0.0002
  6.4830 0.0000 0.0000 0.0000 0.0000
> plot(y, plottype=6, lty=2, lwd=3)
```

4 Deriving group sequential designs

There are many ways to specify a group sequential design to obtain a desired power and Type I error. For planning purposes, the number, k and relative timing $0 < t_1 < ... < t_k = 1$ of interim

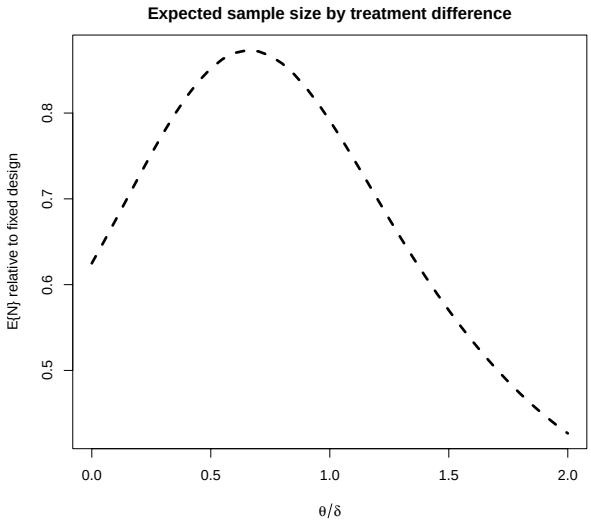

Figure 2: Average sample size plot

analyses are fixed. Given these values, there are two general approaches to deriving boundaries for a group sequential trial:

- **The error spending approach**. Specify boundary crossing probabilities at each analysis and derive a sample size and boundary values based on these values. This is most commonly done with the error spending function approach proposed by Lan and DeMets [12], which is discussed at some length in Section 6. We present this method in brief in Section 4.1 and follow this with simple examples.
- **The boundary family approach.** Specify how big boundary values should be relative to each other and adjust these relative values by a constant multiple to control overall error rates. Sample size adjustment is also part of this derivation. The commonly applied boundary family approach uses the Wang-Tsiatis [23] family which includes bounds by Pocock [17] and O'Brien and Fleming [15]. This will be discussed in Section 4.2.

4.1 Boundary derivation using boundary crossing probabilities

4.1.1 Types of error probabilities used: `test.type`

Before starting a discussion of spending functions, different methods of computing Type I error are discussed. Boundary crossing probabilities for upper bounds may be specified to `gsDesign()` using either $\alpha_i(0)$ or $\alpha_i^+(0)$, $i = 1, 2, \ldots, k$. In the first case, it is assumed that a trial stops when either a lower or upper bound is crossed and the only Type I error occurs when the first time a boundary is crossed it is an upper bound. In the second case, it is assumed that a lower bound may be ignored if crossed and the Type I error is the probability of ever crossing an upper bound if the trial is never stopped for crossing a lower bound. As we have seen, the difference between these boundary crossing probabilities may be small. The differences can be meaningful, however, when agressive futility bounds are employed to require, say, an early positive treatment effect trend.

For lower bounds, either $\beta_i(\delta)$ or $\beta_i(0)$, $i = 1, 2, \ldots, k$, may be specified.

Table 2: Boundary crossing probabilities used to set boundaries in `gsDesign()` by `test.type`.

`test.type`	Upper bound	Lower bound
1	$\alpha_i^+(0)$	None
2	$\alpha(0)$	$\beta_i(0)$
3	$\alpha_i(0)$	$\beta_i(\delta)$
4	$\alpha_i^+(0)$	$\beta_i(\delta)$
5	$\alpha(0)$	$\beta_i(0)$
6	$\alpha^+(0)$	$\beta_i(0)$

Sample size and boundaries that have appropriate boundary crossing probabilities and power are derived numerically using computational methods given in detail in Chapter 19 of Jennison and Turnbull [9]. The `gsDesign()` parameter `test.type` specifies which boundary crossing probabilities are used as outlined in Table 2.

For `test.type`=1, 2 and 5, boundaries can be computed in a single step just by knowing the cumulative proportion of the final planned statistical information (sample size/number of events) at each analysis that is specified using the `timing` input variable. For `test.type`=6, the upper and lower boundaries are computed separately and independently using these same methods. For `test.type`=1, 2, 5 or 6 the total sample size is then set to obtain the desired power under the alternative hypothesis by using a root finding algorithm.

For `test.type`=3 and 4 sample size and bounds are set simultaneously using an iterative algorithm. This computation is slightly more complex than the above. This does not make any noticeable difference in normal use of the `gsDesign()`. However, for user-developed routines that require repeated calls to `gsDesign()` (*e.g.*, finding an optimal design), there may be noticeably slower performance when test.type=3 or 4 is used.

4.1.2 Specifying boundary crossing probabilities in `gsDesign()`

We use the CAPTURE example, working with the desired 80% power ($\beta = .2$) to demonstrate deriving bounds with specified boundary crossing probabilities. For simplicity, we will let $\alpha_i^+(0) = .025/4$ and $\beta_i(\delta) = .2/4$, $i = 1, 2, 3, 4$. Setting the `gsDesign()` parameters `sfu` and `sfl` to `sfLinear`, the vector `p` below is used to equally allocate the boundary crossing probabilities for each analysis. Note that `sfLinear()` requires an even number of elements in `param`. The first half specify the timepoints using an increasing set of values strictly between 0 and 1 to indicate the proportion of information at which the spending function is specified. The second half specify the proportion of total error spending at each of these points. (Aside: those interested in plotting with special characters note the special handling of the character + in the argument `main` to `plot()`.)

```
# Cumulative proportion of spending planned at each analysis
# In this case, this is also proportion of final observations at each interim
p <- c(.25, .5, .75)
t <- c(.25,.5,.75)
# Cumulative spending intended at each analysis (for illustration)
p * 0.025
n.fix <- nBinomial(p1=.15, p2=.1, beta=.2)
x <- gsDesign(k=4, n.fix=n.fix, beta=.2, sfu=sfLinear, sfupar=c(t,p),
              sfl=sfLinear, sflpar=c(t,p))
plot(x, main=expression(paste("Equal ", alpha[i]^{"+"}, (0), " and ",
beta[i](delta), " for each analysis")))
```

```
x
```

The printed output from the above is shown below and a plot of the derived boundaries is in Figure 3. The columns labeled `Spend+` and `Spend++` show the values $\beta_i(\delta)$ and $\alpha_i(0)$, respectively, are equal for each analysis, $i = 1, 2, 3, 4$. The nominal p-values for the upper bound increase and thus the bounds themselves decrease for each analysis. That equal error probabilities results in unequal bounds is because of the correlation between the test statistics used for analysis that was indicated in (4). Note that the requirement of 1372 patients for the fixed design has now increased to a maximum sample size of 1786 which is an inflation of 30%. On the other hand, the expected number of patients when a boundary is crossed is 770 under the assumption of no treatment difference and 1054 under the alternative hypothesis of a 15% event rate in the control group and 10% in the experimental group. Thus, this redesign seems reasonably effective at controlling the sample size when the experimental regimen has no underlying benefit. The nominal $\alpha-$level of .0124 required for a positive result at the end of the study is almost exactly half that of the overall .025 for the study. We will propose other designs that will not require such a small final nominal α by setting higher early efficacy bounds.

```
Asymmetric two-sided group sequential design with 80 % power and 2.5 % Type I Error.
Upper bound spending computations assume trial continues if lower bound is crossed.

                 ----Lower bounds----  ----Upper bounds-----
  Analysis   N    Z   Nominal p Spend+  Z   Nominal p Spend++
         1  447 -0.05    0.4816   0.05 2.50    0.0063  0.0063
         2  893  0.82    0.7953   0.05 2.41    0.0080  0.0063
         3 1340  1.53    0.9370   0.05 2.32    0.0101  0.0063
         4 1786  2.24    0.9876   0.05 2.24    0.0124  0.0062
     Total                      0.2000                0.0250
+ lower bound beta spending (under H1): Piecewise linear spending function with
line points = 0.25 0.5 0.75 0.25 0.5 0.75
++ alpha spending: Piecewise linear spending function with
line points = 0.25 0.5 0.75 0.25 0.5 0.75

Boundary crossing probabilities and expected sample size assuming any cross stops the trial

Upper boundary (power or Type I Error)
          Analysis
   Theta      1      2      3      4  Total   E{N}
  0.0000 0.0063 0.0062 0.0059 0.0042 0.0225  769.7
  0.0757 0.1843 0.2805 0.2253 0.1100 0.8000 1054.1

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2      3      4  Total
  0.0000 0.4816 0.3321 0.1299 0.0339 0.9775
  0.0757 0.0500 0.0500 0.0500 0.0500 0.2000
```

Now we display piecewise linear spending functions. The plot resulting from the code below is in 4.

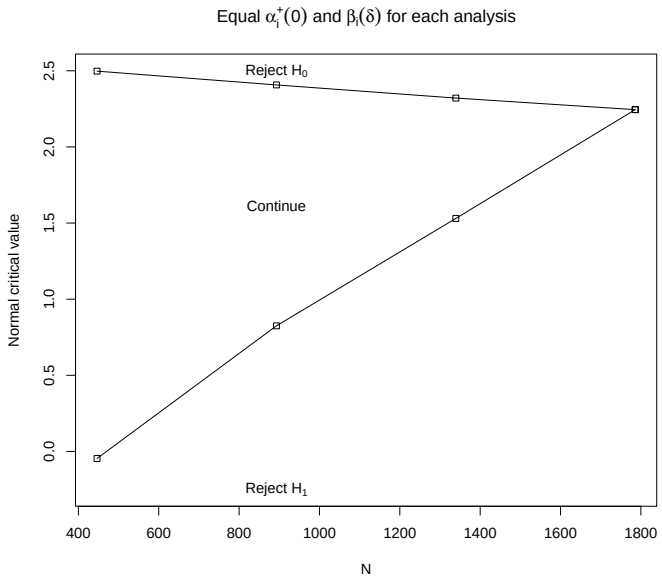

Figure 3: Boundary plot with equal $\alpha_i^+(0)$ and $\beta_i(\delta)$, $i = 1, 2, 3, 4$.

```
# Cumulative proportion of spending planned at each analysis
# Now use a piecewise linear spending
p <- c(.05, .2, .5)
p2 <- c(.6,.8,.85)
x <- gsDesign(k=4, n.fix=n.fix, beta=.2, sfu=sfLinear, sfupar=c(t,p),
              sfl=sfLinear, sflpar=c(t,p2))
plot(x, plottype="sf", main="Piecewise linear spending")
```

4.2 Deriving group sequential designs using boundary families

The second method of setting boundaries uses the relative z-value for a design cutoff at each interim analysis, $c_i > 0$, $i = 1, 2, \ldots, k$. We define vectors $\mathbf{t} \equiv (t_1, t_2, \ldots, t_k)$ and $\mathbf{c} \equiv (c_1, c_2, \ldots, c_k)$. For 2-sided testing, Wang and Tsiatis [23] defined the boundary function

$$-a_i = b_i = C(\mathbf{t}, \mathbf{c})c_i \tag{13}$$

where the constant $C(\mathbf{t}, \mathbf{c}) > 0$ is chosen to appropriately control Type I error.

Wang and Tsiatis [23] specifically defined the boundary function family

$$g(t; \Delta) = C(\mathbf{t}; \Delta)t^{\Delta - .5}. \tag{14}$$

For $i = 1, 2, \ldots, k$ the boundary at analysis i are given by

$$-a_i = b_i = C(\mathbf{t}; \Delta)t_i^{\Delta - .5}.$$

For 2-sided testing, note that for $\Delta = .5$ the boundary values are all equal. Thus, this is equivalent to a Pocock [17] boundary when analyses are equally spaced. The value $\Delta = 0$ generates O'Brien-Fleming bounds [15].

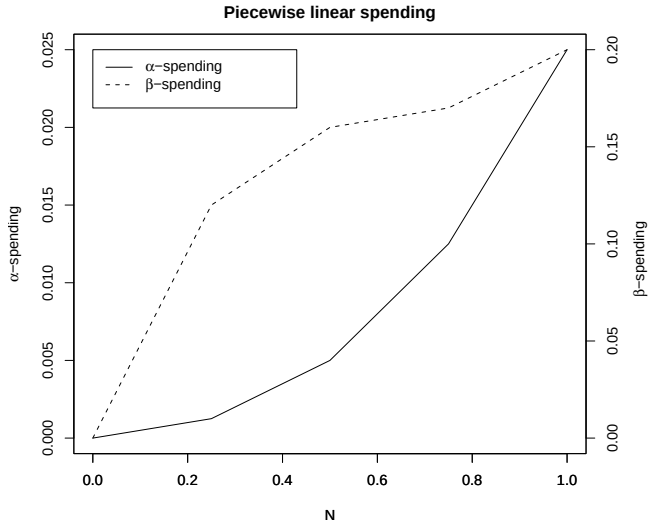

Figure 4: Piecewise linear spending functions.

Pampalona and Tsiastis [16] derived a related method of using boundary families to set asymmetric bounds; this is not currently implemented in `gsDesign()`. Using constants $c_i' > 0$, $i = 1, 2, \ldots, k$ and a constant $C'(\mathbf{t}; \mathbf{I_k})$ that along with I_k is used to appropriately control Type II error, they set

$$a_i = \delta\sqrt{t_i} - C'(\mathbf{t})c_i'. \tag{15}$$

O'Brien-Fleming, Pocock, or Wang-Tsiatis are normally used with equally-spaced analyses. They are used only with one-sided (`test.type=1`) and symmetric two-sided (`test.type=2`) designs. We will use the CAPTURE example, again with 80% power rather than the default of 90%. Notice that this requires specifying `beta=.2` in both `nBinomial()` and `gsDesign()`. O'Brien-Fleming, Pocock, or Wang-Tsiatis (parameter of 0.15) bounds for equally space analyses are generated using the parameters `sfu` and `sfupar` below. If you print the Pocock design (`xPocock`), you will see that the upper bounds are all equal and that the upper boundary crossing values $\alpha_i(0)$ printed in the `Spend` column decrease from .0091 for the first analysis to .0041 for the final analysis.

```
n.fix <- nBinomial(p1=.15, p2=.1, beta=.2)
xOF <- gsDesign(k=4, test.type=2, n.fix=n.fix, sfu="OF", beta=.2)
xPocock <- gsDesign(k=4, test.type=2, n.fix=n.fix, sfu="Pocock", beta=.2)
xWT <- gsDesign(k=4, test.type=2, n.fix=n.fix, sfu="WT", sfupar=.15, beta=.2)
```

The resulting sample sizes for these designs can be computed using

```
nOF <- 2 * ceiling(xOF$n.I[4] / 2)
nPocock <- 2 * ceiling(xPocock$n.I[4] / 2)
nWT <- 2 * ceiling(xWT$n.I[4] / 2)
```

We now present an example of how is fairly simple to produce custom plots using `gsDesign()` output and standard R plotting functions. The resulting output is in Figure 5. If you are not familiar with R

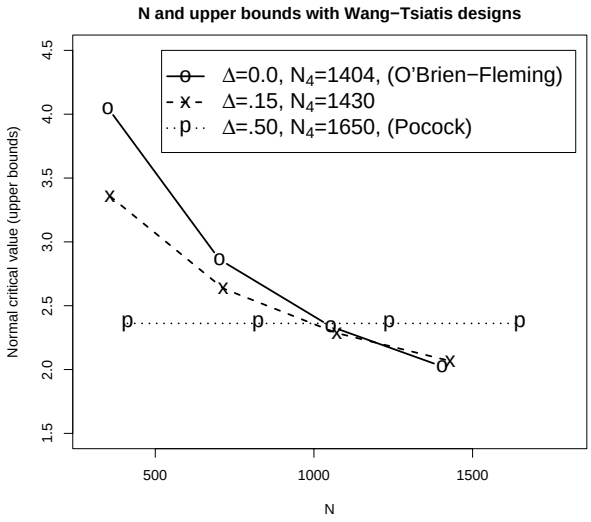

Figure 5: Wang-Tsiatis bounds for the CAPTURE example

plotting, executing the following statements one at a time may be instructive. The call `help(plot)` and its "See also" links (especially `par`) can be used to find explanations of parameters below. The `legend` call below particularly demonstrates a nice strength of R for printing greek characters and subscripts in plots.

```
plot(xOF$n.I,xOF$upper$bound,xlim=c(300,1800),ylim=c(1.5,4.5),cex=1.5,lwd=2,
     type="b",xlab="N",ylab="Normal critical value (upper bounds)",pch="o",
     main="N and upper bounds with Wang-Tsiatis designs")
lines(xPocock$n.I,xPocock$upper$bound,lty=3,lwd=2)
points(xPocock$n.I,xPocock$upper$bound,pch="p",cex=1.5)
lines(xWT$n.I,xWT$upper$bound,lty=2,lwd=2)
points(xWT$n.I,xWT$upper$bound,pch="x",cex=1.5)
legend(x=c(750,1825), y = c(3.7,4.5), lty=c(1,2,3), lwd=2, pch=c("o","x","p"),
       cex=1.5,
       legend=c(expression(paste(Delta,"=0.0, ",N[4],
                "=1404, (O'Brien-Fleming)")),
       c(expression(paste(Delta,"=.15, ",N[4],"=1430")),
       c(expression(paste(Delta,"=.50, ",N[4],"=1650, (Pocock)"))))))
```

Figure 5 shows how the upper bounds and sample size change as Δ changes for Wang-Tsiatis bounds. For the O'Brien-Fleming design, the final sample size is only inflated to 1402 from the 1372 required for a fixed design. The relatively aggressive early bounds for the the Pocock design result in sample size inflation to 1650. This design is not frequently used because of the relatively low bounds at early analyses and the substantial sample size inflation required to maintain the desired power. Since the nominal p-value required for stopping at the initial analysis for the O'Brien-Fleming design is .00005 (2-sided), an intermediate design with $\Delta = .15$ might be of some interest. This has a relatively small sample size inflation to 1430 in order to maintain power and the nominal p-value

required to stop the trial at the first interim analysis is .0008 (2-sided). Examine the boundary crossing probabilities by reviewing, for example, `xOF$upper$spend`. Also consider reviewing `plot(xWT, plottype=3)` to see the observed treatment effect required at each analysis to cross a boundary.

5 Other `gsDesign()` parameters

5.1 Setting Type I error and power

Type I error as input to `gsDesign()` is always one-sided and is set through the parameter `alpha`. Type II error (1-power) is set in the parameter `beta`. A standard design modified to have Type I error of .05 and Type II error of .2 (80% power) rather than the default of .025 Type I and .1 Type II error is produced with the command

```
x <- gsDesign(alpha=.05, beta=.2)
```

5.2 Number and timing of analyses

The number of analyses is set in `gsDesign()` through the parameter `k>1`, which has a default of 3. The default for timing of analyses is to have them equally-spaced, which is indicated by the default value of `timing=1`. This will often not be feasible or desired due to logistical or other reasons. The parameter `timing` can be input as a vector of length `k` or `k-1` where $0 <$ `timing[1]` $<$ `timing[2]` $< \ldots <$ `timing[k]` $= 1$. It is optional to specify `timing[k]` since it is always 1. The values in `timing` set the proportion of statistical information available for the data analyzed at each interim analysis. The statistical information is generally proportional to the number of observations analyzed or, for survival analysis, the number of time-to-event endpoints that have been observed. The following compares upper bounds, number of observations at each analysis, and average number of observations at the analysis where a boundary is crossed for the default design (stored in `x`) versus an alternative analyzing after 25%and 50% of observations (stored in `xt`) for the CAPTURE example. You can see that the upper bounds are more stringent when analyses are done earlier in the trial.

