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

Keaven M. Anderson, Dan (Jennifer) Sun, Zhongxin (John) Zhang
Merck Research Laboratories

December 18, 2008

Abstract

The gsDesign package supports group sequential clinical trial design. While there is a strong focus on designs using $\alpha-$ and $\beta-$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 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.

Contents

1 Overview

Three R functions are supplied to provide basic computations related to designing 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 these routines 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.

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.

To get started with `gsDesign`, read Section 2, Basic Features, and then proceed to Section 4, Detailed Examples, to get a feel for how the routines work. To try the routines out, read Section 3, Installation and Online Help. These three sections along with online help allow you to develop a design quickly without reading the full specification given in succeeding sections. For those interested in the theory behind this package, Section 9, Statistical Methods, provides background.

Complete clean-up and review of the manual (e.g., adding the references) will occur in version 1.20. Generally, the manual should be up-to-date with package revisions.

2 Basic Features

There are several key design features common to `gsDesign()`, `gsProbability()`, and `gsCP()`:

1. Computations are based on asymptotic approximations as provided by Jennison and Turnbull [5].

2. Power plots and boundary plots are available, in addition to various printing formats for summarization.

In addition, the following apply to `gsDesign()`:

1. Rather than supporting a wide variety of endpoint or design types (e.g., normal, binomial, time to event), the `gsDesign()` routine allows input of the sample size for a fixed design with no interim analysis and adjusts the sample size appropriately for a group sequential design.

2. Two-sided symmetric and asymmetric designs are supported, as well as one-sided designs.

3. The spending function approach to group sequential design first published by Lan and DeMets [7] is implemented. Commonly used spending functions published by Hwang, Shih, and DeCani [4] and by Kim and DeMets [6] are provided. Other built-in spending functions are included. Two- and three-parameter spending functions are particularly flexible. There is also point-wise specification of spending available. Finally, specifications are given for users to write their own spending functions.

4. As an alternative to the spending function approach, the Wang and Tsiatis [9] family of boundaries is also available for symmetric or one-sided designs. This family includes O'Brien-Fleming and Pocock boundaries as members.

5. For asymmetric designs, lower bound spending functions may be used to specify lower boundary crossing probabilities under the alternative hypothesis (beta spending) or null hypothesis (recommended when number of analyses is large or when faster computing is required—e.g., for optimization).

6. Normally it is assumed that when a boundary is crossed at the time of an analysis, the clinical trial must stop without a positive finding. In this case, the boundary is referred to as binding. For asymmetric designs, a user option is available to ignore lower bounds when computing Type I error. Under this assumption the lower bound is referred to as non-binding. That is, the trial may continue rather than absolutely requiring stopping when the lower bound is crossed. This is a conservative design option sometimes requested by regulators to preserve Type I error when they assume a sponsor may choose to ignore an aggressive futility (lower) bound if it is crossed.

3 Installation and Online Help

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.zip). This file includes a copy of this manual in the file gsDesignManual.pdf. The source, available for other platforms (but only tested minimally!) is in the file gsDesign-2.0.tar.gz (may be updated to fix bugs in a file such as gsDesign-2.0.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. 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.

4 Detailed Examples

Below are six examples, each followed by output generated to demonstrate how to use the functions in the package:

- Example 1 shows the calculation of information ratios and boundaries based on default values for `gsDesign()`. This also demonstrates the difference in sample size when assuming binding versus non-binding (the default) lower bounds.

- Example 2 demonstrates two-sided testing and user-specified spending; commands demonstrating O'Brien-Fleming, Pocock and Wang-Tsiatis bounds are also shown.
- Example 3 demonstrates use of the logistic spending function. It also gives further comments on the other two-parameter spending functions as well as the three-parameter t-distribution spending function.
- Example 4 shows how to use `gsProbability()` to calculate boundary crossing probabilities.
- Example 5 demonstrates how to design a non-inferiority study.
- Example 6 demonstrates a non-inferiority study that also evaluates superiority.

Example 1: Default input and standard spending functions

For this example, we begin by noting some defaults for `gsDesign()`, and continue with the default call and its associated standard print and plot output. Next, we show the structure of information returned by `gsDesign()`. Since the defaults provide sample size ratios for a group sequential design compared to a design with no interim analysis, we demonstrate how to generate sample sizes for a binomial trial with a binomial endpoint. Finally, we demonstrate some standard spending functions and how to set their corresponding parameters.

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 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.
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.
7. 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.

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. Next, a boundary plot is generated using `plot(x)`. A power plot is generated with the statement `plot(x, plottype = 2)`. The solid lines in this plot are, in ascending order, the cumulative power of the design first and second interims and final analysis, respectively, for different values of θ/δ, where δ is the standardized treatment effect for which the trial is powered and θ is the true/underlying standardized treatment effect. The dashed lines, in descending order, are one minus the probability of crossing the lower boundary for the first and second interims, respectively.

```
> 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)
> plot(x, plottype=2)
```

Above we have seen standard output for `gsDesign()`. Now we look at how to access individual items of information about what is returned from `gsDesign()`. First, we use `summary(x)` to list the elements of `x`. To view an individual element of `x`, we provide the example `x$delta`. See Section 9, Statistical Methods, for an explanation of this value of `x$delta`. Other elements of `x` can be accessed in the same way. 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. The command summary `x$upper` shows what these variables are. The upper boundary is shown with the command `x$upper$bound`.

```
> summary(x)
           Length Class   Mode
k          1      -none-  numeric
test.type  1      -none-  numeric
alpha      1      -none-  numeric
beta       1      -none-  numeric
astar      1      -none-  numeric
delta      1      -none-  numeric
```