```
> n.fix <- nBinomial(p1=.15, p2=.1)
> x <- gsDesign(n.fix=n.fix)
> xt <- gsDesign(n.fix=n.fix, timing=c(.25, .5))
> 2*ceiling(x$n.I/2)
[1]  656 1310 1964
> x$upper$bound
[1] 3.010739 2.546531 1.999226
> x$en
[1] 1146.391 1451.709
> 2*ceiling(xt$n.I/2)
[1]  482  964 1926
> xt$upper$bound
[1] 3.155373 2.818347 1.983563
> xt$en
[1] 1185.173 1547.649
```

Comparing the designs, we see that the average sample number is lower for the default design with evenly spaced analyses compared to the design analyzing after 25% and 50% of observations. This is true both under the null hypothesis (1146 versus 1185) and the alternate hypothesis (1452 versus 1548) in spite of a lower maximum sample size (1926 versus 1964) for the latter design. To understand this further we look first at the probability of crossing the lower bound at each analysis

for each design below. The columns of the matrices printed correspond to the `theta` values under the null and alternate hypotheses, respectively, while rows correspond to the analyses. Thus, the default design has probability of 41% of crossing the lower bound at the first interim analysis compared to 25% for the design with first analysis at 25% of observations. By examining these probabilities as well as corresponding upper boundary crossing probabilities (*e.g.*, `x$upper$prob`) we see that by moving analyses earlier without changing spending functions we have decreased the probability of crossing an interim boundary, which explains the smaller expected sample size for the default design which uses later interim analyses.

```
> x$lower$prob
          [,1]       [,2]
[1,] 0.4056598 0.01483371
[2,] 0.4290045 0.02889212
[3,] 0.1420312 0.05627417
> xt$lower$prob
          [,1]       [,2]
[1,] 0.2546094 0.01015363
[2,] 0.3839157 0.01674051
[3,] 0.3375615 0.07310586
```

5.3 Standardized treatment effect: `delta`

5.3.1 Normally distributed data

The "usual" formula for sample size for a fixed design is

$$n = \left(\frac{Z_{1-\alpha} + Z_{1-\beta}}{\delta} \right)^2. \tag{16}$$

This formula originates from testing the mean of a sample of normal random variables with variance 1. The null hypothesis is that the true mean θ equals 0, while the alternate hypothesis is that $\theta = \delta$. The distribution of the mean of n observations $\bar{X}_n$ follows a normal distribution with mean δ and standard deviation $1/n$ (*i.e.*, N(δ,1/n)). Assuming $\delta > 0$, the standard statistic for testing this is $Z_n = \sqrt{n}\bar{X}_n \sim \mathrm{N}(\sqrt{n}\delta, 1)$ which rejects the hypothesis that the true mean is 0 if $Z_n > Z_{1-\alpha}$. The null hypothesis is rejected with probability α when the null hypothesis is true (Type I error), while the probability of rejecting under the alternate hypothesis (power or one minus Type II error) is

$$\Phi(Z_{1-\alpha} - \sqrt{n}\delta). \tag{17}$$

By fixing this probability as $1 - \beta$ and solving for n, equation (16) is derived.

Assume a set of patients is evaluated at baseline for a measure of interest, then treated with a therapy and subsequently measured for a change from baseline. Assume the within subject variance for the change from baseline is 1. Suppose $\delta = .1$. The default group sequential design can be obtained for such a study using the call `gsDesign(delta = .1)`, yielding a planned maximum sample size of 1125.

5.3.2 Time to event data

Equations (16) and (17) are used as approximations for many situations where test statistics are approximated well by the normal distribution as n gets large. A useful example of this approximation is comparing survival distributions for two groups under the assumption that the hazard rate ("instantaneous failure rate") for the control group ($\lambda_1(t)$) and experimental group ($\lambda_2(t)$) for any time $t > 0$ are proportional as expressed by

$$\lambda_2(t) = e^{-\gamma}\lambda_1(t). \tag{18}$$

We have used $-\gamma$ in the exponent so that a positive value of γ indicates lower risk in the experimental treatment group. The value e^{-gamma} will be referred to as the hazard ratio and γ as minus the log hazard ratio.

Note that when $\gamma = 0$ there is no difference in the hazard rates between treatment groups. A common test statistic for the null hypothesis that $\gamma = 0$ is the logrank test. We will denote this by $T(d)$ where d indicates the number of events observed in the combined treatment groups. A reasonably good approximation for its distribution is

$$T(d) \sim \mathrm{N}(\gamma \times V(d), V(d)). \tag{19}$$

For equally sized treatment groups, $V(d)$ is approximately $d/4$. Thus,

$$Z = T(d)2/\sqrt{d} \sim \mathrm{N}(\sqrt{d}\gamma/2, 1). \tag{20}$$

For the formulation from Section 2.1 we have $\theta = \gamma/2$. If $\gamma = \mu$ is the alternative hypothesis to the null hypothesis $\gamma = 0$, then we have $\delta = \mu/2$. In fact, Tsiatis [22], Sellke and Siegmund [19] and Slud and Wei [20] have all shown that group sequential theory may be applied to censored survival data using this distributional assumption; this is also discussed by Jennison and Turnbull [9]. If we assume there are k analyses after $d_1 < d_2 < \ldots < d_k$ events and let $I_i = d_i/4$, $i = 1, 2, \ldots, k$ then we may apply the canonical distribution assumptions from (3) and (4).

For the cancer trial example in Section 1.6 we assumed $e^{-\mu} = .7$ which yields $\delta = -\ln(.7)/2 = .178$. Applying (16) with $\alpha = .025$ and $\beta = .1$ and this value of δ, the number of events required is calculated as 331 compared to 330 calculated previously using the Lachin and Foulkes [11] method. We may obtain the default group sequential design by specifying δ rather than the fixed design number of events as follows: `gsDesign(delta = -log(.7)/2)`.

We also apply this distribution theory to the non-inferiority trial for a new treatment for diabetes. We wish to rule out a hazard ratio of 1.3 for the experimental group compared to the control group under the assumption that the risk of cardiovascular events is equal in the two treatment groups. This implies that our null hypothesis is that $\gamma = \ln(1.3) = .262$ and the alternate hypothesis is that $\gamma = 0$. Letting $\theta = (\gamma - \ln(1.3))/2$ the null hypothesis is re-framed as $\theta = 0$ and the alternative as $\theta = \ln(1.3)/2$. The test statistic $Z = (T(d) - \ln(1.3) \times d/4) \times 2/\sqrt{d}$ is then approximately distributed $\mathrm{N}(\sqrt{d}\theta, 1)$. Substituting $\delta = \ln(1.3)/2$, $\alpha = .025$ and $\beta = .1$ in (16) we come up with $d = 611$. This is within 1% of the 617 events suggested in Section 1.8.

6 Spending Functions

6.1 Spending function definitions.

For any given $0 < \alpha < 1$ we define a non-decreasing function $f(t; \alpha)$ for $t \geq 0$ with $\alpha\,(0) = 0$ and for $t \geq 1$, $f(t; \alpha) = \alpha$. For $i = 1, 2, \ldots, K$ we define $t_i = I_i/I_K$ and then set $\alpha_i(0)$ through the equation

$$f(t_i; \alpha) = \sum_{j=1}^{i} \alpha_j(0). \tag{21}$$

We consider a spending function proposed by Lan and DeMets [12] to approximate a Pocock bound.

$$f(t; \alpha) = \alpha \log(1 + (e - 1)t) \tag{22}$$

This spending function is implemented in the function `sfLDPocock`. We again consider a 2-sided design with equally spaced analyses, $t_i = i/4$, $i = 1, 2, 3, 4$. The values for $\alpha_i(0)$ are obtained as follows:

```
> sfLDPocock(alpha=.025, t=1:4/4)$spend
[1] 0.00893435 0.01550286 0.02069972 0.02500000
```

We will discuss the exact nature of this call to `sfLDPocock` in Section 7 below. We now derive a design for the CAPTURE study using this spending function

```
> n.fix <- nBinomial(p1=.15, p2=.1, beta=.2)
> x <- gsDesign(k=4, test.type=2, n.fix=n.fix, sfu=sfLDPocock, beta=.2)
> cumsum(x$upper$prob[,1])
[1] 0.00893435 0.01550287 0.02069973 0.02500001
```

The boundary crossing probabilities under the assumption of no treatment difference are in `x$upper$prob[,1]` and there cumulative totals are produced by the above call to `cumsum()`. Note that these values match those produced by the call to `sfLDPocock` above. Next we compare the bounds produced by this design with the actual Pocock bounds to see they are nearly identical:

```
> xPocock <- gsDesign(k=4, test.type=2, n.fix=n.fix, sfu="Pocock", beta=.2)
> x$upper$bound
[1] 2.368328 2.367524 2.358168 2.350030
> xPocock$upper$bound
[1] 2.361298 2.361298 2.361298 2.361298
>
```

The reader may wish to compare the O'Brien-Fleming design presented in 4.2 using the spending function `sfLDOF`, which implements a spending function proposed by Lan and DeMets [12] to approximate this design:

$$\alpha_i(t) = 2\left(1 - \Phi\left(\frac{\Phi^{-1}(\alpha/2)}{\sqrt{t}}\right)\right) \tag{23}$$

You will see that this approximation is not as good as the Pocock bound approximation.

6.2 Spending function families

The function $f(t;\alpha)$ may be generalized to a family $f(t;\alpha,\gamma)$ of spending functions using one or more parameters. For instance, the default Hwang-Shih-DeCani spending function family is defined for $0 \le t \le 1$ and any real γ by

$$f(t;\alpha,\gamma) = \begin{array}{ll} \alpha\frac{1-\exp(-\gamma t)}{1-\exp(-\gamma)}, & \gamma \neq 0 \\ \alpha t, & \gamma = 0 \end{array}$$

The boundary crossing probabilities $\alpha_i^+(\theta)$ and $\beta_i(\theta)$ may be defined in a similar fashion, $i = 1, 2, \ldots, K$ with the same or different spending functions f where:

$$f(t_i;\alpha) = \sum_{j=1}^{i} \alpha_j^+(0) \tag{24}$$

$$f(t_i;\beta(\theta)) = \sum_{j=1}^{i} \beta_j(\theta) \tag{25}$$

The argument `test.type` in `gsDesign()` provides two options for how to use (25) to set lower bounds. For `test.type`=2, 5 and 6, lower boundary values are set under the null hypothesis by specifying $\beta(t;0)$, $0 \le t$. For `test.type`=3 and 4, we compute lower boundary values under the

alternative hypothesis by specifying $\beta(t;\delta)$, $0 \le t$. $\beta(t;\delta)$ is referred to as the β-spending function and the value $\beta_i(\delta)$ is referred to as the amount of β (Type II error rate) spent at analysis i, $1 \le i \le K$.

Standard published spending functions commonly used for group sequential design are included as part of the gsDesign package. Several 'new' spending functions are included that are of potential interest. Users can also write their own spending functions to pass directly to `gsDesign()`. Available spending functions and the syntax for writing a new spending function are documented here. We begin here with simple examples of how to apply standard spending functions in calls to `gsDesign()`. This may be as far as many readers may want to read. However, for those interested in more esoteric spending functions, full documentation of the extensive spending function capabilities available is included. Examples for each type of spending function in the package are included in the online help documentation.

6.3 Spending Function Basics

The parameters `sfu` and `sfl` are used to set spending functions for the upper and lower bounds, respectively, each having a default value of `sfHSD`, the Hwang-Shih-DeCani spending function. The default parameter for the upper bound is $\gamma = -4$ to produce a conservative, O'Brien-Fleming-like bound. The default for the lower bound is $\gamma = -2$, a less conservative bound. This design was presented at some length in 3.

To change these to -3 (less conservative than an O'Brien-Fleming bound) and 1 (an aggressive Pocock-like bound), respectively, requires the parameter `sfupar` for the upper bound and `sflpar` for the lower bound: Next we consider some simple alternatives to the default spending function parameters. The Kim-DeMets function, `sfPower()`, with upper bound parameter $\rho = 3$ (a conservative, O'Brien-Fleming-like bound) and lower bound parameter $\rho = 0.75$ (an aggressive, Pocock-like bound) requires resetting the upper bound spending function `sfu` and the lower bound spending function `sfl`. In the first code line following, we replace lower and upper spending function parameters with 1 and -2, respectively; the default Hwang-Shih-DeCani spending function family is still used. In the second line, we specify a Kim-DeMets (power) spending function for both the lower bound (with the parameters `sfl=sfPower` and `sflpar=2`) and the upper bounds (with the parameters `sfu=sfPower` and `sfupar=3`). Then we compare bounds from the three designs. Bounds for the power spending function design are quite comparable to the default design. Generally, choosing between these two spending function families is somewhat arbitrary. The alternate Hwang-Shih-DeCani design uses more aggressive stopping boundaries. The last lines below show that sample size inflation from a fixed design is about 25% for the the design with more aggressive stopping boundaries compared to about 7% for each of the other designs.

```
> x <- gsDesign()
> xHSDalt <- gsDesign(sflpar=1, sfupar=-2)
> xKD <- gsDesign(sfl=sfPower, sflpar=2, sfu=sfPower, sfupar=3)
> x$upper$bound
[1] 3.010739 2.546531 1.999226
> xHSDalt$upper$bound
[1] 2.677524 2.385418 2.063740
> xKD$upper$bound
[1] 3.113017 2.461933 2.008705
> x$lower$bound
[1] -0.2387240  0.9410673  1.9992264
> xHSDalt$lower$bound
[1] 0.3989132 1.3302944 2.0637399
> xKD$lower$bound
```

```
[1] -0.3497491  0.9822541  2.0087052
> x$n.I[3]
[1] 1.069883
> xHSDalt$n.I[3]
[1] 1.254268
> xKD$n.I[3]
[1] 1.071011
>
```

Following is example code to plot Hwang-Shih-DeCani spending functions for three values of the γ parameter. The first two γ values are the defaults for upper bound spending ($\gamma = -4$; a conservative bound somewhat similar to an O'Brien-Fleming bound) and lower bound spending ($\gamma = -2$; a less conservative bound). The third ($\gamma = 1$) is included as it approximates a Pocock stopping rule; see Hwang, Shih and DeCani [8]. The Hwang-Shih-DeCani spending function class implemented in the function `sfHSD()` may be sufficient for designing many clinical trials without considering the other spending function forms available in this package. The three parameters in the calls to `sfHSD()` below are the total Type I error, values for which the spending function is evaluated (and later plotted), and the γ parameter for the Hwang-Shih-DeCani design. The code below yields the plot in Figure 6 below (note the typesetting of Greek characters!):

```
> plot(0:100/100, sfHSD(.025, 0:100/100, -4)$spend, type="l", lwd=2,
+      xlab="Proportion of information",
+      ylab=expression(paste("Cumulative \ ",alpha,"-spending")),
+      main="Hwang-Shih-DeCani Spending Functions")
> lines(0:100/100, sfHSD(.025, 0:100/100, -2)$spend, lty=2, lwd=2)
> lines(0:100/100, sfHSD(.025, 0:100/100, 1)$spend, lty=3, lwd=2)
> legend(x=c(.0, .27), y=.025 * c(.8, 1), lty=1:3, lwd=2,
+        legend=c(expression(paste(gamma," = -4")),
+                 expression(paste(gamma," = -2")),
+                 expression(paste(gamma," = 1"))))
```

Similarly, Jennison and Turnbull [9], suggest that the Kim-DeMets spending function is flexible enough to suit most purposes. To compare the Kim-DeMets family with the Hwang-Shih-DeCani family just demonstrated, substitute `sfPower()` instead of `sfHSD()`; use parameter values 3, 2 and 0.75 to replace the values -4,-2, and 1 in the code shown above:

```
> plot(0:100/100,sfPower(.025, 0:100/100, 3)$spend, type="l", lwd=2,
+      xlab="Proportion of information",
+      ylab=expression(paste("Cumulative \ ",alpha,"-spending")),
+      main="Kim-DeMets Spending Functions")
> lines(0:100/100, sfPower(.025, 0:100/100, 2)$spend, lty=2, lwd=2)
> lines(0:100/100, sfPower(.025, 0:100/100, 0.75)$spend, lty=3, lwd=2)
> legend(x=c(.0, .27), y=.025 * c(.8, 1), lty=1:3, lwd=2,
+        legend=c(expression(paste(gamma," = 3")),
+                 expression(paste(gamma," = 2")),
+                 expression(paste(gamma," = 0.75"))))
```

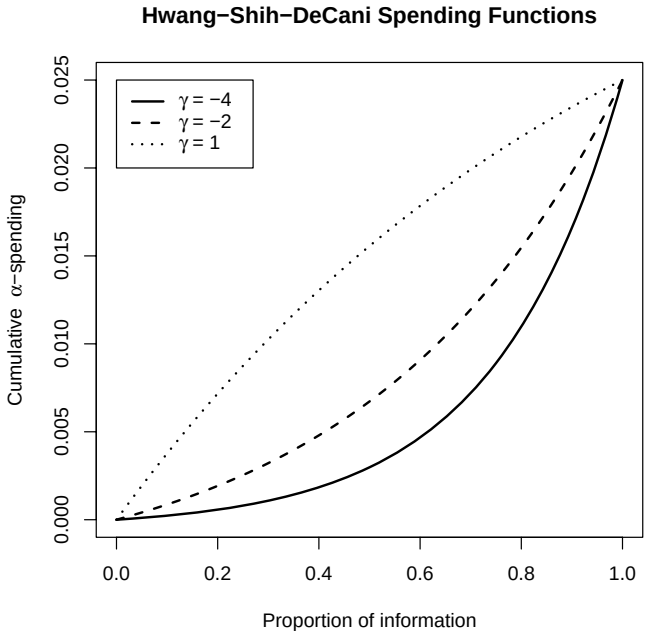

Figure 6: Hwang-Shih-DeCani spending function example

6.4 Resetting timing of analyses

When designed with a spending function, the timing and number of analyses may be altered during the course of the trial. This is very easily handled in the `gsDesign()` routine using the input arguments `n.I` and `maxn.IPlan`. We demonstrate this by example. Suppose a trial was originally designed with the call:

```
> x <- gsDesign(k=5, n.fix=800)
> x$upper$bound
> x$n.I
```

The second and third lines above show the planned upper bounds and sample sizes at analyses. Suppose that when executed the final interim was skipped, the first 2 interims were completed on time, the third interim was completed at 575 patients (instead of 529 as originally planned), the fourth interim was skipped, and the final analysis was performed after 875 patients (instead of after 882 as originally planned). The boundaries for the analyses can be obtained as follows:

```
> gsDesign(k=4, n.fix=800, n.I=c(177,353,575,875), maxn.IPlan=x$n.I[x$k])
```

This design now has slightly higher power (90.4%) than the originally planned 90%. This is because the final boundary was lowered relative to the original plan when the α-spending planned for the fourth interim was saved for the final analysis by skipping the final interim. Note that all of the arguments for the original design must be supplied when the study is re-designed—in addition

to adding `n.I`, which may have the same number, fewer, or more interim analyses compared to the original plan. If the sample size for the final analysis is changed, `maxn.IPlan` should be passed in as the original final sample size in order to appropriately assign α- and β-spending for the interim analyses.

7 Advanced spending function details

7.1 Spending functions as arguments

Except for the "OF", "Pocock" and "WT" examples given in Section 4.2, a spending function passed to `gsDesign()` through the arguments `sfu` (upper bound) and `sfl` (lower bound) must have the following calling sequence:

```
sfname(alpha, t, param)
```

where `sfname` is an arbitrary name for a spending function available from the package or written by the user. The arguments are as follows:

- `alpha`: a real value ($0 <$ `alpha` < 1). The total error spending for the boundary to be determined. This would be replaced with the following values from a call to `gsDesign()`: `alpha` for the upper bound, and either `beta` (for `test.type = 3` or `4`) or `astar` (for `test.type = 5` or `6`) for the lower bound.
- `t`: a vector of arbitrary length m of real values, $0 \leq t_1 < t_2 < \ldots t_m \leq 1$. Specifies the proportion of spending for which values of the spending function are to be computed.
- `param`: for all cases here, this is either a real value or a vector of real values. One or more parameters that (with the parameter `alpha`) fully specify the spending function. This is specified in a call to `gsDesign()` with `sfupar` for the upper bound and `sflpar` for the lower bound.

The value returned is of the class `spendfn` which was described in Section 6, The `spendfn` Class.

The following code and output demomstrate that the default value `sfHSD` for the upper and lower spending functions `sfu` and `sfl` is a function:

```
> sfHSD
function (alpha, t, param)
{
    checkScalar(param, "numeric", c(-40, 40))
    t[t > 1] <- 1
    x <- list(name = "Hwang-Shih-DeCani", param = param, parname = "gamma",
        sf = sfHSD, spend = if (param == 0) t * alpha else alpha *
            (1 - exp(-t * param))/(1 - exp(-param)), bound = NULL,
        prob = NULL)
    class(x) <- "spendfn"
    x
}
<environment: namespace:gsDesign>
```

Table 1 summarizes many spending functions available in the package. A basic description of each type of spending function is given. The table begins with standard spending functions followed

Table 3: Spending function definitions and parameterizations.

Function (parameter)	Spending Function Family	Functional Form	Parameter (`sfupar` or `sflpar`)
`sfHSD` (`gamma`)	Hwang-Shih-DeCani	$\alpha\frac{1-\exp(-\gamma t)}{1-\exp(-\gamma)}$	Value in [-40,40). -4=O'Brien-Fleming like; 1=Pocock-like
`sfPower` (`rho`)	Kim-DeMets	αt^ρ	Value in $(0,+\infty)$ 3=O'Brien-Fleming like 1=Pocock-like
`sfLDPocock` (`none`)	Pocock approximation	$\alpha\log(1+(e-1)t)$	None
`sfLDOF` (`none`)	O'Brien-Fleming approximation	$2\left(1-\Phi\left(\frac{\Phi^{-1}(\alpha/2)}{\sqrt{t}}\right)\right)$	None
`sfLinear` ($\mathtt{t}_1$,$\mathtt{t}_2$,...,$\mathtt{t}_m$) ($\mathtt{p}_1$,$\mathtt{p}_2$,...,$\mathtt{p}_m$)	Piecewise linear specification	Specified points $0<t_1\ldots<t_m<1$ $0<p_1\ldots<p_m<1$	Cumulative proportion of information and error spending for given timepoints
`sfExponential` (`nu`)	Exponential	$\alpha^{t^{-\nu}}$	$(0,10]$ Recommend $\nu<1$ 0.75 =O'Brien-Fleming-like
`sfLogistic` (`a,b`)	Logistic	$\alpha\frac{e^a\left(\frac{t}{1-t}\right)^b}{1+e^a\left(\frac{t}{1-t}\right)^b}$	$b>0$
`"WT"` (`Delta`)	Wang-Tsiatis bounds	$C(k,\alpha,\Delta)(i/K)^{\Delta-1/2}$	0=O'Brien-Fleming bound 0.5=Pocock bound
`"Pocock"` (`none`)	Pocock bounds		This is a special case of WT with $\Delta=1/2$.
`"OF"` (`none`)	O'Brien-Fleming bounds		This is a special case of WT with $\Delta=0$.

by two investigational spending functions: `sfExponential()` and `sfLogistic()`. These spending functions are discussed further in Section 7.2, Investigational Spending Functions, but are included here due to their simple forms. The logistic spending function represented by `sfLogistic()` has been used in several trials. It represents a class of two-parameter spending functions that can provide flexibility not available from one-parameter families. The `sfExponential()` spending function is included here as it provides an excellent approximation of an O'Brien-Fleming design as follows:

```
gsDesign(test.type=2, sfu=sfExponential, sfupar=0.75)
```

See also subsections below and online documentation of spending functions.

7.2 Investigational spending functions

When designing a clinical trial with interim analyses, the rules for stopping the trial at an interim analysis for a positive or a negative efficacy result must fit the medical, ethical, regulatory and statistical situation that is presented. Once a general strategy has been chosen, it is not unreasonable to discuss precise boundaries at each interim analysis that would be considered ethical for the purpose

of continuing or stopping a trial. Given such specified boundaries, we discuss here the possibility of numerically fitting α- and β-spending functions that produce these boundaries. Commonly-used one-parameter families may not provide an adequate fit to multiple desired critical values. We present a method of deriving two-parameter families to provide some additional flexibility along with examples to demonstrate their usefulness. This method has been found to be useful in designing multiple trials, including the CAPTURE trial [3], the GUSTO V trial [21] and three ongoing trials at Merck.

One method of deriving a two-parameter spending function is to use the incomplete beta distribution which is commonly denoted by $I_x(a, b)$ where $a > 0$, $b > 0$. We let

$$\alpha(t; a, b) = \alpha I_t(a, b).$$

This spending function is implemented in `sfBetaDist()`; developing code for this is also demonstrated below in Section 7.4, Writing Code for a New Spending Function.

Another approach allows fitting spending functions by solving a linear system of 2 equations in 2 unknowns. A two-parameter family of spending function is defined using an arbitrary, continuously increasing cumulative distribution function $F()$ defined on $(-\infty, \infty)$, a real-valued parameter a and a positive parameter b:

$$\alpha(t; a, b) = \alpha F(a + bF^{-1}(t)). \tag{26}$$

Fix two points of the spending function $0 <$ `t0` $<$ `t1` < 1 to have spending function values specified by `u0` $\times$ `alpha` and `u1` $\times$ `alpha`, respectively, where $0 <$ `u0` $<$ `u1` < 1. Equation (26) now yields two linear equations with two unknowns, namely for $i = 1, 2$

$$F^{-1}(u_i) = a + bF^{-1}(t_i).$$

The four value specification of `param` for this family of spending functions is `param=c(t0, t1, u0, u1)` where the objective is that `sf(t0) = alpha*u0` and `sf(t1) = alpha*u1`. In this parameterization, all four values must be between 0 and 1 and `t0` $<$ `t1`, `u0` $<$ `u1`.

The logistic distribution has been used with this strategy to produce spending functions for ongoing trials at Merck Research Laboratories in addition to the GUSTO V trial [21]. The logit function is defined for $0 < u < 1$ as $\text{logit}(u) = \log(u/(1-u))$. Its inverse is defined for $x \in (-\infty, \infty)$ as $\text{logit}^{-1}(x) = e^x/(1 + e^x)$. Letting $b > 0$, $c = e^a > 0$, $F(x) = \text{logit}^{-1}(x)$ and applying (26) we obtain the logistic spending function family:

$$\alpha(t; a, b) = \alpha \times \text{logit}^{-1}(\log(c) + b \times \text{logit}(u)) \tag{27}$$

$$= \alpha \frac{c\left(\frac{t}{1-t}\right)^b}{1 + c\left(\frac{t}{1-t}\right)^b} \tag{28}$$

The logistic spending function is implemented in `sfLogistic()`. Functions are also available replacing $F()$ with the cumulative distribution functions for the standard normal distribution (`sfNormal()`), two versions of the extreme value distribution (sfExtremeValue() with $F(x) = exp(-exp(-x))$ and sfExtremeValue2 with $F(x) = 1 - exp(-exp(x))$), the central t-distribution (`sfTDist()`), and the Cauchy distribution (`sfCauchy()`). Of these, `sfNormal()` is fairly similar to `sfLogistic()`. On the other hand, `sfCauchy()` can behave quite differently. The function `sfTDist()` provides intermediary spending functions bounded by `sfNormal()` and `sfCauchy()`; it requires an additional parameter, the degrees of freedom See online help for more complete documentation, particularly for `sfTDist()`. Following is an example that plots several of these spending functions that fit through the same two points (`t1`=0.25, `t2`=0.5, `u1`=0.1, `u2`=0.2) but behave differently for $t > 1/2$.

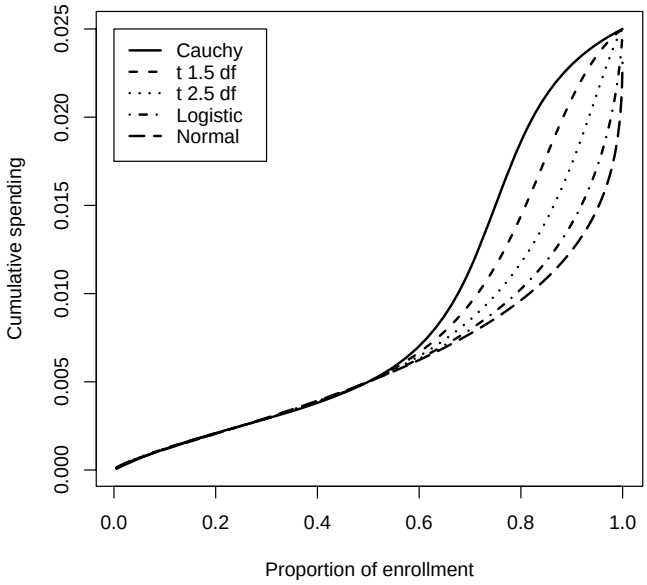

Figure 7: Example fitting two- and three-parameter spending functions

```
> plotsf <- function(alpha,t,param)
{
    plot(t, sfCauchy(alpha, t, param)$spend, lwd=2,
         xlab="Proportion of enrollment",
         ylab="Cumulative spending", type="l")
    lines(t, sfLogistic(alpha, t, param)$spend, lty=4, lwd=2)
    lines(t, sfNormal(alpha, t, param)$spend, lty=5, lwd=2)
    lines(t, sfTDist(alpha, t, c(param, 1.5))$spend, lty=2, lwd=2)
    lines(t, sfTDist(alpha, t, c(param,2.5))$spend, lty=3, lwd=2)
    legend(x=c(.0, .3), y=alpha * c(.7,1), lty=1:5, lwd=2,
           legend=c("Cauchy", "t 1.5 df", "t 2.5 df", "Logistic", "Normal"))
}
> t <- 1:199/200
> t <- c(t, .9975, .99875, .9995, .99975)
> param <- c(.25, .5, .1, .2)
> plotsf(.025,t,param)
```

7.3 Optimized spending functions

The following two examples demonstrate some of the flexibility and research possibilities for the `gsDesign` package. The particular examples may or may not be of interest, but the strategy may be applied using a variety of optimization criteria. First, we consider finding a spending function to match a Wang-Tsiatis design. This could be useful to adjust a Wang-Tsiatis design if the timing of interim analyses are not as originally planned. Second, we replicate a result from Anderson [1] which minimized expected value of the square of sample size over a family of spending functions and

a prior distribution.

Example 1 *Approximating a Wang-Tsiatis design*

We have noted several approximations of O'Brien-Fleming and Pocock spending rules using spending functions in the table above. Following is sample code to provide a good approximation of Wang-Tsiatis bounds with a given parameter Δ. This includes O'Brien-Fleming (Δ=0) and Pocock (Δ=0.5) designs. First, we define a function that computes the sum of squared deviations of the boundaries of a Wang-Tsiatis design compared to a one-parameter spending function family with a given parameter value of `x`. Note that `Delta` is the parameter for the Wang-Tsiatis design that we are trying to approximate. Other parameters are as before; recall that `test.type` should be limited to 1 or 2 for Wang-Tsiatis designs. Defaults are used for parameters for `gsDesign()` not included here.

```
WTapprox <- function(x, alpha=0.025, beta=.1, k=3, sf=sfHSD,
                     Delta=.25, test.type=2)
{
    # Wang-Tsiatis comparison with a one-parameter spending function
    y1 <- gsDesign(k=k, alpha=alpha, beta=beta, test.type=test.type, sfu="WT",
                   sfupar=Delta)$upper$bound
    y2 <- gsDesign(k=k, alpha=alpha, beta=beta, test.type=test.type, sfu=sf,
                   sfupar=x)$upper$bound
    z <- y1-y2
    return(sum(z*z))
}
```

We consider approximating a two-sided O'Brien-Fleming design with `alpha`=0.025 (one-sided) using the exponential spending function. The function `nlminb()` is a standard R function used for minimization. It minimizes a function passed to it as a function of that function's first argument, which may be a vector. The first parameter of `nlminb()` is a starting value for the minimization routine. The second is the function to be minimized. The parameter `lower` below provides a lower bound for first argument to the function being passed to `nlminb()`. Following parameters are fixed parameters for the function being passed to `nlminb()`. The result suggests that for $k = 4$, an exponential spending function with $\nu = 0.75$ approximates an O'Brien-Fleming design well. Examining this same optimization for $k = 2$ to 10 suggests that $\nu = 0.75$ provides a good approximation of an O'Brien-Fleming design across this range.

```
> nu <- nlminb(.67, WTapprox, lower=0, sf=sfExponential, k=4, Delta=0, test.type=2)$par
> nu
[1] 0.7562779
```

Running comparable code for `sfHSD()` and `sfPower()` illustrates that the exponential spending function can provide a better approximation of an O'Brien-Fleming design than either the Hwang-Shih-DeCani or Kim-DeMets spending functions. For Pocock designs, the Hwang-Shih-DeCani spending function family provides good approximations.

Example 2 *Minimizing the expected value of a power of sample size*

In this example, we first define a function that computes a weighted average across a set of `theta` values of the expected value of a given power of the sample size for a design. Note that `sfupar` and `sflpar` are specified in the first input argument so that they may later be optimized using the R routine `nlminb()`. The code is compact, which is very nice for writing, but it may be difficult to interpret. A good way to see how the code works is to define values for all of the parameters and run each line from the R command prompt, examining the result.

```
# Expected value of the power of sample size of a trial
# as a function of upper and lower spending parameter.
# Get sfupar from x[1] and sflpar from x[2].
# val is the power of the sample size for which expected
#    values are computed.
# theta is a vector for which expected values are to be computed.
# thetawgt is a vector of the same length used to compute a
#    weighted average of the expected values.
# Other parameters are as for gsDesign.

enasfpar <- function(x, timing=1, theta, thetawgt, k=4,
                     test.type=4, alpha=0.025, beta=0.1,
                     astar=0, delta=0, n.fix=1, sfu=sfHSD,
                     sfl=sfHSD, val=1, tol=0.000001, r=18)
{
    # derive design
    y <- gsDesign(k=k, test.type=test.type, alpha=alpha, beta=beta,
                  astar=astar, delta=delta, n.fix=n.fix, timing=timing,
                  sfu=sfu, sfupar=x[1], sfl=sfl, sflpar=x[2], tol=tol, r=r)
    # compute boundary crossing probabilities for input theta
    y <- gsProbability(theta=theta, d=y)
    # compute sample sizes to the val power
    n <- y$n.I^val
    # compute expected values
    en <- n %*% (y$upper$prob + y$lower$prob)
    # compute weighted average of expected values
    en <- sum(as.vector(en) * as.vector(thetawgt))
    return(en)
}
```

Now we use this function along with the R routine `nlminb()` which finds a minimum across possible values of `sfupar` and `sflpar`. The design derived using the code below and a more extensive discussion can be found in [1]. The code above is more general than in [1], however, as that paper was restricted to `test.type`=5 (the program provided there also worked for `test.type`=6).

```
# example from Anderson (2006)
delta <- abs(qnorm(.025) + qnorm(.1))
# use normal distribution to get weights
x <- normalGrid(mu=delta, sigma=delta/2)
x <- nlminb(start=c(.7, -.8), enasfpar, theta=x$z, timing=1,
       thetawgt=x$wgts, val=2, k=4, test.type=5, tol=0.0000001)
x$message
```

```
y <- gsDesign(k=4, test.type=5, sfupar=x$par[1], sflpar=x$par[2])
y
```

7.4 Writing code for a new spending function

Following is sample code using a cumulative distribution function for a beta-distribution as a spending function. Let B(a,b) denote the complete beta function. The beta distribution spending function is denoted for any fixed $a > 0$ and $b > 0$ by

$$\alpha(t) = \frac{\alpha}{B(a,b)} \int_0^t x^{a-1}(1-x)^{b-1} dx.$$

This spending function provides much of the flexibility of spending functions in the last subsection, but is not of the same general form. This sample code is intended to provide guidance in writing code for a new spending function, if needed.

```
# implementation of 2-parameter version of beta distribution spending function
# assumes t and alpha are appropriately specified (does not check!)
sfbdist <- function(alpha, t, param)
{
    # set up return list and its class
    x <- list(name="B-dist example", param=param, parname=c("a","b"),
         sf=sfbdist, spend=NULL, bound=NULL, prob=NULL, errcode=0,
         errmsg="No error")
    class(x) <- "spendfn"
    # check for errors in specification of a and b
    # gsReturnError is a simple function available from the
    # package for saving errors in the returned value}
    if (length(param) != 2)
    {
        return(gsReturnError(x,errcode=.3,
               errmsg="b-dist example spending function parameter must be of length 2"))
    }
    if (!is.numeric(param))
    {
        return(gsReturnError(x,errcode=.1,
               errmsg="Beta distribution spending function parameter must be numeric"))
    }
    if (param[1] <= 0)
    {
        return(gsReturnError(x,errcode=.12,
               errmsg="1st Beta distribution spending function parameter must be > 0."))
    }
    if (param[2] <= 0)
    {
        return(gsReturnError(x,errcode=.13,
               errmsg="2nd Beta distribution spending function parameter must be > 0."))
    }
    # set spending using cumulative beta distribution function and return
    x$spend <- alpha * pbeta(t, x$param[1], x$param[2])
    return(x)
}
```

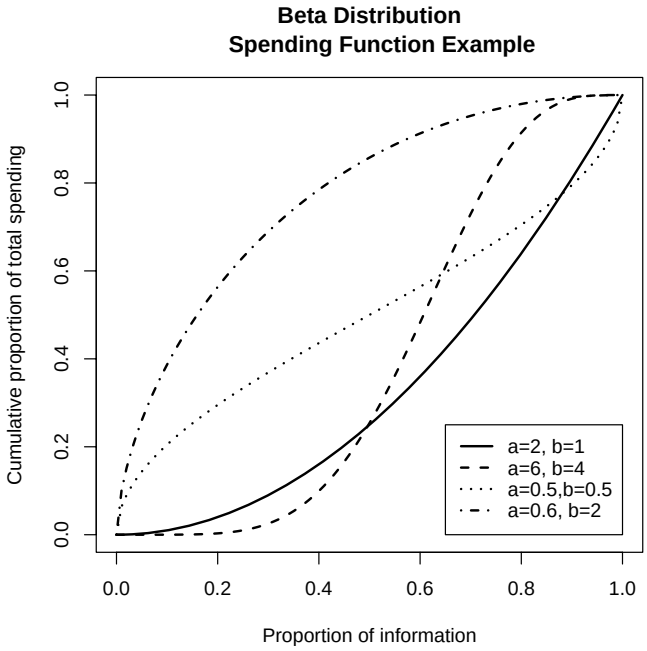

Figure 8: Example plotting user-written beta-distribution spending function

The flexibility of this spending function is demonstrated by the following code which produces the plot below. Using a=ρ, b=1 produces a Kim-DeMets spending function αt^{ρ} as shown by the solid line with ρ=2. The dashed line (a=6, b=4) shows a spending function that is conservative very early, but then aggressive in its spending pattern starting after about 40% of data are available. The dotted (a=0.5, b=0.5) and dot-dashed (a=0.6, b=2) show increasingly aggressive early spending. These may be useful in setting an initial high futility bound when the first part of a trial is used as a proof of concept for a clinical endpoint.

```
> # plot some beta distribution spending functions
> plot(0:100/100, sfbdist(1, 0:100/100, c(2,1))$spend, type="l", lwd=2,
+      xlab="Proportion of information",
+      ylab="Cumulative proportion of total spending",
+      main="Beta Distribution Spending Function Example")
> lines(0:100/100, sfbdist(1, 0:100/100, c(6,4))$spend, lty=2, lwd=2)
> lines(0:100/100, sfbdist(1, 0:100/100, c(.5,.5))$spend, lty=3, lwd=2)
> lines(0:100/100, sfbdist(1, 0:100/100, c(.6,2))$spend, lty=4, lwd=2)
> legend(x=c(.65, 1), y=1 * c(0, .25), lty=1:4, lwd=2,
+        legend=c("a=2, b=1", "a=6, b=4", "a=0.5,b=0.5", "a=0.6, b=2"))
```

8 Analyzing group sequential trials

We present several ways to review and interpret interim and final results from group sequential trials. Generally, regulatory agencies will have interest in well-controlled Type I error and unbiased treatment estimates.

Table 4: The CAPTURE data

	Placebo			Experimental			
Analysis	Events	n	%	Events	n	%	Z
1	30	175	17.1%	14	175	8.0%	2.58
2	55	353	15.6%	37	347	10.7%	1.93
3	84	532	15.8%	55	518	10.6%	2.47
4	101	635	15.9%	71	630	11.3%	2.41

8.1 The CAPTURE data

We consider interim [4] and final [3] data from the CAPTURE trial. The interim data presented here is from finalized datasets rather than the actual data that was analyzed at the time of interim analyses. Also, we take a binomial analysis approach here using the method of Miettinen and Nurminen [13]; the original study analyses used the logrank test. The CAPTURE study was orginally designed using symmetric bounds, and the final planned sample size was 1400 patients. The trial was stopped after analyzing the data from 1050 patients. Enrollment continued during follow-up, data entry and cleaning, adjudication and analysis of the data. By the time of the 1050 patient analysis was evaluated and the decision was made to stop the trial, a total of 1265 patients had been enrolled.

The Z-statistics in Table 4 can be computed with the `testBinomial` function as follows.