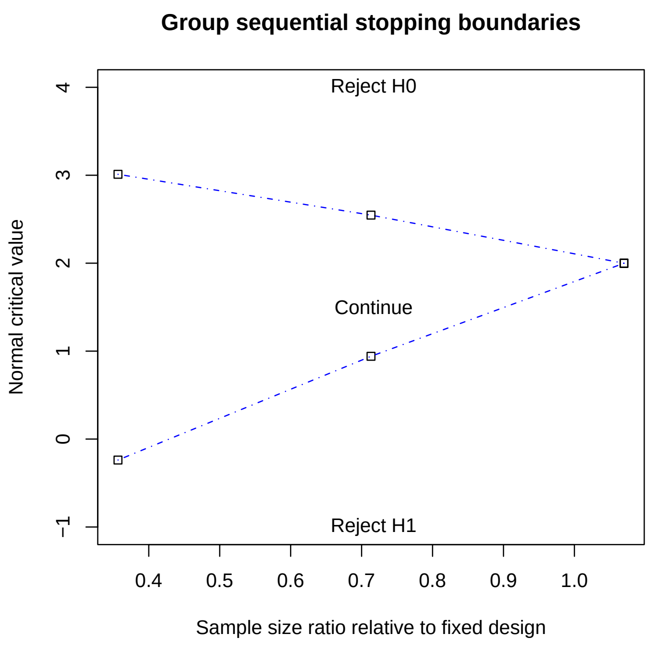

Figure 1: Default plot for gsDesign object from example 1

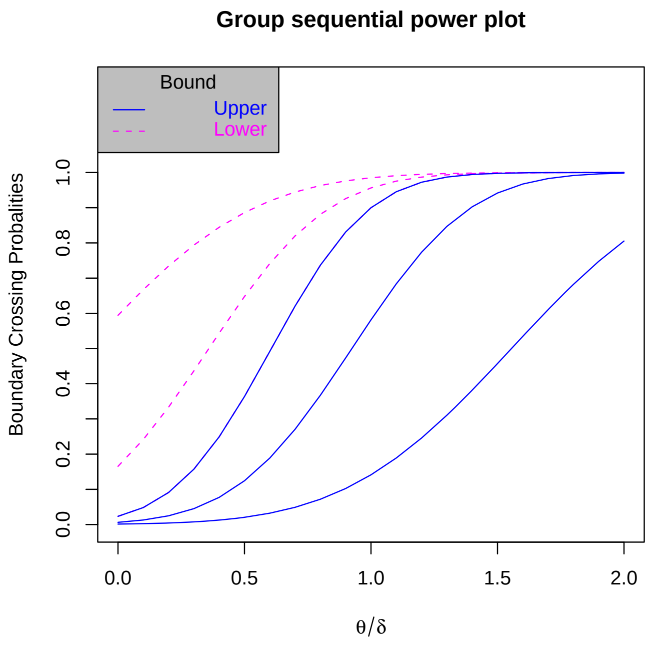

Figure 2: Power plot (plottype=2) for gsDesign object from example 1

```
n.fix      1      -none-  numeric
timing     3      -none-  numeric
tol        1      -none-  numeric
r          1      -none-  numeric
n.I        3      -none-  numeric
maxn.IPlan 1      -none-  numeric
errcode    1      -none-  numeric
errmsg     1      -none-  character
upper      9      spendfn list
lower      9      spendfn list
theta      2      -none-  numeric
falseposnb 3      -none-  numeric
en         2      -none-  numeric
> x$delta
[1] 3.241516
> summary(x$upper)
        Length Class  Mode
name    1      -none- character
param   1      -none- numeric
parname 1      -none- character
sf      1      -none- function
spend   3      -none- numeric
bound   3      -none- numeric
prob    6      -none- numeric
errcode 1      -none- numeric
errmsg  1      -none- character
> x$upper$bound
[1] 3.010739 2.546531 1.999226
```

Now suppose you wish to design a trial for a binomial outcome 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. A trial with no interim analysis has a sample size of 918 per arm or 1836 using `FarrMannSS()`. To get a sample size for the above design, we compute interim and final sample sizes (both arms combined) as follows:

```
> n.fix <- FarrMannSS(p1=.15, p2=.1, beta=.1, outtype=1)}
> n.fix
{[1] 1834.641
```

That is, we use `x$n.I` to adjust a fixed sample size design and obtain sample sizes for testing at the interim and final analyses. This method of calculating `x$n.I` is done automatically with the default input value of `n.fix = 1`. The following gets this specific trial design with the original call to `gsDesign()`, now with a calculated standardized effect size of $\delta = 0.0741$:

```
> gsDesign(n.fix=2*957)
```

If it were acceptable to use a logrank test for the above design with a fixed follow-up period per patient, nQuery returns a sample size of 645 per arm for a fixed design. In this case, the following sample sizes could be used at the interim and final analyses (the ceiling function is used to round up):

```
> ceiling(gsDesign(n.fix=2*645)$n.I)
[1] 461 921 1381
```

If, in addition, it were acceptable to assume the lower bound was binding, the sample size would be:

```
> ceiling(gsDesign(n.fix=2*645,test.type=3)$n.I)
[1] 451 902 1353
```

Before we proceed to example 2, we consider some simple alternatives to the standard spending function parameters. 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 use a Kim-DeMets (power) spending function for both lower and upper bounds with parameters 2 and 3, respectively. Then we compare bounds with the above design.

```
> 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
```

Example 2: 2-sided testing, including pointwise spending, O'Brien-Fleming, Pocock and Wang-Tsiatis designs

For this example we consider a two-sided test with user-specified spending and five analyses with unequal spacing. We again assume the binomial example with fixed n per arm of 957. Note the difference in labeling inside the plot due to the two-sided nature of testing. Note also that spending is printed as one-sided, and that only the upper spending function is needed/used. The cumulative spending at each analysis

```
> # Cumulative proportion of spending planned at each analysis
> p <- c(.05, .1, .15, .2, 1)
> # Cumulative spending intended at each analysis (for illustration)
> p * 0.025
```

```
[1] 0.00125 0.00250 0.00375 0.00500 0.02500
> # Incremental spending intended at each analysis
> # for comparison to spend column in output
> (p - c(0, p[0:4])) * 0.025
> x <- gsDesign(k=5, test.type=2, n.fix=1904, timing=c(.1,.25,.4,.6),
+ sfu=sfPoints,sfupar=p)
> x
Symmetric two-sided group sequential design with 90 % power and 2.5 % Type I Error.
Spending computations assume trial stops if a bound is crossed.


  Analysis    N    Z   Nominal p  Spend
         1  196 3.02    0.0013 0.0013
         2  488 2.99    0.0014 0.0013
         3  781 2.93    0.0017 0.0012
         4 1171 2.90    0.0019 0.0013
         5 1952 2.01    0.0222 0.0200
     Total                       0.0250
++ alpha spending: User-specified spending function with Points = 0.05 0.1 0.15 0.2 1

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      5 Total   E{N}
  0.0000 0.0013 0.0013 0.0013 0.0013 0.0200 0.025 1938.4
  0.0743 0.0235 0.0758 0.1218 0.1760 0.5029 0.900 1519.1

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2      3      4    5 Total
  0.0000 0.0013 0.0013 0.0013 0.0013 0.02 0.025
  0.0743 0.0000 0.0000 0.0000 0.0000 0.00 0.000
> plot(x)
```

O'Brien-Fleming, Pocock, or Wang-Tsiatis are normally used with equally-spaced analyses. O'Brien-Fleming, Pocock, or Wang-Tsiatis (parameter of 0.4) bounds for equally space analyses are generated as follows:

```
> xOF <- gsDesign(k=5, test.type=2, n.fix=1904, sfu="OF")
> xPk <- gsDesign(k=5, test.type=2, n.fix=1904, sfu="Pocock")
> xWT <- gsDesign(k=5, test.type=2, n.fix=1904, sfu="WT", sfupar=.4)
```

Once you have generated these designs, examine the upper bounds as in Example 1. Also, look at the spending by looking at, for example, `xOF$upper$spend`.

Example 3: Logistic spending function

Assume we would like the cumulative spending at 10% of enrollment to be 0.00125 (5% of total spending) and at 60% of enrollment to be 0.005 (20% of total spending) so that α-spending of 0.02

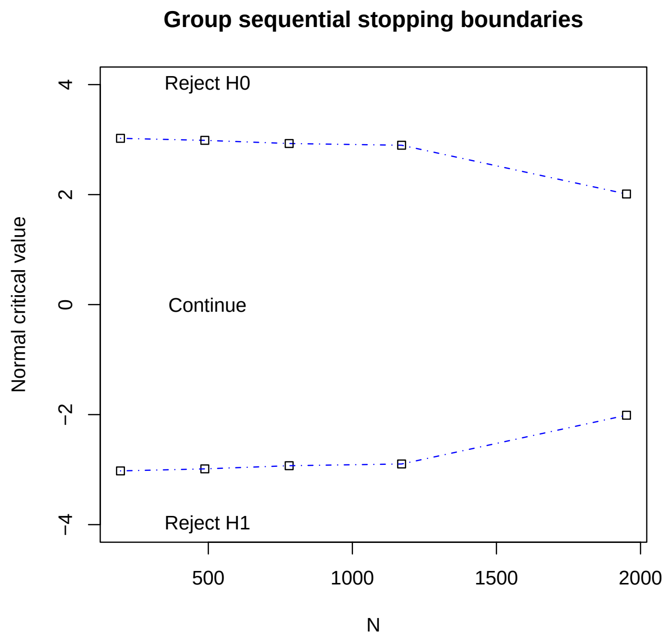

Figure 3: Boundary plot for example 2

is available for the final analysis; see `sfupar` in the following to find these numbers. This is the four-parameter specification of a logistic spending function (`sfLogistic()`, or the other two-parameter spending functions `sfNormal()` and `sfCauchy()`). In each case, the four-parameter specification is translated to the two essential parameters, but allows the specification to be done simply.

```
> gsDesign(k=5, timing=c(.1, .25, .4, .6), test.type=2, n.fix=1904,
+ sfu=sfLogistic, sfupar=c(.1, .6, .05, .2))
Symmetric two-sided group sequential design with 90 % power and 2.5 % Type I Error.
Spending computations assume trial stops if a bound is crossed.


  Analysis    N    Z   Nominal p  Spend
         1  195 3.02    0.0013 0.0013
         2  488 3.04    0.0012 0.0011
         3  780 2.99    0.0014 0.0010
         4 1170 2.83    0.0023 0.0017
         5 1949 2.01    0.0223 0.0200
     Total                       0.0250
++ alpha spending: Logistic spending function with a b = -1.629033 0.5986671

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      5 Total   E{N}
  0.0000 0.0013 0.0011 0.0010 0.0017 0.020 0.025 1936.1
  0.0743 0.0235 0.0686 0.1123 0.2077 0.488 0.900 1514.0

Lower boundary (futility or Type II Error)
          Analysis
   Theta      1      2     3      4    5 Total
  0.0000 0.0013 0.0011 0.001 0.0017 0.02 0.025
  0.0743 0.0000 0.0000 0.000 0.0000 0.00 0.000
```

The same output can be obtained using the two-parameter specification of the logistic spending function as follows (with values for `a` and `b` from above in `param`):

```
> y <- gsDesign(k=5, timing=c(.1, .25, .4, .6), test.type=2, n.fix=1904,
{+ sfu=sfogistic, sfupar=c(-1.629033, 0.5986671))
```

To verify the initial spend is 0.00125 (since it was rounded to 0.0013 above), we examine the appropriate element of `y` just computed:

```
> y$upper$spend[1]
[1] 0.00125
```

Example 4: gsProbability()

We reconsider Example 1 and obtain the properties for the design for a larger set of θ values than in the standard printout for `gsDesign()`. The standard plot design for a `gsProbability` object is the power plot shown in Example 1. The boundary plot is obtainable below using the command `plot(y, plottype=1)`.

```
> 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
```

Example 5: Non-inferiority testing

We consider a trial examining a new drug that is more convenient to administer than an approved control. There is no expectation of a substantially improved response with the new drug. While the new drug may be a little better or worse than control, there is some suggestion that the new drug may not be as efficacious as control. Rather than powering the trial to show non-inferiority when the new drug is slightly worse than control, the strategy is taken to stop the trial early for futility if there is a 'substantial' trend towards the new drug being inferior. The control drug has provided a (binomial) response rate of 67.7% in a past trial and regulators have agreed with a non-inferiority margin of 7%. Let the underlying event rate in the control and experimental groups be denoted by p_C and p_E, respectively. Let $\delta = 0.07$ represent the non-inferiority margin. There is no desire to stop the trial early to establish non-inferiority. That is, this is a one-sided testing problem for interim analyses. We let H_0: $p_C - p_A \leq 0$ and test against the alternative H_1: $p_C - p_A \geq \delta$., only stopping early if H_0 can be rejected. We must have 97.5% power to reject H_0 when H_1 is true ($\beta = 0.025$) and can have a 10% chance of rejecting H_0 when H_0 is true ($\alpha = 0.1$). In this case, an aggressive stopping boundary is desirable to stop the trial 40% of the way through enrollment if the experimental drug is, in fact, not as efficacious as control. The routine `FarrMannSS()` included with this package uses the method of Farrington and Manning [3] to compute the sample size for a two-arm binomial trial for superiority or non-inferiority; see the help file for documentation. As shown below, this requires 1966 patients for a trial with no interim analysis; both nQuery and PASS2005 also yield this result. Using this fixed sample size as input to `gsDesign()` yields a sample size of 2332 for the trial compared to 2333 from EAST 5.2. This design requires less than approximately a 4.5% difference in event rates at the interim analysis to continue and a final difference of no more than approximately 3.3% to achieve non-inferiority. These differences were carefully evaluated in choosing the appropriate value of `gamma` for the spending function.

```
> n.fix <- FarrMannSS(p1=.607, p2=.677, alpha=.1, beta=.025, sided=1, outtype=1)
> n.fix
[1] 1965.059
> gsDesign(k=2, alpha=.1, beta=.025, n.fix=n.fix, test.type=1, sfupar=3, timing=.4)
One-sided group sequential design with 97.5 % power and 10 % Type I Error.

  Analysis    N    Z   Nominal p  Spend
         1  933 1.45    0.0735 0.0735
         2 2332 1.68    0.0468 0.0265
     Total                      0.1000
++ alpha spending: Hwang-Shih-DeCani spending function with gamma = 3

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

Upper boundary (power or Type I Error)
          Analysis
   Theta      1      2 Total   E{N}
  0.0000 0.0735 0.0265 0.100 2228.7
  0.0731 0.7832 0.1918 0.975 1235.8
```