```
n1 <- c(175, 353, 532, 635)
n2 <- c(175, 347, 518, 630)
x1 <- c( 30,  55,  84, 101)
x2 <- c( 14,  37,  55,  71)
z <- testBinomial(x1=x1, x2=x2, n1=n1, n2=n2)
```

8.2 Testing significance of the CAPTURE data

The Z-statistics computed above can only be interpreted in context of the study design. The original design used a custom spending function which will has not been published and will not be discussed further here. We will use the default spending functions for `gsDesign()` along with using 80% power and starting with a fixed design sample size obtained using the Farrington and Manning [6] method. Largely because of the futility bound used for the default design, the sample size for this design is 1491 as opposed to the 1400 planned for CAPTURE.

```
n.fix <- nBinomial(p1=.15, p2=.1, beta=.2)
x <- gsDesign(k=4, n.fix=n.fix, beta=.2)
x$n.I
```

To reset the interim analyses to those actually performed in the CAPTURE trial, we re-run `gsDesign()`, resetting the timing of analyses and adding the 1265 patient analysis as an additional interim analysis.

```
xmod <- gsDesign(k=5, n.fix=n.fix, beta=.2,
                 n.I=c(ntot, x$n.I[4]), maxn.IPlan=x$n.I[4])
```

The resulting upper bounds are 3.18, 2.87, 2.52, 2.31, and 2.04. Thus, if analyses had been performed as above for the default design it would have demonstrated statistical significance at the n=1265 analysis had a Z-statistic of 2.41 which is greater than the boundary of 2.31 for that analysis. Note that this is a numerical example only, as the analysis at n=1265 was performed because of the interim treatment effect at n=1050. The distribution theory for group sequential design requires that timing of interims is independent of interim test statistics. We will provide a technical resolution of this problem in a discussion of sample size adaptation.

8.3 Stage-wise p-values

Fairbanks and Madsen [5] provide a method for computing p-values for a symmetric group sequential trial design once a boundary has been crossed. Here we will consider just p-values for positive efficacy findings for asymmetric designs. We assume for some i in $1, 2, \ldots, k$ that an upper bound is first crossed at analysis i with a test statistic values z_i. The stage-wise p-value uses the same computational method as $\alpha^+(0)$ from (6).

$$p_{SW} = P_0\{\{Z_i \geq z_i\} \cap_{j=1}^{i-1} \{Z_j < b_j\}\} \tag{29}$$

This formula can still be used for the final analysis when the upper bound is never crossed. This method of computing p-values emphasizes early positive results and de-emphasizes late results. No matter how positive a result is after the first analysis, the p-value associated with a positive result will not be smaller than a first analysis result that barely crosses its bound. There is no way to compute a p-value if, for some reason, you stop a trial early without crossing a bound. For the CAPTURE data analyzed according to the default design derived above, we compute a stagewise p-value of 0.11 as follows

```
> y <- gsProbability(k=4, theta=0, n.I=ntot, a=array(-20,4),
+                    b=c(xmod$upper$bound[1:3], z[4]))
> y$upper$prob
             [,1]
[1,] 0.0007264271
[2,] 0.0018577690
[3,] 0.0047510699
[4,] 0.0041621310
> sum(y$upper$prob)
[1] 0.01149740
```

8.4 Repeated confidence intervals and p-values

Repeated confidence intervals use the nominal Type I error at each interim analysis to compute confidence bounds in the usual fashion. For the binomial analysis of the CAPTURE trial we use Miettinen and Nurminen [13] confidence intervals.

```
y <- NULL
for(i in 1:4)
{    y <- c(y, ciBinomial(x1=x1[i], x2=x2[i], n1=n1[i], n2=n2[i],
           alpha=2 * pnorm(-xmod$upper$bound[i])))
}
```

9 Conditional power and B-values

9.1 Group sequential test statistics as sums of independent increments

In some cases, rather than working with Z_1, Z_2,...,Z_k as in Section 2.3, it is desirable to consider variables representing incremental sets of observations between analyses. This approach will be applied here to define conditional power and B-values, two common measures of interim results and boundaries. Letting $I_0 = n_0 = 0$ we define $Y_i = \sum_{j=n_{i-1}+1}^{n_i} X_j/\sqrt{I_i - I_{i-1}}$ for $i = 1, 2, \ldots, k$. This implies $Y_1, Y_2, ..., Y_k$ are independent and normally distributed with

$$Y_i \sim N(\sqrt{I_i - I_{i-1}}\theta, 1),\ i = 1, 2, ..., k. \tag{30}$$

For $i = 1, 2, ..., k$ if we let $w_i = \sqrt{I_i - I_{i-1}}$, note that

$$Z_i = \frac{\sum_{j=1}^{i} \sqrt{I_j - I_{j-1}} Y_j}{\sqrt{I_i}} = \frac{\sum_{j=1}^{i} w_j Y_j}{\sqrt{\sum_{j=1}^{i} w_j^2}}. \tag{31}$$

Finally, we define notation for independent increments between arbitrary analyses. Select i and j with $1 \le i < j \le k$ and let $Z_{i,j} = \sum_{m=n_i+1}^{n_j} X_m/\sqrt{I_j - I_i}$. Thus, $Z_{i,j} \sim N(\sqrt{I_j - I_i}\theta, 1)$ is independent of Z_i and

$$Z_j = \frac{\sqrt{I_i} Z_i + \sqrt{I_j - I_i} Z_{i,j}}{\sqrt{I_j}}. \tag{32}$$

By definition, for $i = 2, 3, ...k$,

$$Y_i = Z_{i-1,i}. \tag{33}$$

For the more general canonical form not defined using $X_1, X_2, ...$ we define $Y_1 = Z_1$ and for $1 \le j < i \le k$

$$Z_{j,i} = \frac{\sqrt{I_i} Z_i - \sqrt{I_j} Z_j}{\sqrt{I_i - I_j}}. \tag{34}$$

The variables $Z_{j,i}$ and Z_j are independent, as before, for any $1 \le j < i \le k$. We use (33) to define Y_i, $i = 2, 3, ..., k$. As before $Y_i \tilde{} N(\sqrt{I_i - I_{i-1}}\theta, 1)$, $1 < i \le k$, and these random variables are independent of each other.

9.2 Conditional power

As an alternative to β-spending, stopping rules for futility are interpreted by considering the conditional power of a positive trial given the value of a test statistic at an interim analysis. Thus, we consider the conditional probabities of boundary crossing for a group sequential design given an interim result. Assume $1 \le i < m \le j \le k$ and let z_i be any real value. Define

$$u_{m,j}(z_i) = \frac{u_j \sqrt{I_j} - z_i \sqrt{I_m}}{\sqrt{(I_j - I_m)}} \tag{35}$$

and

$$l_{m,j}(z_i) = \frac{l_j \sqrt{I_j} - z_i \sqrt{I_m}}{\sqrt{(I_j - I_m)}}. \tag{36}$$

Recall (32) and consider the conditional probabilities

$$\begin{aligned}\alpha_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \geqslant u_j\} \cap_{m=i+1}^{j-1} \{l_m < Z_m < u_m\}|Z_i = z_i\} \\ &= P_\theta\left\{\left\{\frac{\sqrt{I_i}z_i + \sqrt{I_j - I_i}Z_{i,j}}{\sqrt{I_j}} \geqslant u_j\right\} \cap_{m=i+1}^{j-1} \left\{l_m < \frac{\sqrt{I_i}z_i + \sqrt{I_j - I_i}Z_{i,m}}{\sqrt{I_j}}\right\} < u_m\right\} \\ &= P_\theta\{\{Z_{i,j} \geqslant u_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{l_{m,j}(z_i) < Z_{m,j} < u_{m,j}(z_i)\}\}.\end{aligned} \tag{37}$$

This last line is of the same general form as $\alpha_i(\theta)$ and can thus be computed in a similar fashion. For a non-binding bound, the same logic applied ignoring the lower bound yields

$$\begin{aligned}\alpha^+_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \geqslant u_j\} \cap_{m=i+1}^{j-1} \{Z_m < u_m\}|Z_i = z_i\} \\ &= P_\theta\{\{Z_{i,j} \geqslant u_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{Z_{m,j} < u_{m,j}(z_i)\}\}.\end{aligned} \tag{38}$$

Finally, the conditional probability of crossing a lower bound at analysis j given a test statistic z_i at analysis i is denoted by

$$\begin{aligned}\beta_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \leq l_j\} \cap_{m=i+1}^{j-1} \{l_m < Z_m < u_m\}|Z_i = z_i\} \\ &= P_\theta\{\{Z_{i,j} \leq l_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{l_{m,j}(z_i) < Z_{m,j} < u_{m,j}(z_i)\}\}.\end{aligned} \tag{39}$$

Since $\alpha^+_{i,j}(\theta|z_i)$ and $\beta_{i,j}(\theta|z_i)$ are of the same general form as $\alpha^+_i(\theta)$ and $\beta_i(\theta)$, respectively, they can be computed using the same tools.

9.3 B-values

Proschan, Lan and Wittes [18].

10 Bayesian design properties

All of the properties of group sequential designs we have considered so far have depended on knowing an exact value θ measuring treatment effect. Answering some important questions requires taking into account the uncertainty of *a priori* knowledge of θ:

- What is the probability of success of the trial?
- For a sponsor who remains blinded to interim results, how is this probability of success modified by knowing only that boundaries have not been crossed at a given interim analysis?
- Predictive power is conditional power given an exact interim result averaged over values of θ by a prior distribution.
- In decision theory, we would want to express the value of a trial in that we should give up on experimental agents that do not work with a minimal investment and get drugs to market as quickly as possible with an appropriate evaluation of risk and benefit.

Examples in this section compute answers to all of these questions when framed in terms of a particular trial.

10.1 Probability of success

The probability of a positive trial depends on the distribution of outcomes in the control and experimental groups. The probability of a positive trial given a particular parameter value θ was defined in (7) and (8) as

$$\alpha(\theta) = \sum_{i=1}^{K} P_\theta\{\{Z_i \geq b_i\} \cap_{j=1}^{i-1} \{a_j < Z_j < b_j\}\}. \tag{40}$$

If we consider θ to have a given prior distribution at the beginning of the trial, we can compute an unconditional probability of success for the trial. In essence, since we do not know if the experimental treatment works better than control treatment, we assign some prior beliefs about the likelihood that experimental is better than control and use those along with the size of the trial to compute the probability of success. The prior distribution for θ can be discrete or continuous. If the distribution is discrete, we define $m + 1$ values $\theta_0 < \theta_2 \ldots < \theta_m$ and assign prior probabilities $P\{\theta = \theta_j\}$, $0 \leq j \leq m$ such that $\sum_{j=1}^{m} P\{\theta_j\} = 1$. The probability of success for the trial is then defined as

$$POS = \sum_{j=0}^{m} P\{\theta = \theta_j\}\alpha(\theta_j) \tag{41}$$

The simplest practical example is perhaps assuming a 2-point prior where the prior probability of the difference specified in the alternate hypothesis used to power the trial is p and the prior probability of no treatment difference is $1 - p$. Suppose the trial is designed to have power $1 - \beta = \alpha(\delta)$ when $\theta = \delta$ and Type I error $\alpha = \alpha(0)$ when $\theta = 0$. Then the probability of success for the trial is

$$\text{POS} = p \times (1 - \beta) + (1 - p) \times \alpha.$$

Assuming a 70% prior probability of a treatment effect δ, a 30% prior probability of no treatment effect, power of 90% and Type I error of 2.5% results in an unconditional probability of a positive trial of $.7 \times .9 + .3 \times .025 = .6375$. In this case, it is arguable that POS should be defined as $.7 \times .9 = .63$ since the probability of a positive trial when $\theta = 0$ should not be considered a success. This alternative definition becomes ambigous when the prior distribution for θ becomes more diffuse. We will address this issue below in the discussion of the value of a trial design.

We consider a slightly more ambitious example and show how to use `gsProbability()` to compute (41). We derive a design using `gsDesign()`, in this case using default parameters. We assume the possible parameter values are $\theta_i = i \times \delta$ where δ is the value of θ for which the trial is powered and $i = 0, 2, \ldots, 6$. The respective prior probabilities for these θ_i values are assumed to be 1/20, 2/20, 2/20, 3/20, 7/20, 3/20 and 2/20. We show the calculation and then explain in detail.

```
y <- gsDesign()
theta <- y$theta[2] * array(0:6)/4
ptheta <- c(1, 2, 2, 3, 7, 3, 2) / 20
x <- gsProbability(theta = theta, d=y)
one <- array(1, 3)
pos <- one %*% x$upper$prob %*% ptheta
pos
```

Note that `theta[2]` is the value δ for which the trial is powered as noted in the first example in the introduction [check this!]. This calculation could be written as a short macro:

```
"pos" <- function(x, theta, ptheta)
{  x <- gsProbability(theta = theta, d=x)
   one <- array(1, x$k)
   as.real(one %*% x$upper$prob %*% ptheta)
}
```

which would reduce the last 4 lines of code above to `pos(x, theta, ptheta)`. For those not familiar with it `%*%` represents matrix multiplication, and thus the code `one %*% x$upper$prob %*% ptheta` is doing the computation

$$\sum_{j=0}^{m} P\{\theta_j\} \sum_{i=0}^{K} \alpha_i(\theta_j).$$

For a prior distribution that is continuous with density $f(\theta)$ we define

$$POS = \int_{-\infty}^{\infty} f(\theta)\alpha(\theta)d\theta. \tag{42}$$

Numerical integration is required to implement this calculation, but we can still use the `pos()` function just defined. For instance, assuming $\theta \sim N(\mu = \delta, \sigma^2 = (\delta/2)^2)$ we can use `normalGrid()` from the `gsDesign` package to generate a grid and normal densities on that grid that can be used to perform numerical integration. For this particular case

```
y <- gsDesign()
delta <- y$theta[2]
g <- normalGrid(mu=delta, sigma=delta / 2)
plot(g$z, g$wgts)
pos(y, g$z, g$wgts)
```

This computation yields a probability of success of .748. The `normalGrid()` function is a lower-level function used by `gsProbability()` and `gsDesign()` that is normally obscured from the user. For Bayesian computations with a normal prior distribution, such as here, it can be quite useful as in the above example. The values returned above in `g$wgts` are the normal density multiplied by weights generated using Simpson's rule. The plot generated by the above code shows that these values alternate as higher and lower values about a smooth function. If you compute `sum(g$wgts)` you will confirm that the approximated integral over the real line of this density is 1, as desired.

To practice with this, assume a more pessimistic prior with $\mu = \sigma = \delta/2$ to obtain a probability of success of .428.

We generalize (41) and (42) by letting $F()$ denote the cumulative distribution function for θ and write

$$POS = \int_{-\infty}^{\infty} \alpha(\theta)dF(\theta). \tag{43}$$

This notation will be used in further discussions to provide formulas applicable to both continuous and discrete distributions.

10.2 Updating probability of success based on blinded results

Futility bounds are often set up to be informative about emerging treatment effects. That is, a positive trend is often required to pass a futility bound. Efficacy bounds usually are only informative to a lesser extent, but knowing that an efficacy bound has not been crossed at an interim analysis generally rules out an extremely positive effect after early interim analyses and a moderately positive effect later in the trial. Thus, knowing that a trial has not crossed a futility or efficacy bound at an interim analysis can be helpful in updating the probability of success we have computed above. In this subsection we will restrict consideration to the probability of ultimate trial success. For $1 \leq i < K$ we denote the event that no boundary has been crossed at or before analysis i by

$$A_i = \cap_{j=1}^{i-1}\{a_j < Z_j < b_j\} \tag{44}$$

The probability of observing A_i is

$$P\{A_i\} = 1 - \int \sum_{j=1}^{i} (\alpha_j(\theta) + \beta_j(\theta)) dF(\theta) \tag{45}$$

Letting B denote the event that the trial crosses an upper bound at or before the end of the trial and before crossing a lower bound compute

$$P\{A_i \cap B\} = \int \sum_{j=i+1}^{K} \alpha_j(\theta) dF(\theta) \tag{46}$$

Based on these 2 equations, we can now compute for $1 \leq i < K$ the conditional probability of a positive trial given that no boundary has been crossed by interim i as

$$P\{B|A_i\} = P\{A_i \cap B\}/P\{A_i\}. \tag{47}$$

Calculations for the 2 probabilities needed are quite similar to the `pos()` function considered in the previous subsection. We write a function to compute this conditional probability of success based on not crossing a boundary at or before analysis `i` for a design `x` derived from `gsDesign()` or `gsProbability()` and a prior distribution expressed by a vector `theta` and its associated probabilities in `ptheta`:

```
"cpos" <- function(i, x, theta, ptheta)
{   x <- gsProbability(theta = theta, d=x)
    v <- c(array(1, i), array(0, (x$k - i)))
    pAi <- 1 - as.real(v %*% (x$upper$prob + x$lower$prob) %*% ptheta)
    v <- 1 - v
    pAiB <- as.real(v %*% x$upper$prob %*% ptheta)
    pAiB / pAi
}
```

For the case considered previously with $\theta \sim N(\mu = \delta, \sigma = \delta/2)$ and a default design we had a probability of success of .748. The following code shows that if neither trial boundary is crossed at the first interim, the updated (posterior) probability of success is .733. After 2 analyses, the posterior probability of success is .669.

```
y <- gsDesign()
delta <- y$theta[2]
g <- normalGrid(bounds=c(-30, 30) * delta / 2, mu=delta, sigma=delta / 2)
pos(y, theta=g$z, ptheta=g$wgts)
cpos(1, y, theta=g$z, ptheta=g$wgts)
cpos(2, y, theta=g$z, ptheta=g$wgts)
```

To ensure a higher conditional probability of success for the trial, a more aggressive futility bound could be employed at the expense of requiring an increased sample size to maintain the desired power. The code `y$n.I` shows that the default design requires an inflation factor of 1.07 for the sample size compared to a fixed design with the same Type I error and power. By employing an aggressive Hwang-Shih-DeCani spending function with $\gamma = 1$ for the futility bound, the sample size inflation factor is increased to 1.23 ((y ¡- gsDesign(sflpar=1))). The probability of success for this design at the beginning of the trial using the same prior distribution as above is still .748, but the conditional probability of success after not hitting a boundary by interim 1 (interim 2) is now .788 (.761). While there are many other considerations in choosing a futility bound and other prior distributions give other results, this example suggests that something more aggressive than the default futility bound in `gsDesign()` may be desirable.

10.3 Calculating the value of a clinical trial design

Here we generalize the concept of the probability of success of a trial given above to the value of a trial. We assume that a trial that stops with a positive result with information I_i at analysis i of a trial when the true treatment effect is θ can be given by a function $u(\theta, I_i)$, $1 \le i \le K$. Now the formula for probability of success can be generalized to

$$U = \int_{-\infty}^{\infty} f(\theta) \sum_{i=1}^{K} \alpha_i(\theta) u(\theta, I_i) d\theta. \tag{48}$$

A more general formula that incorporates a costs that are incurred whether or not a trial is positive. If this formula also discounted future costs and benefits to present-day values, it would be termed a net present value and can be defined in a simplified form as

$$NPV = \int_{-\infty}^{\infty} f(\theta) \sum_{i=1}^{K} \alpha_i(\theta) u(\theta, I_i) - (\alpha_i(\theta) + \beta_i(\theta)) c(\theta, I_i) (d\theta. \tag{49}$$

While the underlying computations are not much more difficult to allow the utility and cost functions $u()$ and to depend on the test statistic at the time the trial stops, this capability has not been implemented in the package at this time. Arguably, however, the true value of a treatment depends on its true benefit rather than exactly what is observed in a clinical trial.

The following function computes the integral (48).