Example 6: Non-inferiority and superiority testing in the same trial

We consider a safety trial where a drug is given chronically to patients who are expected to have a 3.5% annual risk (exponential parameter $\lambda_0 = -\ln(1-0.035) = 0.035627$) of developing cardiovascular disease (CVD) and we wish to rule out an elevated risk that would be indicated by a hazard ratio of 1.2 ($\lambda_1 = 1.2\lambda_0 = 0.042753$). The desire is that if there is a hazard ratio of 1.2 that there is at most a 2.5% chance of demonstrating non-inferiority. On the other hand, if the true hazard ratio is 1 (no excess risk), we wish to have a 90% probability of showing 'no disadvantage' (that is, we wish to rule out $\lambda_1 \geq 1.2\lambda_0$). In hypothesis testing terms, the role of the null hypothesis (no difference) and alternative hypothesis (20% increased hazard ratio) have been reversed. We label the hypotheses as before, but the error levels are reversed to satisfy the above. We let the null hypothesis of no difference be denoted by H_0: $\log(\lambda_1/\lambda_0) = 0$ and the alternate hypothesis is denoted by H_1: $\log(\lambda_1/\lambda_0) = \log(1.2)$. To achieve the desired performance, Type I error is set to 10% and Type II error is set to 2.5%. Assume the trial is to be enrolled in a 2-year period and the dropout rate is 15% per year ($\lambda_D = -\ln(1-0.15) = 0.162519$). As seen below, fixed design with no interim analysis requires a sample size of 6,386 per treatment group to obtain 1,570 total events (under in 6 years H_1). Note that under the null hypothesis, the overall event rate would be lower and it would take longer to obtain the number of events required.

```
> # exponential control group failure rate of 3.5 percent per year
> lambda.0 <- -log(1-.035)
> # wish to rule out experimental group hazard ratio of 1.2
> lambda.1 <- 1.2*lambda.0
> # dropout rate of 15 percent per year
> eta <-  -log(1-.15)
> # fixed design sample size
> # recruitment time Tr=2 years, total study time Ts=6 years
> SSFix <-nSurvival(lambda.0=lambda.0, lambda.1=lambda.1, eta=eta, alpha=.1, beta=.025,
+ type="rr", Ts=6, Tr=2)
> # show sample size per group and total number of events required for fixed design
> ceiling(SSFix$Sample.size/2)
```

```
[1] 6386
> ceiling(SSFix$Num.events)
[1] 1570
```

We use the above sample size and number of events below to adjust this fixed design to a group sequential design and obtain the number of events required at each analysis. In addition, we consider the possibility that the experimental drug may actually reduce cardiovascular risk. For a fixed design, superiority testing may be performed following non-inferiority testing. For a group sequential trial, the situation is slightly more complex; this is not a scenario that can be dealt with in EAST 5.2. We take two approaches to this. First, we consider non-inferiority of control to treatment to be of no interest, which leads to an asymmetric design. Second, we consider the problem to be symmetric: inference on one arm relative to the other uses identical criteria; this will be deferred to example 7

Let H_0: $\theta = 0$ and H_1: $\theta \neq 0$ where θ indicates the underlying difference between two treatment groups. We wish to show that a new treatment is non-inferior compared to control. That is, for some $\delta > 0$, under H_{1A}: $\theta = \delta$ we wish to have 97.5% power to reject H_0: $\theta = 0$ (i.e., `beta=0.025` to yield a 2.5% chance of accepting non-inferiority when in fact the underlying effect for the experimental group is higher by δ). On the other hand, under H_0: $\theta = 0$ we are willing to have a 10% chance of rejecting H_0: $\theta = 0$ in favor of H_{1A}: $\theta = \delta$ (`alpha=0.10` to yield 90% power to show non-inferiority). We have a slightly asymmetric test for superiority in that we would like to reject H_0: θ=0 in favor of H_{1B}: $\theta = -\delta$ at the 2.5% (one-sided) level (`astar=0.025`) to control Type I error in that direction. Thus, the trial could stop early if θ is substantially different from 0 in either direction. A Hwang-Shih-DeCani spending function with `sfupar=sflpar=-4` is used for each bound. The bounds are asymmetric due to the different levels of the test in each direction. The appropriate confidence interval approach for this design is probably a stage-wise ordering approach (see Jennison & Turnbull [5], sections 8.4 and 8.5). This gives the tightest intervals at the end of the trial and does not result in conflicts between the confidence intervals and the testing approach just outlined.

See output below for a design with four equally spaced interims. With `n.fix=1264` we find that $\delta = 0.0818$. Interestingly, there is 90% power to cross the lower bound under H_{1B}: $\theta = -\delta$, so the trial is well-powered for superiority of the experimental treatment. The design requires an increase from 1570 events to 1595 in order to maintain the desired error rates with the given stopping rules. Note that `test.type=5` indicates that all bounds are binding and both upper and lower bound error spending is under the null hypothesis. The upper bound is a futility bound for showing non-inferiority. If it is never crossed, the trial establishes non-inferiority. The lower bound is the superiority bound.

```
> # show number of events required at interim and final analysis
> # of group sequential design
> x <- gsDesign(test.type=5, k=5, alpha=.1, beta=.025, astar=.025, sflpar=-4,
+               sfupar=-4, n.fix=SSFix$Num.events)
>
> # show sample size per group required using same inflation factor as
> # for number of events
> ceiling(x$n.I[5]/SSFix$Num.events*SSFix$Sample.size/2)}
[1] 6491
>
> # show power at + or - delta and 0
> y <- gsProbability(d=x, theta=c(-x$delta, 0, x$delta))
> y
Asymmetric two-sided group sequential design with 97.5 % power and 10 % Type I Error.
Spending computations assume trial stops if a bound is crossed.
```

```
                ----Lower bounds----  ----Upper bounds-----
  Analysis   N    Z   Nominal p Spend+  Z   Nominal p Spend++
         1  319 -3.25    0.0006 0.0006 2.84    0.0023  0.0023
         2  638 -2.99    0.0014 0.0013 2.52    0.0059  0.0051
         3  957 -2.69    0.0036 0.0028 2.17    0.0150  0.0113
         4 1276 -2.37    0.0088 0.0063 1.78    0.0376  0.0252
         5 1595 -2.03    0.0214 0.0140 1.33    0.0916  0.0561
     Total                      0.0250                 0.1000
+ lower bound spending (under H0): Hwang-Shih-DeCani spending function with gamma = -4
++ 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      4      5 Total   E{N}
  -0.0818 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 1150.8
   0.0000 0.0023 0.0051 0.0113 0.0252 0.0561 0.100 1566.1
   0.0818 0.0847 0.2516 0.3181 0.2255 0.0951 0.975  971.2

Lower boundary (futility or Type II Error)
          Analysis
    Theta      1      2      3      4      5 Total
  -0.0818 0.0366 0.1497 0.2632 0.2700 0.1785 0.898
   0.0000 0.0006 0.0013 0.0028 0.0063 0.0140 0.025
   0.0818 0.0000 0.0000 0.0000 0.0000 0.0000 0.000
> plot(y, plottype="xbar")
```

The plot below shows the cumulative probabilities of stopping at each analysis for different values of θ. The solid lines go from the probability of stopping at the first interim analysis to the highest line which shows the probability of an inferiority finding at any analysis. The dashed lines provide 1 minus the cumulative probability of stopping for superiority for different values of θ for each analysis. Note that the expected sample size required to come to a conclusion if the experimental regiment is truly inferior to control (H$_{1A}$: $\theta = \delta$) is 653 events compared to 1264 for a fixed design. In addition, there is only a 6.8% chance of continuing the trial to the final analysis. This is comprised of a 2.5% of continuing the trial to the final analysis and falsely establishing non-inferiority plus a 4.27% of continuing to the trial to the final analysis to establish inferiority. If experimental therapy is truly superior, the expected number of events required to demonstrate a difference is 950. If there is truly no difference the trial usually goes to the end (about 91% of the time; this is seen by the last result below) and finds no difference (87.5% of the time; this is the sum of the Type I error in both directions).

Evaluation of the cutoffs has not been performed as in Example 5. This exercise would be worthwhile to verify that the cutoffs of the design are appropriate.

```
> x <- gsDesign(test.type=5, k=5, alpha=.1, beta=.025, astar=.025, sflpar=-3,
+ sfupar=0, n.fix=1264)
> y <- gsProbability(d=x, theta=c(-x$delta, 0, x$delta))
> y
Asymmetric two-sided group sequential design with 97.5 % power and 10 % Type I Error.
```

```
Spending computations assume trial stops if a bound is crossed.

                  ----Lower bounds----  ----Upper bounds-----
  Analysis   N    Z   Nominal p Spend+  Z   Nominal p Spend++
         1  284 -3.07    0.0011 0.0011 2.05    0.0200    0.02
         2  567 -2.84    0.0022 0.0020 1.91    0.0278    0.02
         3  850 -2.60    0.0047 0.0036 1.79    0.0368    0.02
         4 1133 -2.34    0.0097 0.0065 1.68    0.0465    0.02
         5 1417 -2.06    0.0197 0.0119 1.58    0.0568    0.02
     Total                      0.0250                 0.1000
+ lower bound spending (under H0): Hwang-Shih-DeCani spending function with gamma = -3
++ alpha spending: Hwang-Shih-DeCani spending function with gamma = 0

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      5  Total   E{N}
  -0.0912 0.0002 0.000 0.0000 0.0000 0.0000 0.0002  950.0
   0.0000 0.0200 0.020 0.0200 0.0200 0.0200 0.1000 1352.8
   0.0912 0.3018 0.325 0.2048 0.1007 0.0427 0.9750  653.6

Lower boundary (futility or Type II Error)
          Analysis
    Theta      1      2      3      4      5  Total
  -0.0912 0.0625 0.1988 0.2796 0.2396 0.1401 0.9207
   0.0000 0.0011 0.0020 0.0036 0.0065 0.0119 0.0250
   0.0912 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
> plot(gsProbability(k=5, n.I=x$n.I, a=x$lower$bound, b=x$upper$bound,
+ theta=x$delta*(-25:25)/25))
> # compute probability of continuing to the end of the trial under H0
> 1-sum(y$upper$prob[1:4, 2] + y$lower$prob[1:4, 2])
[1] 0.9068707
```

5 Syntax

The two primary functions available in this package are:

- `gsDesign(argument1 = value1, ...)`
- `gsProbability(argument1 = value1, ...)`

All arguments are in lower case with the exception of `n.I` in `gsProbability()`. Although there are many arguments available in `gsDesign()`, many of these need not be specified as the defaults are adequate. Returned values from `gsDesign()` and `gsProbability()` are objects from newly defined classes named `gsDesign` and `gsProbability`. The `gsDesign` class is an extension of (inherits the characteristics of) the `gsProbability` class. These classes are described in Section 6, The `gsDesign` and `gsProbability` Object Types. The output functions `print()`, and `plot()` for the `gsDesign` and `gsProbability` classes are described further in Section 7, Formatted Output.

In the following, there is a single parameter θ for which we generally are trying to test a null hypothesis H_0: $\theta = 0$ against some alternative such as H1: $\theta \neq 0$. In general, the parameter θ

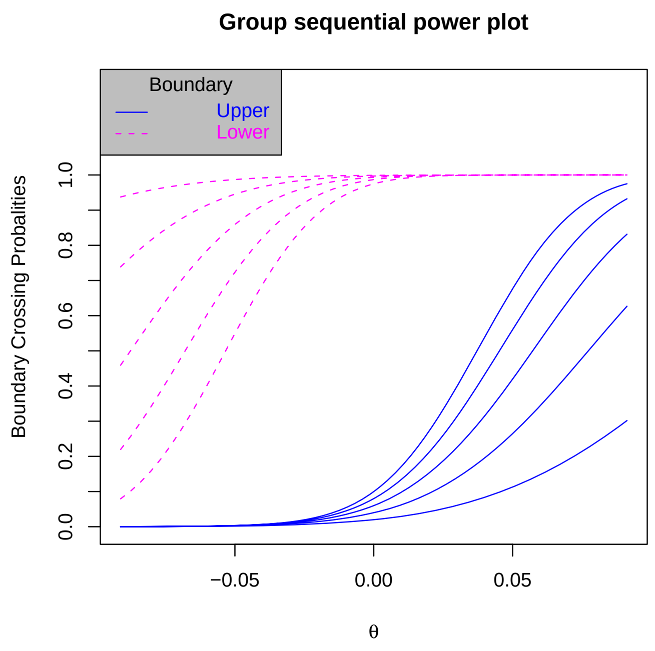

Figure 4: Power plot for non-inferiority design in example 6

is a standardized treatment difference and the statistic Z_n for testing after n observations have a distribution that is well approximated by a normal distribution with mean $\sqrt{n}\theta$ and variance 1. See Jennison and Turnbull [5] for extensive discussions of various types of endpoints for which the approximation is reasonable. For example, normal, binomial and time-to-event endpoints may be considered. Section 9, Statistical Methods, provides more detail.