```
value <- function(x, theta, ptheta, u)
{  x <- gsProbability(theta = theta, d=x)
   one <- array(1, x$k)
   as.real(one %*% (u * x$upper$prob) %*% ptheta)
}
```

For this implementation, `u` must be a scalar, a vector of length `x$k` or a matrix of the same dimension as `x$upper$prob` (`k` rows and `length(theta)` columns) rather than a function. We return to an example from the previous section. Assuming $\theta \sim N(\mu = \delta, \sigma^2 = (\delta/2)^2)$ we showed that the probability of success as defined in (41) is .748. We will now change this definition so that a trial is considered a success only if it is positive and the true value of $\theta > \delta/2$. This is computed as .521 as follows:

```
x <- gsDesign()
delta <- x$theta[2]
g <- normalGrid(mu=delta, sigma=delta / 2)
u <- 1 * (g$z > delta/2)
value(x, theta=g$z, ptheta=g$wgts, u=u)
```

We finish with an example computing a futilty bound that optimizes the value of a design. We will assume a fixed efficacy bound is used and will select an optimal spending function for the lower bound from a one-parameter family. We allow the user to specify the number of interim analyses as well as the desired Type I and Type II error and the prior distribution for the treatment effect. We will assume a function `f(n, theta)` provides the value of a trial that stops for a positive result after enrolling `n` patients when the true treatment effect is `theta`.

```
gsDesign(k=3, test.type=4, alpha=0.025, beta=0.1, astar=0,
         delta=0, n.fix=1, timing=1, sfu=sfHSD, sfupar=-4,
         sfl=sfHSD, sflpar=-2, tol=0.000001, r=18, n.I = 0, maxn.IPlan = 0)
```

```
function lbValue(sflpar=-2, k=3, test.type=4, alpha=0.025, beta=0.1, astar=0,
         delta=0, n.fix=1, timing=1, sfu=sfHSD, sfupar=-4,
         sfl=sfHSD, tol=0.000001, r=18, f, theta, ptheta)
{   x <- gsDesign(
    u <- f()
    value(x, theta=theta, ptheta=ptheta, u=u)
}
```

A Package help files

---

gsDesign package overview

1.0 Group Sequential Design

Description

gsDesign is a package for deriving and describing group sequential designs. The package allows particular flexibility for designs with alpha- and beta-spending. Many plots are available for describing design properties.

Details

Package:	gsDesign
Version:	2
License:	GPL (version 2 or later)

Index:

```
Wang-Tsiatis Bounds     5.0: Wang-Tsiatis Bounds
checkScalar             6.0: Utility functions to verify variable properties
```

The gsDesign package supports group sequential clinical trial design. While there is a strong focus on designs using α- and β-spending functions, Wang-Tsiatis designs, including O'Brien-Fleming and Pocock designs, are also available. The ability to design with non-binding futility rules allows control of Type I error in a manner acceptable to regulatory authorities when futility bounds are employed.

The routines are designed to provide simple access to commonly used designs using default arguments. Standard, published spending functions are supported as well as the ability to write custom spending functions. A `gsDesign` class is defined and returned by the `gsDesign()` function. A plot function for this class provides a wide variety of plots: boundaries, power, estimated treatment effect at boundaries, conditional power at boundaries, spending function plots, expected sample size plot, and B-values at boundaries. Using function calls to access the package routines provides a powerful capability to derive designs or output formatting that could not be anticipated through a gui interface. This enables the user to easily create designs with features they desire, such as designs with minimum expected sample size.

Thus, the intent of the gsDesign package is to easily create, fully characterize and even optimize routine group sequential trial designs as well as provide a tool to evaluate innovative designs.

Author(s)

Keaven Anderson

Maintainer: Keaven Anderson `<keaven_anderson@merck.com>`

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Proschan, MA, Lan, KKG, Wittes, JT (2006), *Statistical Monitoring of Clinical Trials. A Unified Approach*. New York: Springer.

See Also

`gsDesign`, `gsProbability`

Examples

```
# assume a fixed design (no interim) trial with the same endpoint
# requires 200 subjects for 90% power at alpha=.025, one-sided
x <- gsDesign(n.fix=200)
plot(x)
```

Description

`gsDesign()` is used to find boundaries and trial size required for a group sequential design.

Usage

```
gsDesign(k=3, test.type=4, alpha=0.025, beta=0.1, astar=0,
         delta=0, n.fix=1, timing=1, sfu=sfHSD, sfupar=-4,
         sfl=sfHSD, sflpar=-2, tol=0.000001, r=18, n.I = 0,
         maxn.IPlan = 0)

print.gsDesign(x,...)
```

Arguments

`k`	Number of analyses planned, including interim and final.
`test.type`	`1`=one-sided `2`=two-sided symmetric `3`=two-sided, asymmetric, beta-spending with binding lower bound `4`=two-sided, asymmetric, beta-spending with non-binding lower bound `5`=two-sided, asymmetric, lower bound spending under the null hypothesis with binding lower bound `6`=two-sided, asymmetric, lower bound spending under the null hypothesis with non-binding lower bound. See details, examples and manual.
`alpha`	Type I error, always one-sided. Default value is 0.025.
`beta`	Type II error, default value is 0.1 (90% power).
`astar`	Normally not specified. If `test.type=5` or `6`, `astar` specifies the total probability of crossing a lower bound at all analyses combined. This will be changed to 1−`alpha` when default value of 0 is used. Since this is the expected usage, normally `astar` is not specified by the user.
`delta`	Standardized effect size. See details and examples.
`n.fix`	Sample size for fixed design with no interim; used to find maximum group sequential sample size. See details and examples.
`timing`	Sets relative timing of interim analyses. Default of 1 produces equally spaced analyses. Otherwise, this is a vector of length `k` or `k-1`. The values should satisfy `0 < timing[1] < timing[2] < ... < timing[k-1] < timing[k]=1`.
`sfu`	A spending function or a character string indicating a boundary type (that is, "WT" for Wang-Tsiatis bounds, "OF" for O'Brien-Fleming bounds and "Pocock" for Pocock bounds). For one-sided and symmetric two-sided testing is used to completely specify spending (`test.type=1, 2`), `sfu`. The default value is `sfHSD` which is a Hwang-Shih-DeCani spending function. See details, Spending function overview, manual and examples.

`sfupar`	Real value, default is −4 which is an O'Brien-Fleming-like conservative bound when used with the default Hwang-Shih-DeCani spending function. This is a real-vector for many spending functions. The parameter `sfupar` specifies any parameters needed for the spending function specified by `sfu`; this will be ignored for spending functions (`sfLDOF`, `sfLDPocock`) or bound types ("OF", "Pocock") that do not require parameters.
`sfl`	Specifies the spending function for lower boundary crossing probabilities when asymmetric, two-sided testing is performed (`test.type = 3`, `4`, `5`, or `6`). Unlike the upper bound, only spending functions are used to specify the lower bound. The default value is `sfHSD` which is a Hwang-Shih-DeCani spending function. The parameter `sfl` is ignored for one-sided testing (`test.type=1`) or symmetric 2-sided testing (`test.type=2`). See details, spending functions, manual and examples.
`sflpar`	Real value, default is −2, which, with the default Hwang-Shih-DeCani spending function, specifies a less conservative spending rate than the default for the upper bound.
`tol`	Tolerance for error (default is 0.000001). Normally this will not be changed by the user. This does not translate directly to number of digits of accuracy, so use extra decimal places.
`r`	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally `r` will not be changed by the user.
`n.I`	Used for re-setting bounds when timing of analyses changes from initial design; see examples.
`maxn.IPlan`	Used for re-setting bounds when timing of analyses changes from initial design; see examples.
`x`	In `print.gsDesign` this is an object of class gsDesign.
`...`	This should allow optional arguments that are standard when calling `print`.

Details

Many parameters normally take on default values and thus do not require explicit specification. One- and two-sided designs are supported. Two-sided designs may be symmetric or asymmetric. Wang-Tsiatis designs, including O'Brien-Fleming and Pocock designs can be generated. Designs with common spending functions as well as other built-in and user-specified functions for Type I error and futility are supported. Type I error computations for asymmetric designs may assume binding or non-binding lower bounds. The print function has been extended using `print.gsDesign` to print `gsDesign` objects; see examples.

The user may ignore the structure of the value returned by `gsDesign()` if the standard printing and plotting suffice; see examples.

`delta` and `n.fix` are used together to determine what sample size output options the user seeks. The default, `delta=0` and `n.fix=1`, results in a 'generic' design that may be used with any sampling situation. Sample size ratios are provided and the user multiplies these times the sample size for a fixed design to obtain the corresponding group sequential analysis times. If `delta>0`, `n.fix` is ignored, and `delta` is taken as the standardized effect size - the signal to noise ratio for a single observation; for example, the mean divided by the standard deviation for a one-sample normal problem. In this case, the sample size at each analysis is computed.

When `delta=0` and `n.fix>1`, `n.fix` is assumed to be the sample size for a fixed design with no interim analyses. See examples below.

Following are further comments on the input argument `test.type` which is used to control what type of error measurements are used in trial design. The manual may also be worth some review in order to see actual formulas for boundary crossing probabilities for the various options. Options 3 and 5 assume the trial stops if the lower bound is crossed for Type I and Type II error computation (binding lower bound). For the purpose of computing Type I error, options 4 and 6 assume the trial continues if the lower bound is crossed (non-binding lower bound); that is a Type I error can be made by crossing an upper bound after crossing a previous lower bound. Beta-spending refers to error spending for the lower bound crossing probabilities under the alternative hypothesis (options 3 and 4). In this case, the final analysis lower and upper boundaries are assumed to be the same. The appropriate total beta spending (power) is determined by adjusting the maximum sample size through an iterative process for all options. Since options 3 and 4 must compute boundary crossing probabilities under both the null and alternative hypotheses, deriving these designs can take longer than other options. Options 5 and 6 compute lower bound spending under the null hypothesis.

Value

An object of the class `gsDesign`. This class has the following elements and upon return from `gsDesign()` contains:

`k`	As input.
`test.type`	As input.
`alpha`	As input.
`beta`	As input.
`astar`	As input, except when `test.type=5` or `6` and `astar` is input as 0; in this case `astar` is changed to `1-alpha`.
`delta`	The standardized effect size for which the design is powered. Will be as input to `gsDesign()` unless it was input as 0; in that case, value will be computed to give desired power for fixed design with input sample size `n.fix`.
`n.fix`	Sample size required to obtain desired power when effect size is `delta`.
`timing`	A vector of length `k` containing the portion of the total planned information or sample size at each analysis.
`tol`	As input.
`r`	As input.
`upper`	Upper bound spending function, boundary and boundary crossing probabilities under the NULL and alternate hypotheses. See Spending function overview and manual for further details.
`lower`	Lower bound spending function, boundary and boundary crossing probabilities at each analysis. Lower spending is under alternative hypothesis (beta spending) for `test.type=3` or `4`. For `test.type=2`, `5` or `6`, lower spending is under the null hypothesis. For `test.type=1`, output value is `NULL`. See Spending function overview and manual.
`n.I`	Vector of length `k`. If values are input, same values are output. Otherwise, `n.I` will contain the sample size required at each analysis to achieve desired `timing` and `beta` for the output value of `delta`. If `delta=0` was input, then this is the sample size required for the specified group sequential design when

a fixed design requires a sample size of `n.fix`. If `delta=0` and `n.fix=1` then this is the relative sample size compared to a fixed design; see details and examples.

`maxn.IPlan` As input.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

gsDesign package overview, Plots for group sequential designs, `gsProbability`, Spending function overview, Wang-Tsiatis Bounds

Examples

```
#  symmetric, 2-sided design with O'Brien-Fleming-like boundaries
#  lower bound is non-binding (ignored in Type I error computation)
#  sample size is computed based on a fixed design requiring n=800
x <- gsDesign(k=5, test.type=2, n.fix=800)

# note that "x" below is equivalent to print(x) and print.gsDesign(x)
x
plot(x)
plot(x, plottype=2)

# Assuming after trial was designed actual analyses occurred after
# 300, 600, and 860 patients, reset bounds
y <- gsDesign(k=3, test.type=2, n.fix=800, n.I=c(300,600,860),
   maxn.IPlan=x$n.I[x$k])
y

#  asymmetric design with user-specified spending that is non-binding
#  sample size is computed relative to a fixed design with n=1000
sfup <- c(.033333, .063367, .1)
sflp <- c(.25, .5, .75)
timing <- c(.1, .4, .7)
x <- gsDesign(k=4, timing=timing, sfu=sfPoints, sfupar=sfup, sfl=sfPoints,
                   sflpar=sflp,n.fix=1000)
x
plot(x)
plot(x, plottype=2)

# same design, but with relative sample sizes
gsDesign(k=4, timing=timing, sfu=sfPoints, sfupar=sfup, sfl=sfPoints,
```

```
sflpar=sflp)
```

---

gsBound *2.6: Boundary derivation - low level*

Description

gsBound() and gsBound1() are lower-level functions used to find boundaries for a group sequential design. They are not recommended (especially gsBound1()) for casual users. These functions do not adjust sample size as gsDesign() does to ensure appropriate power for a design.

gsBound() computes upper and lower bounds given boundary crossing probabilities assuming a mean of 0, the usual null hypothesis. gsBound1() computes the upper bound given a lower boundary, upper boundary crossing probabilities and an arbitrary mean (theta).

Usage

```
gsBound(I, trueneg, falsepos, tol=0.000001, r=18)
gsBound1(theta, I, a, probhi, tol=0.000001, r=18, printerr=0)
```

Arguments

Note that all vector arguments should have the same length which will be denoted here as k.

theta	Scalar containing mean (drift) per unit of statistical information.
I	Vector containing statistical information planned at each analysis.
a	Vector containing lower bound that is fixed for use in gsBound1.
trueneg	Vector of desired probabilities for crossing upper bound assuming mean of 0.
falsepos	Vector of desired probabilities for crossing lower bound assuming mean of 0.
probhi	Vector of desired probabilities for crossing upper bound assuming mean of theta.
tol	Tolerance for error (scalar; default is 0.000001). Normally this will not be changed by the user. This does not translate directly to number of digits of accuracy, so use extra decimal places.
r	Single integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally r will not be changed by the user.
printerr	If this scalar argument set to 1, this will print messages from underlying C program. Mainly intended to notify user when an output solution does not match input specifications. This is not intended to stop execution as this often occurs when deriving a design in gsDesign that uses beta-spending.

Details

The function gsBound1() requires special attention to detail and knowledge of behavior when a design corresponding to the input parameters does not exist.

Value

Both routines return a list. Common items returned by the two routines are:

`k`	The length of vectors input; a scalar.
`theta`	As input in `gsBound1()`; 0 for `gsBound()`.
`I`	As input.
`a`	For `gsbound1`, this is as input. For `gsbound` this is the derived lower boundary required to yield the input boundary crossing probabilities under the null hypothesis.
`b`	The derived upper boundary required to yield the input boundary crossing probabilities under the null hypothesis.
`tol`	As input.
`r`	As input.
`error`	Error code. 0 if no error; greater than 0 otherwise.
`rates`	a list containing two items:
`falsepos`	vector of upper boundary crossing probabilities as input.
`trueneg`	vector of lower boundary crossing probabilities as input.
`problo`	vector of lower boundary crossing probabilities; computed using input lower bound and derived upper bound.
`probhi`	vector of upper boundary crossing probabilities as input.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

gsDesign package overview, `gsDesign`, `gsProbability`

Examples

```
# set boundaries so that probability is .01 of first crossing
# each upper boundary and .02 of crossing each lower boundary
# under the null hypothesis
x <- gsBound(I=c(1, 2, 3)/3, trueneg=array(.02, 3),
             falsepos=array(.01, 3))
x

#  use gsBound1 to set up boundary for a 1-sided test
```

```
x <- gsBound1(theta= 0, I=c(1, 2, 3) / 3, a=array(-20, 3),
              probhi=c(.001, .009, .015))
x$b

# check boundary crossing probabilities with gsProbability
y <- gsProbability(k=3, theta=0, n.I=x$I, a=x$a, b=x$b)$upper$prob

#  Note that gsBound1 only computes upper bound
#  To get a lower bound under a parameter value theta:
#      use minus the upper bound as a lower bound
#      replace theta with -theta
#      set probhi as desired lower boundary crossing probabilities
#  Here we let set lower boundary crossing at 0.05 at each analysis
#  assuming theta=2.2
y <- gsBound1(theta=-2.2, I=c(1, 2, 3)/3, a= -x$b,
              probhi=array(.05, 3))
y$b

#  Now use gsProbability to look at design
#  Note that lower boundary crossing probabilities are as
#  specified for theta=2.2, but for theta=0 the upper boundary
#  crossing probabilities are smaller than originally specified
#  above after first interim analysis
gsProbability(k=length(x$b), theta=c(0, 2.2), n.I=x$I, b=x$b, a= -y$b)
```

`gsProbability` *2.2: Boundary Crossing Probabilities*

Description

Computes power/Type I error and expected sample size for a group sequential design across a selected set of parameter values for a given set of analyses and boundaries. The print function has been extended using `print.gsProbability` to print `gsProbability` objects; see examples.

Usage

```
gsProbability(k=0, theta, n.I, a, b, r=18, d=NULL)
```

Arguments

`k`	Number of analyses planned, including interim and final.
`theta`	Vector of standardized effect sizes for which boundary crossing probabilities are to be computed.
`n.I`	Sample size or relative sample size at analyses; vector of length k. See `gsDesign` and manual.
`a`	Lower bound cutoffs (z-values) for futility or harm at each analysis, vector of length k.
`b`	Upper bound cutoffs (z-values) for futility at each analysis; vector of length k.
`r`	Control for grid as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Normally this will not be changed by the user.

d	If not NULL, this should be an object of type gsDesign returned by a call to gsDesign(). When this is specified, the values of k, n.I, a, b, and r will be obtained from d and only theta needs to be specified by the user.

Details

Depending on the calling sequence, an object of class gsProbability or class gsDesign is returned. If it is of class gsDesign then the members of the object will be the same as described in gsDesign. If d is input as NULL (the default), all other arguments (other than r) must be specified and an object of class gsProbability is returned. If d is passed as an object of class gsProbability or gsDesign the only other argument required is theta; the object returned has the same class as the input d. On output, the values of theta input to gsProbability will be the parameter values for which the design is characterized.

Value

k	As input.
theta	As input.
n.I	As input.
lower	A list containing two elements: bound is as input in a and prob is a matrix of boundary crossing probabilities. Element i,j contains the boundary crossing probability at analysis i for the j-th element of theta input. All boundary crossing is assumed to be binding for this computation; that is, the trial must stop if a boundary is crossed.
upper	A list of the same form as lower containing the upper bound and upper boundary crossing probabilities.
en	A vector of the same length as theta containing expected sample sizes for the trial design corresponding to each value in the vector theta.
r	As input.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Plots for group sequential designs, gsDesign, gsDesign package overview

Examples

```
# making a gsDesign object first may be easiest...
x <- gsDesign()

# take a look at it
x

# default plot for gsDesign object shows boundaries
plot(x)

# plottype=2 shows boundary crossing probabilities
plot(x, plottype=2)

# now add boundary crossing probabilities and
# expected sample size for more theta values
y <- gsProbability(d=x, theta=x$delta*seq(0, 2, .25))
class(y)

# note that "y" below is equivalent to print(y) and
# print.gsProbability(y)
y

# the plot does not change from before since this is a
# gsDesign object; note that theta/delta is on x axis
plot(y, plottype=2)

# now let's see what happens with a gsProbability object
z <- gsProbability(k=3, a=x$lower$bound, b=x$upper$bound,
    n.I=x$n.I, theta=x$delta*seq(0, 2, .25))

# with the above form,  the results is a gsProbability object
class(z)
z

# default plottype is now 2
# this is the same range for theta, but plot now has theta on x axis
plot(z)
```

`plot.gsDesign` *2.3: Plots for group sequential designs*

Description

The `plot()` function has been extended to work with objects returned by `gsDesign()` and `gsProbability()`. For objects of type `gsDesign`, seven types of plots are provided: z-values at boundaries (default), power, estimated treatment effects at boundaries, conditional power at boundaries, spending functions, expected sample size, and B-values at boundaries. For objects of type `gsProbability` plots are available for z-values at boundaries, power (default), estimated treatment effects at boundaries, conditional power, expected sample size and B-values at boundaries.

Usage