As a supplement to the primary functions `gsDesign()` and `gsProbability()`, we also consider functions `gsCP()` and `gsBoundCP()` which compute the conditional probabilities of boundary crossing. Background for conditional power computations is also provided in Section 9, Statistical Methods.

Finally, some utility functions for working with two-arm binomial or survival trials are provided. For the binomial distribution there are functions for fixed (non-group-sequential) designs that compute sample size, perform statistical testing, and perform simulations. For survival, a sample size function is provided.

5.1 gsDesign() syntax

As noted above, `gsDesign()` 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. The most general form of the call to `gsDesign()` with default arguments is:

```
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).
```

The arguments are as follows:

- `k` = integer (`k`>1). Specifies the number of analyses, including interim and final. The default value is 3. **Note:** Use of very large `k` produces an error message. How large varies with the value of `test.type` specified. For example, with default arguments (`test.type=3`), specifying `k` > 23 produces the error message "False negative rates not achieved." For other values of `test.type`, the upper limit is larger.

- `test.type` = integer (1≤ `test.type` ≤ 6). In the following we denote by θ the parameter we are testing for. The null hypothesis will always be H_0: $\theta = 0$. The alternative hypothesis of interest varies with the value of `test.type`.

  =1: One-sided testing. Tests the null hypothesis H_0: $\theta = 0$ against the alternative hypothesis H_1: $\theta > 0$. The user specifies only upper boundary crossing probabilities under the null hypothesis. There is no lower boundary.

  =2: Symmetric, two-sided testing. Tests H_0: $\theta = 0$ against the alternative hypothesis H1: $\theta \neq 0$. Boundary crossing probabilities are specified under the null hypothesis. The user specifies only upper boundary crossing probabilities

  =3: Asymmetric two-sided testing with binding lower bound and beta spending. Asymmetric two-sided testing is also often referred to as two one-sided tests in that testing is done both for efficacy (attempts to reject H_0: $\theta = 0$ in favor of H_1: $\theta > 0$) and futility (attempts to reject H_1: $\theta =$ `delta` in favor of H_0: $\theta <$ `delta`). Upper and lower boundary crossing probabilities are asymmetric. Upper boundary crossing probabilities are specified under H_0: $\theta = 0$ ($\alpha-$spending). Lower boundary crossing probabilities are specified under H_1: $\theta =$ `delta` ($\beta-$spending). Computation of upper boundary crossing probabilities under the null hypothesis (Type I error) assumes the lower boundary is binding; that is, the trial *must* stop if the lower boundary is crossed. See Section 9, Statistical Methods, for details on boundary crossing probabilities for options 3 through 6.

  =4: Default. Asymmetric two-sided testing with non-binding lower bound and beta spending. Same as `test.type=3`, except that Type I error computation assumes lower boundary is non-binding; that is, the calculation assumes that the trial stops only if the upper boundary is crossed—it will continue if the lower boundary is crossed. Type II error computation assumes the trial will stop when either boundary is crossed. The effect of this is to raise the upper boundaries after the first interim analysis relative to `test.type=3`. See Section 9, Statistical Methods, for details.

  =5: Asymmetric two-sided testing with binding lower bound and lower bound spending specified under the null hypothesis. Same as `test.type=3`, except that lower boundary crossing probabilities are specified under H_0: $\theta = 0$. See Section 9, Statistical Methods, for details.

  =6: Asymmetric two-sided testing with non-binding lower bound and lower bound spending specified under the null hypothesis. Same as `test.type=4`, except that lower boundary crossing probabilities are specified under H_0: $\theta = 0$. See Section 9, Statistical Methods, for details.

- `alpha` = probability value (real; 0 < `alpha` < 1; for `test.type=2`, must have `alpha` < 0.5). Specifies the total Type I error summed across all analyses. For all design types (including symmetric, two-sided) this is the probability of crossing the upper boundary under the null hypothesis. The default value is 0.025. For symmetric, two-sided testing (`test.type=2`), this translates into 0.05 for the combined total probability of crossing a lower or upper bound at any analysis. See Section 9, Statistical Methods, for details.

- `beta` = probability value (real; 0 < `beta` < `1 - alpha`). Specifies the total Type II error summed across all analyses. The default value is 0.1, which corresponds to 90% power.

- `astar` = probability value (real; $0 \leq$ `astar` $\leq 1-$ `alpha`). Normally not specified. If `test.type=5` or `test.type=6`, `astar` specifies the probability of crossing a lower bound at all analyses combined. This is changed to $1-$`alpha` when the default value of 0 is used. Since this is the expected usage, `astar` is not normally specified by the user.

- `delta` = real value ($\delta > 0$). This is the standardized effect size used for alternative hypothesis for which the design is powered. If `delta` > 0, sample size is determined using this effect size (see Section 9, Statistical Methods, for details); If the default `delta = 0` is given, sample size is determined by `n.fix` as noted below. Only one of `delta` and `n.fix` need be specified by the user.

- `n.fix` = real value (`n.fix` > 0). If `delta`>0 then `n.fix` is ignored. For the default values (`delta = 0` and `n.fix = 1`) the returned values (R_1, ..., R_K) for sample sizes are actually sample size ratios compared to a fixed design with no interim analysis, where the denominator for all of these ratios is the sample size for a fixed design with no interim analysis based on input values of `alpha` and `beta`. If `delta = 0` and `n.fix` > 1, think of `n.fix` as the fixed sample size of a study without interim analysis for the given values of `alpha` and `beta`. `gsDesign()` then inflates this value to get a maximum sample size to give the specified Type II error/power for the specified group sequential design. We also denote `n.fix` as N_{fix} below. See Section 9, Statistical Methods, for details on how to calculate sample size.

- `timing` = 1 or `c(t`$_1$`, ..., t`$_{k-1}$`)` or `c(t`$_1$`, ..., t`$_k$`)` where $0 < t_1 < \ldots < t_k = 1$. Specifies the cumulative proportionate timing for analyses; for example, $t_2 = 0.4$ means the second interim includes the first 40% of planned observations. For equal spacing, `timing = 1` (default), the timing is determined by the number of interim looks `k`.