```
plot.gsProbability(x, plottype=2, ...)
plot.gsDesign(x, plottype=1, ...)
```

Arguments

x	Object of class gsDesign for plot.gsDesign() or gsProbability for plot.gsProbability().
plottype	1=boundary plot (default for gsDesign), 2=power plot (default for gsProbability), 3=estimated treatment effect at boundaries, 4=conditional power at boundaries, 5=spending function plot (only available if class(x)=="gsDesign"), 6=expected sample size plot, and 7=B-values at boundaries. Character values for plottype may also be entered: "Z" for plot type 1, "power" for plot type 2, "thetahat" for plot type 3, "CP" for plot type 4, "sf" for plot type 5, "ASN", "N" or "n" for plot type 6, and "B", "B-val" or "B-value" for plot type 7.
...	This allows many optional arguments that are standard when calling plot. Other arguments include: theta which is used for plottype=2, 4, 6; normally defaults will be adequate; see details. ses=TRUE which applies only when plottype=3 and class(x)=="gsDesign"; indicates that estimated standardized effect size at the boundary is to be plotted rather than the actual estimate. xval="Default" which is only effective when plottype=2 or 6. Appropriately scaled (reparameterized) values for x-axis for power and expected sample size graphs; see details.

Details

The intent is that many standard plot() parameters will function as expected; exceptions to this rule exist. In particular, main, xlab, ylab, lty, col, lwd, type, pch, cex have been tested and work for most values of plottype; one exception is that type="l" cannot be overridden when plottype=2. Default values for labels depend on plottype and the class of x.

Note that there is some special behavior for values plotted and returned for power and expected sample size (ASN) plots for a gsDesign object. A call to x<-gsDesign() produces power and expected sample size for only two theta values: 0 and x$delta. The call plot(x, plottype="Power") (or plot(x,plottype="ASN") for a gsDesign object produces power (expected sample size) curves and returns a gsDesign object with theta values determined as follows. If theta is non-null on input, the input value(s) are used. Otherwise, for a gsProbability object, the theta values from that object are used. For a gsDesign object where theta is input as NULL (the default), theta=seq(0,2,.05)*x$delta) is used. For a gsDesign object, the x-axis values are rescaled to theta/x$delta and the label for the x-axis *theta/delta*. For a gsProbability object, the values of theta are plotted and are labeled as *theta*. See examples below.

Estimated treatment effects at boundaries are computed dividing the Z-values at the boundaries by the square root of `n.I` at that analysis.

Spending functions are plotted for a continuous set of values from 0 to 1. This option should not be used if a boundary is used or a pointwise spending function is used (`sfu` or `sfl="WT", "OF", "Pocock"` or `sfPoints`).

Conditional power is computed using the function `gsBoundCP()`. The default input for this routine is `theta="thetahat"` which will compute the conditional power at each bound using the estimated treatment effect at that bound. Otherwise, if the input is `gsDesign` object conditional power is computed assuming `theta=x$delta`, the original effect size for which the trial was planned.

Average sample number/expected sample size is computed using `n.I` at each analysis times the probability of crossing a boundary at that analysis. If no boundary is crossed at any analysis, this is counted as stopping at the final analysis.

B-values are Z-values multiplied by `sqrt(t)=sqrt(x$n.I/n$n.I[x$k])`. Thus, the expected value of a B-value at an analysis is the true value of *theta* multiplied by the proportion of total planned observations at that time. See Proschan, Lan and Wittes (2006).

Value

An object of `class(x)`; in many cases this is the input value of `x`, while in others `x$theta` is replaced and corresponding characteristics computed; see details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Proschan, MA, Lan, KKG, Wittes, JT (2006), *Statistical Monitoring of Clinical Trials. A Unified Approach*. New York: Springer.

See Also

gsDesign package overview, `gsDesign`, `gsProbability`

Examples

```
#  symmetric, 2-sided design with O'Brien-Fleming-like boundaries
#  lower bound is non-binding (ignored in Type I error computation)
#  sample size is computed based on a fixed design requiring n=100
x <- gsDesign(k=5, test.type=2, n.fix=100)
x

# the following translate to calls to plot.gsDesign since x was
# returned by gsDesign; run these commands one at a time
```

```
plot(x)
plot(x, plottype=2)
plot(x, plottype=3)
plot(x, plottype=4)
plot(x, plottype=5)
plot(x, plottype=6)
plot(x, plottype=7)

#  choose different parameter values for power plot
#  start with design in x from above
y <- gsProbability(k=5, theta=seq(0, .5, .025), x$n.I,
                   x$lower$bound, x$upper$bound)

# the following translates to a call to plot.gsProbability since
# y has that type
plot(y)
```

---

gsCP *2.4: Conditional Power Computation*

Description

gsCP() takes a given group sequential design, assumes an interim z-statistic at a specified interim analysis and computes boundary crossing probabilities at future planned analyses.

Usage

```
gsCP(x, theta=NULL, i=1, zi=0, r=18)
```

Arguments

x	An object of type gsDesign or gsProbability
theta	θ value(s) at which conditional power is to be computed; if NULL, an estimated value of θ based on the interim test statistic (zi/sqrt(x$n.I[i])) as well as at x$theta is computed.
i	analysis at which interim z-value is given
zi	interim z-value at analysis i (scalar)
r	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally r will not be changed by the user.

Details

See Conditional power section of manual for further clarification. See also Muller and Schaffer (2001) for background theory.

Value

An object of the class `gsProbability`. Based on the input design and the interim test statistic, the output object has bounds for test statistics computed based on observations after interim `i` that are equivalent to the original design crossing boundaries conditional on the interim test statistic value input. Boundary crossing probabilities are computed for the input θ values.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Muller, Hans-Helge and Schaffer, Helmut (2001), Adaptive group sequential designs for clinical trials: combining the advantages of adaptive and classical group sequential approaches. *Biometrics*;57:886-891.

See Also

`gsDesign`, `gsProbability`, `gsBoundCP`

Examples

```
# set up a group sequential design
x <- gsDesign(k=5)
x

# assuming a z-value of .5 at analysis 2, what are conditional
# boundary crossing probabilities for future analyses
# assuming theta values from x as well as a value based on the interim
# observed z
CP <- gsCP(x, i=2, zi=.5)
CP

# summing values for crossing future upper bounds gives overall
# conditional power for each theta value
CP$theta
CP$upper$prob
```

Description

`gsBoundCP()` computes the total probability of crossing future upper bounds given an interim test statistic at an interim bound. For each interim boundary, assumes an interim test statistic at the boundary and computes the probability of crossing any of the later upper boundaries.

Usage

```
gsBoundCP(x, theta="thetahat", r=18)
```

Arguments

`x`	An object of type `gsDesign` or `gsProbability`
`theta`	if `"thetahat"` and `class(x)!="gsDesign"`, conditional power computations for each boundary value are computed using estimated treatment effect assuming a test statistic at that boundary (`zi/sqrt(x$n.I[i])` at analysis `i`, interim test statistic `zi` and interim sample size/statistical information of `x$n.I[i]`). Otherwise, conditional power is computed assuming the input scalar value `theta`.
`r`	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally `r` will not be changed by the user.

Details

See Conditional power section of manual for further clarification. See also Muller and Schaffer (2001) for background theory.

Value

A list containing two vectors, `CPlo` and `CPhi`.

`CPlo`	A vector of length `x$k-1` with conditional powers of crossing upper bounds given interim test statistics at each lower bound
`CPhi`	A vector of length `x$k-1` with conditional powers of crossing upper bounds given interim test statistics at each upper bound.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Muller, Hans-Helge and Schaffer, Helmut (2001), Adaptive group sequential designs for clinical trials: combining the advantages of adaptive and classical group sequential approaches. *Biometrics*;57:886-891.

See Also

`gsDesign`, `gsProbability`, `gsCP`

Examples

```
# set up a group sequential design
x <- gsDesign(k=5)
x

# compute conditional power based on interim treatment effects
gsBoundCP(x)

# compute conditional power based on original x$delta
gsBoundCP(x, theta=x$delta)
```

---

normalGrid *3.1: Normal Density Grid*

Description

normalGrid() is intended to be used for computation of the expected value of a function of a normal random variable. The function produces grid points and weights to be used for numerical integration.

Usage

```
normalGrid(r=18, bounds=c(0,0), mu=0, sigma=1)
```

Arguments

`r`	Control for grid points as in Jennison and Turnbull (2000), Chapter 19; default is 18. Range: 1 to 80. This might be changed by the user (e.g., `r=6` which produces 65 gridpoints compare to 185 points when `r=18`) when speed is more important than precision.
`bounds`	Range of integration. Real-valued vector of length 2. Default value of 0, 0 produces a range of + or - 6 standard deviations (6*sigma) from the mean (=mu).
`mu`	Mean of the desired normal distribution.
`sigma`	Standard deviation of the desired normal distribution.

Value

z	Grid points for numerical integration.
wgts	Weights to be used with grid points in z.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Examples

```
#  standard normal distribution
x <- normalGrid(r=3)
plot(x$z, x$wgts)

#  verify that numerical integration replicates sigma
#  get grid points and weights
x <- normalGrid(mu=2, sigma=3)

# compute squared deviation from mean for grid points
dev <- (x$z-2)^2

# multiply squared deviations by integration weights and sum
sigma2 <- sum(dev * x$wgts)

# square root of sigma2 should be sigma (3)
sqrt(sigma2)

# do it again with larger r to increase accuracy
x <- normalGrid(r=22, mu=2, sigma=3)
sqrt(sum((x$z-2)^2 * x$wgts))

# find expected sample size for default design with
# n.fix=1000
x <- gsDesign(n.fix=1000)
x

y <- normalGrid(r=3, mu=x$theta[2], sigma=x$theta[2] / 1.5)
z <- gsProbability(k=3, theta=y$z, n.I=x$n.I, a=x$lower$bound,
                   b=x$upper$bound)
z <- gsProbability(d=x, theta=y$z)
cat("Expected sample size averaged over normal ")
cat("prior distribution for theta with mu=",
   x$theta[2], "sigma=", x$theta[2]/1.5, ":",
   round(sum(z$en*y$wgt), 1), "\n")
```

```
plot(y$z, z$en, xlab="theta", ylab="E{N}",
   main="Expected sample size for different theta values")
```

`Binomial` *3.2: Testing, Confidence Intervals and Sample Size for Comparing Two Binomial Rates*

Description

Support is provided for sample size estimation, testing confidence intervals and simulation for fixed sample size trials (that is, not group sequential or adaptive) with two arms and binary outcomes. Both superiority and non-inferiority trials are considered. While all routines default to comparisons of risk-difference, options to base computations on risk-ratio and odds-ratio are also included.

`nBinomial()` computes sample size using the method of Farrington and Manning (1990) to derive sample size required to power a trial to test the difference between two binomial event rates. The routine can be used for a test of superiority or non-inferiority. For a design that tests for superiority `nBinomial()` is consistent with the method of Fleiss, Tytun, and Ury (but without the continuity correction) to test for differences between event rates. This routine is consistent with the Hmisc package routine `bsamsize` for superiority designs. Vector arguments allow computing sample sizes for multiple scenarios for comparative purposes.

`testBinomial()` computes a Z- or Chi-square-statistic that compares two binomial event rates using the method of Miettinen and Nurminen (1980). This can be used for superiority or non-inferiority testing. Vector arguments allow easy incorporation into simulation routines for fixed, group sequential and adaptive designs.

`ciBinomial()` computes confidence intervals for 1) the difference between two rates, 2) the risk-ratio for two rates or 3) the odds-ratio for two rates. This procedure provides inference that is consistent with `testBinomial()` in that the confidence intervals are produced by inverting the testing procedures in `testBinomial()`. The Type I error `alpha` input to `ciBinomial` is always interpreted as 2-sided.

`simBinomial()` performs simulations to estimate the power for a Miettinin and Nurminen (1980) test comparing two binomial rates for superiority or non-inferiority. As noted in documentation for `bpower.sim()` in the HMisc package, by using `testBinomial()` you can see that the formulas without any continuity correction are quite accurate. In fact, Type I error for a continuity-corrected test is significantly lower (Gordon and Watson, 1996) than the nominal rate. Thus, as a default no continuity corrections are performed.

Usage

```
nBinomial(p1, p2, alpha=.025, beta=0.1, delta0=0, ratio=1,
          sided=1, outtype=1, scale="Difference")
testBinomial(x1, x2, n1, n2, delta0=0, chisq=0, adj=0,
             scale="Difference", tol=.1e-10)
ciBinomial(x1, x2, n1, n2, alpha=.05, adj=0, scale="Difference")
simBinomial(p1, p2, n1, n2, delta0=0, nsim=10000, chisq=0, adj=0,
            scale="Difference")
```

Arguments

For `simBinomial()` and `ciBinomial()` all arguments must have length 1.

For `testBinomial()`, `x2, x2, n1, n2, delta0, chisq,` and `adj` may be vectors.

For `nBinomial()`, `p1, p2, beta, delta0` and `ratio` may be vectors.

For `nBinomial()` or `testBinomial()`, when one or more arguments is a vector, the routines return a vector of sample sizes and powers, respectively. Where vector arguments are allowed, there may be a mix of scalar and vector arguments. All arguments specified using vectors must have the same length.

`p1`	event rate in group 1 under the alternative hypothesis
`p2`	event rate in group 2 under the alternative hypothesis
`alpha`	type I error; see `sided` below to distinguish between 1- and 2-sided tests
`beta`	type II error
`delta0`	A value of 0 (the default) always represents no difference between treatment groups under the null hypothesis. `delta0` is interpreted differently depending on the value of the parameter `scale`. If `scale="Difference"` (the default), `delta0` is the difference in event rates under the null hypothesis (p10 - p20). If `scale="RR"`, `delta0` is the logarithm of the relative risk of event rates (p10 / p20) under the null hypothesis. If `scale="LNOR"`, `delta0` is the difference in natural logarithm of the odds-ratio under the null hypothesis `log(p10 / (1 - p10)) - log(p20 / (1 - p20))`.
`ratio`	sample size ratio for group 2 divided by group 1
`sided`	2 for 2-sided test, 1 for 1-sided test
`outtype`	`nBinomial` only; (default) returns total sample size; 2 returns sample size for each group (`n1, n2`); 3 and `delta0=0` returns a list with total sample size (`n`), sample size in each group (`n1, n2`), null and alternate hypothesis variance (`sigma0, sigma1`), input event rates (`p1, p2`) and null hypothesis event rates (`p10, p20`).
`x1`	Number of "successes" in the control group
`x2`	Number of "successes" in the experimental group
`n1`	Number of observations in the control group
`n2`	Number of observations in the experimental group
`chisq`	An indicator of whether or not a chi-square (as opposed to Z) statistic is to be computed. If `delta=0` (default), the difference in event rates divided by its standard error under the null hypothesis is used. Otherwise, a Miettinen and Nurminen chi-square statistic for a 2 x 2 table is used.
`adj`	With `adj=1`, the standard variance with a continuity correction is used for a Miettinen and Nurminen test statistic This includes a factor of $n/(n-1)$ where n is the total sample size. If `adj` is not 1, this factor is not applied. The default is `adj=0` since nominal Type I error is generally conservative with `adj=1` (Gordon and Watson, 1996).
`scale`	"Difference", "RR", "OR"; see the `scale` parameter documentation above and Details. This is a scalar argument.
`nsim`	The number of simulations to be performed in `simBinomial()`
`tol`	Default should probably be used; this is used to deal with a rounding issue in interim calculations

Details

Testing is 2-sided when a Chi-square statistic is used and 1-sided when a Z-statistic is used. Thus, these 2 options will produce substantially different results, in general. For non-inferiority, 1-sided testing is appropriate.

You may wish to round sample sizes up using `ceiling()`.

Farrington and Manning (1990) begin with event rates `p1` and `p2` under the alternative hypothesis and a difference between these rates under the null hypothesis, `delta0`. From these values, actual rates under the null hypothesis are computed, which are labeled `p10` and `p20` when `outtype=3`. The rates `p1` and `p2` are used to compute a variance for a Z-test comparing rates under the alternative hypothesis, while `p10` and `p20` are used under the null hypothesis.

Sample size with `scale="Difference"` produces an error if `p1-p2=delta0`. Normally, the alternative hypothesis under consideration would be `p1-p2-delta0`> 0. However, the alternative can have `p1-p2-delta0`< 0.

Value

`testBinomial()` and `simBinomial()` each return a vector of either Chi-square or Z test statistics. These may be compared to an appropriate cutoff point (e.g., `qnorm(.975)` for normal or `qchisq(.95,1)` for chi-square).

With the default `outtype=2`, `nBinomial()` returns a list containing two vectors `n1` and `n2` containing sample sizes for groups 1 and 2, respectively. With `outtype=1`, a vector of total sample sizes is returned. With `outtype=3`, `nBinomial()` returns a list as follows:

`n`	A vector with total samples size required for each event rate comparison specified
`n1`	A vector of sample sizes for group 1 for each event rate comparison specified
`n2`	A vector of sample sizes for group 2 for each event rate comparison specified
`sigma0`	A vector containing the variance of the treatment effect difference under the null hypothesis
`sigma1`	A vector containing the variance of the treatment effect difference under the alternative hypothesis
`p1`	As input
`p2`	As input
`pbar`	Returned only for superiority testing (\`delta0=0`), the weighted average of `p1` and `p2` using weights `n1` and `n2`
`p10`	group 1 event rate used for null hypothesis
`p20`	group 2 event rate used for null hypothesis

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Farrington, CP and Manning, G (1990), Test statistics and sample size formulae for comparative binomial trials with null hypothesis of non-zero risk difference or non-unity relative risk. *Statistics in Medicine*; 9: 1447-1454.

Fleiss, JL, Tytun, A and Ury (1980), A simple approximation for calculating sample sizes for comparing independent proportions. *Biometrics*;36:343-346.

Gordon, I and Watson R (1985), The myth of continuity-corrected sample size formulae. *Biometrics*; 52: 71-76.

Miettinin, O and Nurminen, M (1980), Comparative analysis of two rates. *Statistics in Medicine*; 4 : 213-226.

Examples

```
# Compute z-test test statistic comparing 39/500 to 13/500
# use continuity correction in variance
x <- testBinomial(x1=39, x2=13, n1=500, n2=500, adj=1)
x
pnorm(x, lower.tail=FALSE)

# Compute with unadjusted variance
x0 <- testBinomial(x1=39, x2=23, n1=500, n2=500)
x0
pnorm(x0, lower.tail=FALSE)

# Perform 10k simulations to test validity of the above
# asymptotic p-values
# (you may want to perform more to reduce standard error of estimate)
sum(as.real(x0) <=
    simBinomial(p1=.078, p2=.078, n1=500, n2=500, nsim=10000)) / 10000
sum(as.real(x0) <=
    simBinomial(p1=.052, p2=.052, n1=500, n2=500, nsim=10000)) / 10000

# Perform a non-inferiority test to see if p2=400 / 500 is within 5
# p1=410 / 500 use a z-statistic with unadjusted variance
x <- testBinomial(x1=410, x2=400, n1=500, n2=500, delta0= -.05)
x
pnorm(x, lower.tail=FALSE)

# since chi-square tests equivalence (a 2-sided test) rather than
# non-inferiority (a 1-sided test),
# the result is quite different
pchisq(testBinomial(x1=410, x2=400, n1=500, n2=500, delta0= -.05,
                    chisq=1, adj=1), 1, lower.tail=FALSE)

# now simulate the z-statistic witthout continuity corrected variance
sum(qnorm(.975) <=
    simBinomial(p1=.8, p2=.8, n1=500, n2=500, nsim=100000)) / 100000

# compute a sample size to show non-inferiority
# with 5% margin, 90% power
nBinomial(p1=.2, p2=.2, delta0=.05, alpha=.025, sided=1, beta=.1)

# assuming a slight advantage in the experimental group lowers
# sample size requirement
nBinomial(p1=.2, p2=.19, delta0=.05, alpha=.025, sided=1, beta=.1)

# compute a sample size for comparing 15% vs 10% event rates
# with 1 to 2 randomization
nBinomial(p1=.15, p2=.1, beta=.2, ratio=2, alpha=.05)
```

```
# now look at total sample size using 1-1 randomization
nBinomial(p1=.15, p2=.1, beta=.2, alpha=.05)

# look at power plot under different control event rate and
# relative risk reductions
p1 <- seq(.075, .2, .000625)
p2 <- p1 * 2 / 3
y1 <- nBinomial(p1, p2, beta=.2, outtype=1, alpha=.025, sided=1)
p2 <- p1 * .75
y2 <- nBinomial(p1, p2, beta=.2, outtype=1, alpha=.025, sided=1)
p2 <- p1 * .6
y3 <- nBinomial(p1, p2, beta=.2, outtype=1, alpha=.025, sided=1)
p2 <- p1 * .5
y4 <- nBinomial(p1, p2, beta=.2, outtype=1, alpha=.025, sided=1)
plot(p1, y1, type="l", ylab="Sample size",
     xlab="Control group event rate", ylim=c(0, 6000), lwd=2)
title(main="Binomial sample size computation for 80 pct power")
lines(p1, y2, lty=2, lwd=2)
lines(p1, y3, lty=3, lwd=2)
lines(p1, y4, lty=4, lwd=2)
legend(x=c(.15, .2),y=c(4500, 6000),lty=c(2, 1, 3, 4), lwd=2,
       legend=c("25 pct reduction", "33 pct reduction",
                "40 pct reduction", "50 pct reduction"))
```

`nSurvival` *3.3: Time-to-event sample size calculation (Lachin-Foulkes)*

Description

`nSurvival()` is used to calculate the sample size for a clinical trial with a time-to-event endpoint. The Lachin and Foulkes (1986) method is used.

Usage

```
nSurvival(lambda.0, lambda.1, Ts, Tr, eta = 0, rand.ratio = 1,
      alpha = 0.05, beta = 0.10, sided = 2, approx = FALSE,
      type = c("rr", "rd"), entry = c("unif", "expo"), gamma = NA)
```

Arguments

`lambda.0, lambda.1`	event hazard rate for placebo and treatment group respectively.
`eta`	equal dropout hazard rate for both groups.
`rand.ratio`	randomization ratio between placebo and treatment group. Default is balanced design, i.e., randomization ratio is 1.
`Ts`	maximum study duration.
`Tr`	accrual (recruitment) duration.
`alpha`	type I error rate. Default is 0.05 since 2-sided testing is default.
`beta`	type II error rate. Default is 0.10 (90% power).

`sided`	one or two-sided test? Default is two-sided test.
`approx`	logical. If `TRUE`, the approximation sample size formula for risk difference is used.
`type`	type of sample size calculation: risk ratio ("rr") or risk difference ("rd").
`entry`	patient entry type: uniform entry (`"unif"`) or exponential entry (`"expo"`).
`gamma`	rate parameter for exponential entry. `NA` if entry type is `"unif"` (uniform). A non-zero value is supplied if entry type is `"expo"` (exponential).

Details

`nSurvival` produces the number of subjects and events for a set of pre-specified trial parameters, such as accrual duration and follow-up period. The calculation is based on Lachin and Foulkes method and can be used for risk ratio or risk difference. The function also consider non-uniform entry as well as uniform entry.

If the logical approx is `TRUE`, the variance under alternative hypothesis is used to replace the variance under null hypothesis.

For non-uniform entry. a non-zero value of gamma for exponential entry must be supplied. For positive gamma, the entry distribution is convex, whereas for negative gamma, the entry distribution is concave.

Value

`nSurvival` produces a list with the following component returned:

`Method`	As input.
`Entry`	As input.
`Sample.size`	Number of subjects.
`Num.events`	Number of events.
`Hazard.p, Hazard.t`	hazard rate for placebo and treatment group. As input.
`Dropout`	as input.
`Frac.p, Frac.t`	randomization proportion for placebo and treatment. As input.
`Gamma`	as input.
`Alpha`	as input.
`Beta`	as input.
`Sided`	as input.
`Study.dura`	Study duration.
`Accrual`	Accrual period.

Author(s)

Shanhong Guan ⟨shanhong_guan@merck.com⟩

References

Lachin JM and Foulkes MA (1986), Evaluation of Sample Size and Power for Analyses of Survival with Allowance for Nonuniform Patient Entry, Losses to Follow-Up, Noncompliance, and Stratification. *Biometrics*, 42, 507-519.

Examples

```
# consider a trial with
# 2 year maximum follow-up
# 6 month uniform enrollment
# Treatment/placebo hazards = 0.1/0.2 per 1 person-year
# drop out hazard 0.1 per 1 person-year
# alpha = 0.05 (two-sided)
# power = 0.9 (default beta=.1)

ss <- nSurvival(lambda.0=.2 , lambda.1=.1, eta = .1, Ts = 2, Tr = .5,
                sided=1, alpha=.025)

#  symmetric, 2-sided design with O'Brien-Fleming-like boundaries
#  sample size is computed based on a fixed design requiring n=100
        x<-gsDesign(k = 5, test.type = 2)
        x
# boundary plot
        plot(x)
# power plot
        plot(x, plottype = 2)
# total sample size
   ceiling(x$n.I[x$k] * ss$Sample.size)
# number of events at analyses
   ceiling(ss$Num.events * x$n.I)
```

`Spending functions` *4.0: Spending function overview*

Description

Spending functions are used to set boundaries for group sequential designs. Using the spending function approach to design offers a natural way to provide interim testing boundaries when unplanned interim analyses are added or when the timing of an interim analysis changes. Many standard and investigational spending functions are provided in the gsDesign package. These offer a great deal of flexibility in setting up stopping boundaries for a design.

Usage

```
spendingFunction(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Defaults in calls to `gsDesign()` are `alpha=0.025` for one-sided Type I error specification and `alpha=0.1` for Type II error specification. However, this could be set to 1 if, for descriptive purposes, you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.

`param`	A single real value or a vector of real values specifying the spending function parameter(s); this must be appropriately matched to the spending function specified.

Details

Spending functions have three arguments as noted above and return an object of type `spendfn`. Normally a spending function will be passed to `gsDesign()` in the parameter `sfu` for the upper bound and `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence - only how to specify the parameter(s) for the spending function. The calling sequence is useful when the user wishes to plot a spending function as demonstrated below in examples. In addition to using supplied spending functions, a user can write code for a spending function. See examples.

Value

An object of type `spendfn`.

`name`	A character string with the name of the spending function.
`param`	any parameters used for the spending function.
`parname`	a character string or strings with the name(s) of the parameter(s) in `param`.
`sf`	the spending function specified.
`spend`	a vector of cumulative spending values corresponding to the input values in `t`.
`bound`	this is null when returned from the spending function, but is set in `gsDesign()` if the spending function is called from there. Contains z-values for bounds of a design.
`prob`	this is null when returned from the spending function, but is set in `gsDesign()` if the spending function is called from there. Contains probabilities of boundary crossing at `i`-th analysis for `j`-th theta value input to `gsDesign()` in `prob[i,j]`.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

`gsDesign`, `sfHSD`, `sfPower`, `sfLogistic`, `sfExponential`, Wang-Tsiatis Bounds, gsDesign package overview

Examples

```
# Example 1: simple example showing what mose users need to know

# design a 4-analysis trial using a Hwang-Shih-DeCani spending function
# for both lower and upper bounds
x <- gsDesign(k=4, sfu=sfHSD, sfupar=-2, sfl=sfHSD, sflpar=1)

# print the design
x

# plot the alpha- and beta-spending functions
plot(x, plottype=5)

# Example 2: advance example: writing a new spending function
# Most users may ignore this!

# implementation of 2-parameter version of
# beta distribution spending function
# assumes t and alpha are appropriately specified (does not check!)
sfbdist <- function(alpha,  t,  param)
{
   # check inputs
   checkVector(param, "numeric", c(0, Inf), c(FALSE, TRUE))
   if (length(param) !=2) stop(
   "b-dist example spending function parameter must be of length 2")

   # set spending using cumulative beta distribution and return
   x <- list(name="B-dist example", param=param, parname=c("a", "b"),
             sf=sfbdist, spend=alpha *
            pbeta(t, param[1], param[2]), bound=NULL, prob=NULL)

   class(x) <- "spendfn"

   x
}

# now try it out!
# plot some example beta (lower bound) spending functions using
# the beta distribution spending function
t <- 0:100/100
plot(t, sfbdist(1, t, c(2, 1))$spend, type="l",
    xlab="Proportion of information",
    ylab="Cumulative proportion of total spending",
    main="Beta distribution Spending Function Example")
lines(t, sfbdist(1, t, c(6, 4))$spend, lty=2)
lines(t, sfbdist(1, t, c(.5, .5))$spend, lty=3)
lines(t, sfbdist(1, t, c(.6, 2))$spend, lty=4)
legend(x=c(.65, 1), y=1 * c(0, .25), lty=1:4,
    legend=c("a=2, b=1","a=6, b=4","a=0.5, b=0.5","a=0.6, b=2"))
```

Description

The function `sfHSD` implements a Hwang-Shih-DeCani spending function. This is the default spending function for `gsDesign()`. Normally it will be passed to `gsDesign` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence. The calling sequence is useful, however, when the user wishes to plot a spending function as demonstrated below in examples.

Usage

```
sfHSD(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	A single real value specifying the gamma parameter for which Hwang-Shih-DeCani spending is to be computed; allowable range is [-40, 40]

Details

A Hwang-Shih-DeCani spending function takes the form

$$f(t;\alpha,\gamma) = \alpha(1-e^{-\gamma t})/(1-e^{-\gamma})$$

where γ is the value passed in `param`. A value of $\gamma = -4$ is used to approximate an O'Brien-Fleming design (see `sfExponential` for a better fit), while a value of $\gamma = 1$ approximates a Pocock design well.

Value

An object of type `spendfn`. See Spending function overview for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# design a 4-analysis trial using a Hwang-Shih-DeCani spending function
# for both lower and upper bounds
x <- gsDesign(k=4, sfu=sfHSD, sfupar=-2, sfl=sfHSD, sflpar=1)

# print the design
x

# since sfHSD is the default for both sfu and sfl,
# this could have been written as
x <- gsDesign(k=4, sfupar=-2, sflpar=1)

# print again
x

# plot the spending function using many points to obtain a smooth curve
# show default values of gamma to see how the spending function changes
# also show gamma=1 which is supposed to approximate a Pocock design
t <- 0:100/100
plot(t,  sfHSD(0.025, t, -4)$spend,
   xlab="Proportion of final sample size",
   ylab="Cumulative Type I error spending",
   main="Hwang-Shih-DeCani Spending Function Example", type="l")
lines(t, sfHSD(0.025, t, -2)$spend, lty=2)
lines(t, sfHSD(0.025, t, 1)$spend, lty=3)
legend(x=c(.0, .375), y=.025*c(.8, 1), lty=1:3,
    legend=c("gamma= -4", "gamma= -2", "gamma= 1"))
```

---

sfPower *4.2: Kim-DeMets (power) Spending Function*

Description

The function `sfPower()` implements a Kim-DeMets (power) spending function. This is a flexible, one-parameter spending function recommended by Jennison and Turnbull (2000). Normally it will be passed to `gsDesign()` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence. The calling sequence is useful, however, when the user wishes to plot a spending function as demonstrated below in examples.

Usage

```
sfPower(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	A single, positive value specifying the ρ parameter for which Kim-DeMets spending is to be computed; allowable range is (0,15]

Details

A Kim-DeMets spending function takes the form

$$f(t; \alpha, \rho) = \alpha t^{\rho}$$

where ρ is the value passed in `param`. See examples below for a range of values of ρ that may be of interest (`param=0.75` to `3` are documented there).

Value

An object of type `spendfn`. See Spending function overview for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# design a 4-analysis trial using a Kim-DeMets spending function
# for both lower and upper bounds
x <- gsDesign(k=4, sfu=sfPower, sfupar=3, sfl=sfPower, sflpar=1.5)

# print the design
x

# plot the spending function using many points to obtain a smooth curve
# show rho=3 for approximation to O'Brien-Fleming and rho=.75 for
```

```
# approximation to Pocock design.
# Also show rho=2 for an intermediate spending.
# Compare these to Hwang-Shih-DeCani spending with gamma=-4,  -2,  1
t <- 0:100/100
plot(t,  sfPower(0.025, t, 3)$spend, xlab="Proportion of sample size",
    ylab="Cumulative Type I error spending",
    main="Kim-DeMets (rho) versus Hwang-Shih-DeCani (gamma) Spending",
    type="l", cex.main=.9)
lines(t, sfPower(0.025, t, 2)$spend, lty=2)
lines(t, sfPower(0.025, t, 0.75)$spend, lty=3)
lines(t, sfHSD(0.025, t, 1)$spend, lty=3, col=2)
lines(t, sfHSD(0.025, t, -2)$spend, lty=2, col=2)
lines(t, sfHSD(0.025, t, -4)$spend, lty=1, col=2)
legend(x=c(.0, .375), y=.025*c(.65, 1), lty=1:3,
       legend=c("rho= 3", "rho= 2", "rho= 0.75"))
legend(x=c(.0, .357), y=.025*c(.65, .85), lty=1:3, bty="n", col=2,
       legend=c("gamma= -4", "gamma= -2", "gamma=1"))
```

`sfExponential` *4.3: Exponential Spending Function*

Description

The function `sfExponential` implements the exponential spending function (Anderson and Clark, 2009). Normally `sfExponential` will be passed to `gsDesign` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence. The calling sequence is useful, however, when the user wishes to plot a spending function as demonstrated below in examples.

Usage

```
sfExponential(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	A single positive value specifying the nu parameter for which the exponential spending is to be computed; allowable range is (0, 1.5].

Details

An exponential spending function is defined for any positive `nu` and $0 \leq t \leq 1$ as

$$f(t; \alpha, \nu) = \alpha(t) = \alpha^{t^{-\nu}}.$$

A value of `nu=0.8` approximates an O'Brien-Fleming spending function well.

The general class of spending functions this family is derived from requires a continuously increasing cumulative distribution function defined for $x > 0$ and is defined as

$$f(t;\alpha,\nu) = 1 - F\left(F^{-1}(1-\alpha)/t^{\nu}\right).$$

The exponential spending function can be derived by letting $F(x) = 1-\exp(-x)$, the exponential cumulative distribution function. This function was derived as a generalization of the Lan-DeMets (1983) spending function used to approximate an O'Brien-Fleming spending function (`sfLDOF()`),

$$f(t;\alpha) = 2 - 2\Phi\left(\Phi^{-1}(1-\alpha/2)/t^{1/2}\right).$$

Value

An object of type `spendfn`.

Note

The manual shows how to use `sfExponential()` to closely approximate an O'Brien-Fleming design. An example is given below. The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Lan, KKG and DeMets, DL (1983), Discrete sequential boundaries for clinical trials. *Biometrika*; 70:659-663.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# use 'best' exponential approximation for k=6 to O'Brien-Fleming design
# (see manual for details)
gsDesign(k=6, sfu=sfExponential, sfupar=0.7849295,
         test.type=2)$upper$bound

# show actual O'Brien-Fleming bound
gsDesign(k=6, sfu="OF", test.type=2)$upper$bound

# show Lan-DeMets approximation
# (not as close as sfExponential approximation)
gsDesign(k=6, sfu=sfLDOF, test.type=2)$upper$bound

# plot exponential spending function across a range of values of interest
t <- 0:100/100
```

```
plot(t, sfExponential(0.025, t, 0.8)$spend,
   xlab="Proportion of final sample size",
   ylab="Cumulative Type I error spending",
   main="Exponential Spending Function Example", type="l")
lines(t, sfExponential(0.025, t, 0.5)$spend, lty=2)
lines(t, sfExponential(0.025, t, 0.3)$spend, lty=3)
lines(t, sfExponential(0.025, t, 0.2)$spend, lty=4)
lines(t, sfExponential(0.025, t, 0.15)$spend, lty=5)
legend(x=c(.0, .3), y=.025*c(.7, 1), lty=1:5,
    legend=c("nu = 0.8", "nu = 0.5", "nu = 0.3", "nu = 0.2",
             "nu = 0.15"))
text(x=.59, y=.95*.025, labels="<--approximates O'Brien-Fleming")
```

`sfLDOF` *4.4: Lan-DeMets Spending function overview*

Description

Lan and DeMets (1983) first published the method of using spending functions to set boundaries for group sequential trials. In this publication they proposed two specific spending functions: one to approximate an O'Brien-Fleming design and the other to approximate a Pocock design. Both of these spending functions are available here, mainly for historical purposes. Neither requires a parameter.

Usage

```
sfLDOF(alpha, t, param)
sfLDPocock(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	This parameter is not used and need not be specified. It is here so that the calling sequence conforms the to the standard for spending functions used with `gsDesign()`.

Details

The Lan-DeMets (1983) spending function to approximate an O'Brien-Fleming bound is implemented in the function (`sfLDOF()`):

$$f(t;\alpha) = 2 - 2\Phi\left(\Phi^{-1}(1-\alpha/2)/t^{1/2}\right).$$

The Lan-DeMets (1983) spending function to approximate a Pocock design is implemented in the function `sfLDPocock()`:

$$f(t;\alpha) = ln(1 + (e-1)t).$$

As shown in examples below, other spending functions can be used to get as good or better approximations to Pocock and O'Brien-Fleming bounds. In particular, O'Brien-Fleming bounds can be closely approximated using `sfExponential`.

Value

An object of type `spendfn`. See spending functions for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Lan, KKG and DeMets, DL (1983), Discrete sequential boundaries for clinical trials. *Biometrika*;70: 659-663.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# 2-sided,  symmetric 6-analysis trial Pocock
# spending function approximation
gsDesign(k=6, sfu=sfLDPocock, test.type=2)$upper$bound

# show actual Pocock design
gsDesign(k=6, sfu="Pocock", test.type=2)$upper$bound

# approximate Pocock again using a standard
# Hwang-Shih-DeCani approximation
gsDesign(k=6, sfu=sfHSD, sfupar=1, test.type=2)$upper$bound

# use 'best' Hwang-Shih-DeCani approximation for Pocock,  k=6;
# see manual for details
gsDesign(k=6, sfu=sfHSD, sfupar=1.3354376, test.type=2)$upper$bound

# 2-sided, symmetric 6-analysis trial
# O'Brien-Fleming spending function approximation
gsDesign(k=6, sfu=sfLDOF, test.type=2)$upper$bound

# show actual O'Brien-Fleming bound
gsDesign(k=6, sfu="OF", test.type=2)$upper$bound
```

```
# approximate again using a standard Hwang-Shih-DeCani
# approximation to O'Brien-Fleming
x<-gsDesign(k=6, test.type=2)
x$upper$bound
x$upper$param

# use 'best' exponential approximation for k=6; see manual for details
gsDesign(k=6, sfu=sfExponential, sfupar=0.7849295,
         test.type=2)$upper$bound
```

`sfPoints` *4.5: Pointwise Spending Function*

Description

The function `sfPoints` implements a spending function with values specified for an arbitrary set of specified points. It is now recommended to use sfLinear rather than sfPoints. Normally `sfPoints` will be passed to `gsDesign` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence, just the points they wish to specify. If using `sfPoints()` in a design, it is recommended to specify how to interpolate between the specified points (e.g,, linear interpolation); also consider fitting smooth spending functions; see Spending function overview.

Usage

```
sfPoints(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. The last point should be 1. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	A vector of the same length as `t` specifying the cumulative proportion of spending to corresponding to each point in `t`.

Value

An object of type `spendfn`. See spending functions for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview, sfLogistic

Examples

```
# example to specify spending on a pointwise basis
x <- gsDesign(k=6, sfu=sfPoints, sfupar=c(.01, .05, .1, .25, .5, 1),
              test.type=2)
x

# get proportion of upper spending under null hypothesis
# at each analysis
y <- x$upper$prob[, 1] / .025

# change to cumulative proportion of spending
for(i in 2:length(y))
    y[i] <- y[i - 1] + y[i]

# this should correspond to input sfupar
round(y, 6)

# plot these cumulative spending points
plot(1:6/6, y, main="Pointwise spending function example",
    xlab="Proportion of final sample size",
    ylab="Cumulative proportion of spending",
    type="p")

# approximate this with a t-distribution spending function
# by fitting 3 points
tx <- 0:100/100
lines(tx, sfTDist(1, tx, c(c(1, 3, 5)/6, .01, .1, .5))$spend)
text(x=.6, y=.9, labels="Pointwise Spending Approximated by")
text(x=.6, y=.83, "t-Distribution Spending with 3-point interpolation")
```

`sfLinear` *4.6: Piecewise Linear Spending Function*

Description

The function `sfLinear()` allows specification of a piecewise linear spending function. This provides complete flexibility in setting spending at desired timepoints in a group sequential design. Normally this function will be passed to `gsDesign()` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. When

passed to `gsDesign()`, the value of `param` would be passed to `sfLinear` through the `gsDesign()` arguments `sfupar` for the upper bound and `sflpar` for the lower bound.

Usage

```
sfLinear(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size or information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size or information for which the spending function will be computed.
`param`	A vector with a positive, even length. All values must be strictly between 0 and 1. Letting `k <- length(param/2)`, the first `k` points in param specify increasing values strictly between 0 and 1 where the proportion of total spending is specified. The last k points in param specify increasing values strictly between 0 and 1 with the cumulative proportion of spending at the timepoints in the first part of the vector.