- `sfu` = spending function (default = `sfHSD`). The parameter sfu specifies the upper boundary using a spending function. For one-sided and symmetric two-sided testing (`test.type`=1, 2), `sfu` is the only spending function required to specify spending. The default value is `sfHSD`, which is a Hwang-Shih-DeCani spending function. See Section 8, Spending Functions, for details on the available options.

- `sfupar` = real value (default = -4) The parameter `sfupar` specifies any parameters needed for the spending function specified by `sfu`. The default of -4 when used with the default `sfu=sfHSD` provides a conservative, O'Brien-Fleming-like upper bound. For spending functions that do not require parameters this is ignored. Again, see Section 8, Spending Functions, for details.

- `sfl` = spending function (default = `sfHSD`). Specifies the spending function for lower boundary crossing probabilities. The parameter `sfl` is ignored for one-sided testing (`test.type = 1`) or symmetric two-sided testing (`test.type = 2`).

- `sflpar` = real value (default = -2). The parameter `sflpar` specifies any parameters needed for the spending function specified by `sfl`. The default value of -2 when used with the default `sfl=sfHSD` provides a somewhat conservative lower bound. For spending functions that do not require parameters this is ignored.

- `tol` = value. Specifies the stopping increment for iterative root-finding algorithms used to derive the user-specified design. The default value is 0.000001; must be >0 and ≤ 0.1. This should probably be ignored by the user, but is provided as a tuning parameter.

- `r` = positive integer up to 80 (default = 18). This is a parameter used to set the grid size for numerical integration computations; see Jennison and Turnbull [5], Chapter 19. This should probably be ignored by the user, but is provided as a tuning parameter.

- `n.I` = Used for re-setting bounds when timing of analyses changes from initial design; see Section 8.4, Resetting Timing of Analyses.
- `maxn.I` = Used for re-setting bounds when timing of analyses changes from initial design; see Section 8.4, Resetting Timing of Analyses.

5.2 gsProbability() syntax

`gsProbability()` computes boundary crossing probabilities and expected sample size of a design for arbitrary user-specified treatment effects, interim analysis times, and bounds. The value returned has class `gsProbability` as described in Section 6, The `gsDesign` and `gsProbability` Object Types. The generic call to `gsProbability()` is of the form:

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

The value of `theta` must always be specified. This function is designed to be called in one of two ways. First, using a returned value from a call to `gsDesign()` in the input parameter `d`. If `d` is non-null, the parameterization specified there determines the output rather than the values of `k`, `n.I`, `a`, `b`, and `r`. If `k` is not the default of 0, the user must specify the set of input variables `k`, `n.I`, `a`, `b`, and `r`. In the latter form, the arguments `n.I`, `a`, and `b`, must all be vectors with a common length `k` equaling the number of analyses, interim and final, in the design.

The arguments are as follows:

- `k` = non-negative integer (default = 0). The number of analyses, including interim and final; default value of 0. This *must* be 0 if the argument `d` is non-null.
- `theta` = vector of real values. This specifies parameter values (standardized effect sizes) for which boundary crossing probabilities and expected sample sizes are desired.
- `n.I` = vector of length `k` of real values $0 < \mathtt{I}_1 < \mathtt{I}_2 \ldots < \mathtt{I}_\mathtt{k}$. Specifies the statistical information $\mathtt{I}_1, \ldots, \mathtt{I}_\mathtt{k}$ (see Section 9, Statistical Methods) for each analysis.
- `a` = vector of length `k` of real values $(l_1, \ldots, l_k)$. Specifies lower boundary values.
- `b` = vector of length `k` of real values $(u_1, \ldots, u_k)$. Specifies upper boundary values. For $i <$`k` should have $l_i < u_i$. For $i =$`k`, should have $l_i \leq u_i$.
- `r` = positive integer up to 80 (default = 18). Same as for `gsDesign()`. This should probably be ignored by the user, but is provided as a tuning parameter.
- `d` = a returned value from a call to `gsDesign()`; values of `k`, `n.I`, `a`, `b`, and `r` are taken from `d` when the input value of `k` is 0. Note that if `d` is specified and `k = 0`, then the returned value is an object of type `gsDesign`; otherwise, the type returned is the simpler `gsProbability` class.

5.3 Conditional power: gsCP() and gsBoundCP() syntax

The `gsCP()` function 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. The value returned has class `gsProbability` which contains a design based on observations after interim `i` that is input and probabilities of boundary crossing for that design for the input values of `theta`. These boundary crossing probabilities are the conditional probabilities of crossing future bounds; see Section 9, Statistical Methods, and Section 4, Detailed Examples. The syntax for `gsCP()` takes the form:

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

where `theta`, and `r` are defined as for `gsProbability()`. In addition, we have

- `x` = a returned value from a call to `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` = a specified interim analysis; must be a positive integer less than the total number of analyses specified by `x$k`.
- `zi` = the interim test statistic value at analysis `i`; must be in the interval `x$lower$bound[i]` $\leq$ `zi` $\leq$ `x$upper$bound[i]`.
- `r` = positive integer up to 80 (default = 18). Same as for `gsDesign()`. This should probably be ignored by the user, but is provided as a tuning parameter.

The `gsBoundCP()` function 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. The returned value is a list of two vectors, `cplo` and `cphi`, which contain conditional power at each interim analysis based on an interim test statistic at the low and high boundaries, respectively. The syntax for `gsBoundCP()` takes the form:

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

- `x` = a returned value from a call to `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` = positive integer up to 80 (default = 18). Same as for `gsDesign()`. This should probably be ignored by the user, but is provided as a tuning parameter.

5.4 Binomial distribution: FarrMannSS(), MandNtest(), MandNsim() syntax

All of these routines are designed for two-arm binomial trials and use asymptotic approximations. They may be used for superiority or non-inferiority trials. `FarrMannSS()` computes sample size using the method of Farrington and Manning (1990). `MandNTest()` computes a Z- or Chi-square-statistic that compares two binomial event rates using the method of Miettinen and Nurminen (1980). `MandNsim()` performs simulations to estimate the power for a Miettinin and Nurminen (1980) test.

```
   FarrMannSS(p1, p2, fraction = 0.5, alpha = 0.05, power = 0.8, beta=0, delta0=0,
       ratio=0, sided=2, outtype=2)
   MandNTest(x1,x2,n1,n2,delta0=0,testtype="Chisq",adj=1)
   MandNsim(p1, p2, n1, n2, alpha = 0.05, delta0=0, nsim = 10000, testtype="Chisq",
adj=1)
```

- `p1` = event rate in group 1
- `p2` = event rate in group 2
- `fraction` = fraction of observations in group 1
- `alpha` = type I error; see sided below to distinguish between 1- and 2-sided tests
- `power` = the desired probability of detecting a difference
- `beta` = type II error; used instead of `power` when non-zero
- `delta0` = The absolute difference in event rates that is considered unacceptable.
- `ratio` = sample size ratio for group 2 divided by group 1; used instead of fraction when non-zero
- `sided` = 2 for 2-sided test, 1 for 1-sided test
- `outtype` = 2 (default) returns sample size for each group (`n1`, `n2`); otherwise returns `n1+n2`
- `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
- `testtype` = If `"Chisq"` (default), a Miettinen and Nurminen chi-square statistic for a 2×2 table (no continuity correction) is used. Otherwise, the difference in event rates divided by its standard error under the null hypothesis is used.
- `adj` = With `adj=1`, the standard variance for a Miettinen and Nurminen test statistic is used. 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. See details.
- `nsim` = The number of simulations to be performed in `MandNSim()`

`MandNTest()` and `MandNsim()` each return a vector of either Chi-square or Z test statistics. These may be compared to an appropriate cutoff point (for example, `qnorm(.975)` or `qchisq(.95,1)`).