Value

An object of type `spendfn`. The cumulative spending returned in `sfLinear$spend` is 0 for `t=0` and `alpha` for `t>=1`. For `t` between specified points, linear interpolation is used to determine `sfLinear$spend`. See `Spending function overview` for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# set up alpha spending and beta spending to be piecewise linear
sfupar <- c(.2, .4, .05, .2)
sflpar <- c(.3, .5, .65, .5, .75, .9)
x <- gsDesign(sfu=sfLinear, sfl=sfLinear, sfupar=sfupar, sflpar=sflpar)
```

```
plot(x, plottype="sf")
x
```

---

sfLogistic *4.7: Two-parameter Spending Function Families*

Description

The functions `sfLogistic()`, `sfNormal()`, `sfExtremeValue()`, `sfExtremeValue2()`, `sfCauchy()`, and `sfBetaDist()` are all 2-parameter spending function families. These provide increased flexibility in some situations where the flexibility of a one-parameter spending function family is not sufficient. These functions all allow fitting of two points on a cumulative spending function curve; in this case, four parameters are specified indicating an x and a y coordinate for each of 2 points. Normally each of these functions will be passed to `gsDesign()` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence. The calling sequence is useful, however, when the user wishes to plot a spending function as demonstrated in the examples; note, however, that an automatic α- and β-spending function plot is also available.

Usage

```
sfLogistic(alpha, t, param)
sfNormal(alpha, t, param)
sfExtremeValue(alpha, t, param)
sfExtremeValue2(alpha, t, param)
sfCauchy(alpha, t, param)
sfBetaDist(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size or information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size or information for which the spending function will be computed.
`param`	In the two-parameter specification, `sfBetaDist()` requires 2 positive values, while `sfLogistic()`, `sfNormal()`, `sfExtremeValue()`, `sfExtremeValue2()` and `sfCauchy()` require the first parameter to be any real value and the second to be a positive value. The four parameter specification is `c(t1,t2,u1,u2)` where the objective is that `sf(t1)=alpha*u1` and `sf(t2)=alpha*u2`. In this parameterization, all four values must be between 0 and 1 and `t1 < t2`, `u1 < u2`.

Details

`sfBetaDist(alpha,t,param)` is simply `alpha` times the incomplete beta cumulative distribution function with parameters a and b passed in `param` evaluated at values passed in `t`.

The other spending functions take the form

$$f(t;\alpha,a,b) = \alpha F(a + bF^{-1}(t))$$

where $F()$ is a cumulative distribution function with values > 0 on the real line (logistic for `sfLogistic()`, normal for `sfNormal()`, extreme value for `sfExtremeValue()` and Cauchy for `sfCauchy()`) and $F^{-1}()$ is its inverse.

For the logistic spending function this simplifies to

$$f(t;\alpha,a,b)\alpha(1 - (1 + e^a(t/(1-t))^b)^{-1}).$$

For the extreme value distribution with

$$F(x) = \exp(-\exp(-x))$$

this simplifies to

$$f(t;\alpha,a,b) = \alpha \exp(-e^a(-\ln t)^b).$$

Since the extreme value distribution is not symmetric, there is also a version where the standard distribution is flipped about 0. This is reflected in `sfExtremeValue2()` where

$$F(x) = 1 - \exp(-\exp(x)).$$

Value

An object of type `spendfn`. See `Spending function overview` for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# design a 4-analysis trial using a Kim-DeMets spending function
# for both lower and upper bounds
x<-gsDesign(k=4, sfu=sfPower, sfupar=3, sfl=sfPower, sflpar=1.5)

# print the design
x

# plot the alpha- and beta-spending functions
plot(x, plottype=5)

# start by showing how to fit two points with sfLogistic
# plot the spending function using many points to obtain a smooth curve
# note that curve fits the points x=.1,  y=.01 and x=.4,  y=.1
# specified in the 3rd parameter of sfLogistic
t <- 0:100/100
plot(t,  sfLogistic(1, t, c(.1, .4, .01, .1))$spend,
    xlab="Proportion of final sample size",
    ylab="Cumulative Type I error spending",
    main="Logistic Spending Function Examples",
    type="l", cex.main=.9)
lines(t, sfLogistic(1, t, c(.01, .1, .1, .4))$spend, lty=2)

# now just give a=0 and b=1 as 3rd parameters for sfLogistic
lines(t, sfLogistic(1, t, c(0, 1))$spend, lty=3)

# try a couple with unconventional shapes again using
# the xy form in the 3rd parameter
lines(t, sfLogistic(1, t, c(.4, .6, .1, .7))$spend, lty=4)
lines(t, sfLogistic(1, t, c(.1, .7, .4, .6))$spend, lty=5)
legend(x=c(.0, .475), y=c(.76, 1.03), lty=1:5,
legend=c("Fit (.1, 01) and (.4, .1)", "Fit (.01, .1) and (.1, .4)",
    "a=0,  b=1", "Fit (.4, .1) and (.6, .7)",
     "Fit (.1, .4) and (.7, .6)"))

# set up a function to plot comparsons of all
# 2-parameter spending functions
plotsf <- function(alpha, t, param)
{
    plot(t, sfCauchy(alpha, t, param)$spend,
    xlab="Proportion of enrollment",
    ylab="Cumulative spending", type="l", lty=2)
    lines(t, sfExtremeValue(alpha, t, param)$spend, lty=5)
    lines(t, sfLogistic(alpha, t, param)$spend, lty=1)
    lines(t, sfNormal(alpha, t, param)$spend, lty=3)
    lines(t, sfExtremeValue2(alpha, t, param)$spend, lty=6, col=2)
    lines(t, sfBetaDist(alpha, t, param)$spend, lty=7, col=3)
    legend(x=c(.05, .475), y=.025*c(.55, .9),
             lty=c(1, 2, 3, 5, 6, 7),
             col=c(1, 1, 1, 1, 2, 3),
        legend=c("Logistic", "Cauchy", "Normal", "Extreme value",
        "Extreme value 2", "Beta distribution"))
}
# do comparison for a design with conservative early spending
# note that Cauchy spending function is quite different
```

```
# from the others
param <- c(.25, .5, .05, .1)
plotsf(.025, t, param)
```

`sfTDist` *4.8: t-distribution Spending Function*

Description

The function `sfTDist()` provides perhaps the maximum flexibility among spending functions provided in the `gsDesign` package. This function allows fitting of three points on a cumulative spending function curve; in this case, six parameters are specified indicating an x and a y coordinate for each of 3 points. Normally this function will be passed to `gsDesign()` in the parameter `sfu` for the upper bound or `sfl` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence. The calling sequence is useful, however, when the user wishes to plot a spending function as demonstrated below in examples.

Usage

```
sfTDist(alpha, t, param)
```

Arguments

`alpha`	Real value > 0 and no more than 1. Normally, `alpha=0.025` for one-sided Type I error specification or `alpha=0.1` for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
`t`	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size/information for which the spending function will be computed.
`param`	In the three-parameter specification, the first paramater (a) may be any real value, the second (b) any positive value, and the third parameter (df=degrees of freedom) any real value 1 or greater. When `gsDesign()` is called with a t-distribution spending function, this is the parameterization printed. The five parameter specification is `c(t1,t2,u1,u2,df)` where the objective is that the resulting cumulative proportion of spending at `t` represented by `sf(t)` satisfies `sf(t1)=alpha*u1`, `sf(t2)=alpha*u2`. The t-distribution used has `df` degrees of freedom. In this parameterization, all the first four values must be between 0 and 1 and `t1 < t2`, `u1 < u2`. The final parameter is any real value of 1 or more. This parameterization can fit any two points satisfying these requirements. The six parameter specification attempts to fit 3 points, but does not have flexibility to fit any three points. In this case, the specification for `param` is c(t1,t2,t3,u1,u2,u3) where the objective is that `sf(t1)=alpha*u1`, `sf(t2)=alpha*u2`, and `sf(t3)=alpha*u3`. See examples to see what happens when points are specified that cannot be fit.

Details

The t-distribution spending function takes the form

$$f(t;\alpha) = \alpha F(a + bF^{-1}(t))$$

where $F()$ is a cumulative t-distribution function with `df` degrees of freedom and $F^{-1}()$ is its inverse.

Value

An object of type `spendfn`. See spending functions for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/gsDesignManual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

See Also

Spending function overview, `gsDesign`, gsDesign package overview

Examples

```
# 3-parameter specification: a,  b,  df
sfTDist(1, 1:5/6, c(-1, 1.5, 4))$spend

# 5-parameter specification fits 2 points,  in this case
# the 1st 2 interims are at 25% and 50% of observations with
# cumulative error spending of 10% and 20%, respectively
# final parameter is df
sfTDist(1, 1:3/4, c(.25, .5, .1, .2, 4))$spend

# 6-parameter specification fits 3 points
# Interims are at 25%. 50% and 75% of observations
# with cumulative spending of 10%, 20% and 50%, respectively
# Note: not all 3 point combinations can be fit
sfTDist(1, 1:3/4, c(.25, .5, .75, .1, .2, .5))$spend

# Example of error message when the 3-points specified
# in the 6-parameter version cannot be fit
try(sfTDist(1, 1:3/4, c(.25, .5, .75, .1, .2, .3))$errmsg)

# sfCauchy (sfTDist with 1 df) and sfNormal (sfTDist with infinite df)
# show the limits of what sfTdist can fit
# for the third point are u3 from 0.344 to 0.6 when t3=0.75
sfNormal(1, 1:3/4, c(.25, .5, .1, .2))$spend[3]
```

```
sfCauchy(1, 1:3/4, c(.25, .5, .1, .2))$spend[3]

# plot a few t-distribution spending functions fitting
# t=0.25, .5 and u=0.1, 0.2
# to demonstrate the range of flexibility
t <- 0:100/100
plot(t, sfTDist(0.025, t, c(.25, .5, .1, .2, 1))$spend,
    xlab="Proportion of final sample size",
    ylab="Cumulative Type I error spending",
    main="t-Distribution Spending Function Examples", type="l")
lines(t, sfTDist(0.025, t, c(.25, .5, .1, .2, 1.5))$spend, lty=2)
lines(t, sfTDist(0.025, t, c(.25, .5, .1, .2, 3))$spend, lty=3)
lines(t, sfTDist(0.025, t, c(.25, .5, .1, .2, 10))$spend, lty=4)
lines(t, sfTDist(0.025, t, c(.25, .5, .1, .2, 100))$spend, lty=5)
legend(x=c(.0, .3), y=.025*c(.7, 1), lty=1:5,
    legend=c("df = 1", "df = 1.5", "df = 3", "df = 10", "df = 100"))
```

Description

Utility functions to verify an objects's properties including whether it is a scalar or vector, the class, the length, and (if numeric) whether the range of values is on a specified interval. Additionally, the `checkLengths` function can be used to ensure that all the supplied inputs have equal lengths.

Usage

```
isInteger(x)
checkScalar(x, isType = "numeric", ...)
checkVector(x, isType = "numeric", ..., length=NULL)
checkRange(x, interval = 0:1, inclusion = c(TRUE, TRUE),
   varname = deparse(substitute(x)), tol=0)
checkLengths(..., allowSingle=FALSE)
```

Arguments

`x`	any object.
`isType`	character string defining the class that the input object is expected to be.
`length`	integer specifying the expected length of the object in the case it is a vector. If `length=NULL`, the default, then no length check is performed.
`interval`	two-element numeric vector defining the interval over which the input object is expected to be contained. Use the `inclusion` argument to define the boundary behavior.
`inclusion`	two-element logical vector defining the boundary behavior of the specified interval. A `TRUE` value denotes inclusion of the corresponding boundary. For example, if `interval=c(3,6)` and `inclusion=c(FALSE,TRUE)`, then all the values of the input object are verified to be on the interval (3,6].

varname	character string defining the name of the input variable as sent into the function by the caller. This is used primarily as a mechanism to specify the name of the variable being tested when checkRange is being called within a function.
tol	numeric scalar defining the tolerance to use in testing the intervals of the checkRange function.
...	For the checkScalar and checkVector functions, this input represents additional arguments sent directly to the checkRange function. For the checkLengths function, this input represents the arguments to check for equal lengths.
allowSingle	logical flag. If TRUE, arguments that are vectors comprised of a single element are not included in the comparative length test in the checkLengths function. Partial matching on the name of this argument is not performed so you must specify 'allowSingle' in its entirety in the call.

Details

isInteger is similar to is.integer except that isInteger(1) returns TRUE whereas is.integer(1) returns FALSE.

checkScalar is used to verify that the input object is a scalar as well as the other properties specified above.

checkVector is used to verify that the input object is an atomic vector as well as the other properties as defined above.

checkRange is used to check whether the numeric input object's values reside on the specified interval. If any of the values are outside the specified interval, a FALSE is returned.

checkLength is used to check whether all of the supplied inputs have equal lengths.

Examples

```
# check whether input is an integer
isInteger(1)
isInteger(1:5)
try(isInteger("abc")) # expect error

# check whether input is an integer scalar
checkScalar(3, "integer")

# check whether input is an integer scalar that resides
# on the interval on [3, 6]. Then test for interval (3, 6].
checkScalar(3, "integer", c(3,6))
try(checkScalar(3, "integer", c(3,6), c(FALSE, TRUE))) # expect error

# check whether the input is an atomic vector of class numeric,
# of length 3, and whose value all reside on the interval [1, 10)
x <- c(3, pi, exp(1))
checkVector(x, "numeric", c(1, 10), c(TRUE, FALSE), length=3)

# do the same but change the expected length; expect error
try(checkVector(x, "numeric", c(1, 10), c(TRUE, FALSE), length=2))

# create faux function to check input variable
foo <- function(moo) checkVector(moo, "character")
```

```
foo(letters)
try(foo(1:5)) # expect error with function and argument name in message

# check for equal lengths of various inputs
checkLengths(1:2, 2:3, 3:4)
try(checkLengths(1,2,3,4:5)) # expect error

# check for equal length inputs but ignore single element vectors
checkLengths(1,2,3,4:5,7:8, allowSingle=TRUE)
```

B Acknowledgements

I would first like to thank Dan (Jennifer) Sun and Zhongxin (John) Zhang. Jennifer worked as a summer intern with John and me in the summer of 2006. Her project was to take some C programs I had written to derive group sequential designs and build an R package around them. Together, she and I wrote the first version of the gsDesign package and, with John, the original manual. The package has been used to help design several trials at Merck and I would like to thank Cong Chen, Zhiping (Linda) Sun, Yang Song, Weile He, Santosh Sutradhar, Yanli Zhao, Jason Clark, Yevgen Tymofyeyev and Fang Liu for their interest in applying the methods from the package and/or for reviewing the manual. Shanhong Guan contributed the sample size function for survival endpoints. I apologize to those I have forgotten! Thanks also to Andy Liaw for improving performance of the error handling facilities noted below as well as showing me how to apply parallel computations for simulations. The latter should be useful in a future update of the manual demonstrating simulation of information-based design.

Near the end of 2008, gsDesign development received additional financial support as part of an effort to expand the use of adaptive design at Merck. Thanks to Jerry Schindler, Vikas Patel and Sumeet Bagga for their support of this effort. The support was used to strengthen gsDesign as an example of an open source effort to develop high quality software. REvolution Computing was contracted to work on updating the package by reviewing code and documentation as well as creating a testing package to be used in evaluation of the quality of the programming as well as for installation qualification. They ended up doing much more. Rich Calaway has done substantial editing of the documentation and recommended other revisions, Kellie Wills performed much of the stress testing, and Sue Ranney assembled the installation qualification document. Chris Wiggins was my first contact at REvolution; he and I share common views about open source R packages. Finally, I would like to thank William Constantine who was my primary (nearly daily) contact for this project. Bill completely updated the error handling in gsDesign, set up a tremendous amount of the installation qualification testing and substantially reformatted the R code as well as the text markup for the manual. I look forward to any future potential collaboration with REvolution as the experience was quite productive and pleasant.

Keaven Anderson

References

[1] Keaven M. Anderson. Optimal spending functions for asymmetric group sequential designs. *Biometrical Journal*, 49:337–345, 2007.

[2] Keaven M. Anderson and Jason B. Clark. Fitting spending functions. *Statistics in Medicine*, To appear.

[3] CAPTUREInvestigators. Randomised placebo-controlled trial of abciximab before and during coronary intervention in refractory unstable angina: the capture study. *Lancet*, 349:1429–1435, 1997.

[4] Inc. Cytel. *EAST 5*. Cytel, Inc., Cambridge, MA, 2007.

[5] Kenneth Fairbanks and Richard Madsen. P-values for tests using a repeated significance test design. *Biometrika*, 69:69–74, 1982.

[6] Conor P. Farrington and Godfrey Manning. Test statistics and sample size formulae for comparative binomial trials with null hypothesis of non-zero risk difference or non-unity relative risk. *Statistics in Medicine*, 9:1447–1454, 1990.

[7] Center for Drug Evaluation and Research. Guidance for industry: Diabetes mellitus evaluating cardiovascular risk in new antidiabetic therapies to treat type 2 diabetes. 2008.

[8] I. K. Hwang, W. J. Shih, and J. S. DeCani. Group sequential designs using a family of type 1 error probability spending functions. *Statistics in Medicine*, 9:1439–1445, 1990.

[9] Christopher Jennison and Bruce W. Turnbull. *Group Sequential Methods with Applications to Clinical Trials*. Chapman and Hall/CRC, Boca Raton, FL, 2000.

[10] K. Kim and D. L. DeMets. Design and analysis of group sequential tests based on type i error spending rate functions. *Biometrika*, 74:149–154, 1987.

[11] John M. Lachin and Mary A. Foulkes. Evaluation of sample size and power for analyses of survival with allowance for nonuniform patient entry, losses to follow-up, noncompliance, and stratification. *Biometrics*, 42:507–519, 1986.

[12] K. K. G. Lan and David L. DeMets. Discrete sequential boundaries for clinical trials. *Biometrika*, 70:659–663, 1983.

[13] Ollie Miettinen and Markku Nurminen. Comparative analysis of two rates. *Statistics in Medicine*, 4:213–226, 1985.

[14] H. H. Muller and H. Schafer. Adaptive group sequential designs for clinical trials: combining the advantages of adaptive and of classical group sequential approaches. *Biometrics*, 57:886–891, 2001.

[15] P. C. O'Brien and T. R. Fleming. A multiple testing procedure for clinical trials. *Biometrika*, 35:549–556, 1979.

[16] Sandro Pampallona and Anastasios A. Tsiatis. Group sequential trials for one-sided and two-sided hypothesis testing with provision for early stopping in favor of the null hypothesis. *Journal of Statistical Planning and Inference*, 42:19–35, 1994.

[17] Stuart J. Pocock. Group sequential methods in the design and analysis of clinical trials. *Biometrika*, 64:191–199, 1977.

[18] Michael A. Proschan, K. K. Gordon Lan, and Janet Turk Wittes. *Statistical Monitoring of Clinical Trials. A Unified Approach.* Springer, New York, NY, 2006.

[19] Thomas Sellke and David Siegmund. Sequential analysis of the proportional hazards model. *Biometrika*, 70:315–326, 1983.

[20] Eric V. Slud and L. J. Wei. Two-sample repeated significance tests based on the modified wilcoxon statistic. *Journal of the American Statistical Association*, 77:862–868, xyzzy.

[21] The_GUSTO_V_Investigators. Reperfusion therapy for acute myocardial infarction with fibrinolytic therapy or combination reduced fibrinolytic therapy and platelet glycoprotein iib/iiia inhibition: the gusto v randomised trial. *Lancet*, 357:1905–1914, 2001.

[22] Anastasio A. Tsiatis. Repeated significance testing for a general class of statistics used in censored survival analysis. *Journal of the American Statistical Association*, 77:855–861, 1982.

[23] S. K. Wang and A. A. Tsiatis. Approximately optimal one-parameter boundaries for group sequential trials. *Biometrics*, 43:193–199, 1987.

C Contacts

Keaven M. Anderson: keaven_anderson@merck.com