With the default `outtype=2`, `FarrMannSS()` 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`, `FarrMannSS()` 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 treatment effect used for null hypothesis
- `p20`: group 2 treatment effect used for null hypothesis

6 The gsDesign and gsProbability Object Types

This package creates three object (storage) classes to store the structured information returned by `gsDesign()`, `gsProbability()` and spending function routines. Thus, a single variable name can be used to access, print or plot a set of returned values.

6.1 The spendfn Class

The first class, `spendfn`, is used to store information about either the upper or the lower bound of a design: the spending function, z-values for the stopping bounds, spending at each analysis and boundary crossing probabilities under parameter values of interest. The `spendfn` class contains the following members:

- `sf`: a spending function or character value. Normally this is a spending function, but if Wang-Tsiatis ("WT"), Pocock ("Pocock"), or O'Brien-Fleming ("OF") are specified (these are not spending function designs), then `sf` is a character string. This is useful if you later wish to plot the spending function for a larger set of values than specified in an original call. (**Note for writers of new spending functions:** this is not set in the spending function itself, but in `gsDesign()`)
- `name`: a character string specifying the spending function used.
- `param`: parameter value or values to fully specify the spending function using the spending function family specified in `sf`.
- `parname`: a text string or vector of text strings (matching the length of `param`) supplying the names of parameters.
- `spend`: a vector containing the amount of α- or β-spending at each analysis; this is determined in a call to `gsDesign()`.
- `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 a `k`×`n` matrix of probabilities of boundary crossing at `i`-th analysis for `j`-th `theta` value in `prob[i,j]`. Upon return from `gsDesign()`, `n`=2 and the values of theta considered are 0 and `delta`. More values of `theta` can be added by a call to `gsProbability()`. Columns correspond to the values specified by `theta`. All boundary crossing is assumed to be binding for this computation; that is, the trial must stop if a boundary is crossed.
- `errcode`: 0 if no error in spending function specification, > 0 otherwise.
- `errmsg`: a text string corresponding to `errcode`; for no error, this is "No error."

6.2 The gsProbability Class

The second object class is `gsProbability`. The members of this class on output from `gsProbability()` are: `k`, `n.I`, `lower`, `upper`, `r`, `theta`, `errcode`, and `errmsg`. Most of these are described in the input to `gsProbability()`. The exceptions are as follows:

- `upper`: on output, contains information on upper spending; this is a member of the `spendfn` class described above. Values of `upper` returning in a `gsProbability` class contain only the elements `bound` and `prob`.
- `lower`: on output, contains information on lower spending; this is a member of the `spendfn` class described above. Values of `lower` returning in a `gsProbability` class contain only the elements `bound` and `prob`.
- `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`.
- `errcode`: 0 if no error detected in call to `gsDesign()` or `gsProbability()`, > 0 otherwise.
- `errmsg`: a text string corresponding to `errcode`; for no error, this is "No errors detected."

6.3 The gsDesign Class

The third and final object class is `gsDesign`. This inherits the class `gsProbability` and, in addition, has the following members on output from `gsDesign()`: `test.type`, `alpha`, `beta`, `delta`, `n.fix`, `timing`, `tol`, `maxn.IPlan`. In addition, all elements of the class spendfn are returned in `upper` and `lower`. Most of these variables are as described in the input to `gsDesign()` or in the `spendfn` or `gsProbability` class description. The exceptions are as follows:

- `timing`: when this is input as 1, the output is transformed to a vector of equally spaced analyses as follows: (1, 2, ..., `k`)/`k`. Otherwise, this should be a vector of length `k`, with increasing values greater than 0 and the greatest value equal to 1. A value of `timing[2] = 0.4` is translated as the second interim analysis being performed after 40% of the total observations planned for the final analysis.
- `delta`: the standardized effect size; if this was input as 0, it is recomputed with the following formula:
$$\delta = \frac{\Phi^{-1}(1-\alpha) + \Phi^{-1}(1-\beta)}{\sqrt{N_{fix}}}$$
where δ=`delta`, N_{fix}=`n.fix`, α=`alpha`, and β=`beta`.
- `n.I`: this is the sample size or information required at each analysis, depending on how `gsDesign()` is called. In addition to the following descriptions, see Section 4, Detailed Examples.
  1. If `n.I` was input, the same value remains on output
  2. If input to `gsDesign()` was `n.fix` = 1 and `delta`=0, this results in returning a sample-size multiplier in `n.I` that can be used to set sample size at interim analyses according to the sample size required for a fixed design without interim analysis. That is, use a formula for a fixed design that matches the type of data you are collecting, and multiply this fixed design sample size by `n.I` to obtain sample sizes at interim and final analyses for the desired group sequential design.

3. Similarly, if you have a fixed design sample size which you wish to modify using `gsDesign()` to obtain an appropriate group sequential design, input `n.fix` as the fixed sample size design and input `delta`=0. In this case `n.I` returns with sample sizes for each analysis in the group sequential design.

- `upper`, `lower`: these are output as class `spendfn`, as described above. All elements of the `spendfn` class are returned when these are generated from a call to `gsDesign()`.

7 Formatted Output

`gsDesign()` returns an object of the class `gsDesign`. `gsProbability()` returns an object of class gsDesign if a `gsDesign` class object is passed in or of class `gsProbability`, if not. The standard R functions `print()`, and `plot()` are extended to work for both the `gsDesign` and the `gsProbability` classes. Note also that `summary()` prints a brief summary of either object type if you need a reminder of what is in a class. The `print()` function for each object class has a single argument and is implemented through the functions `print.gsDesign()` and `print.gsProbability()`. The `plot()` functions for `gsDesign` and `gsProbability` objects have a second argument that specifies which type of plot is to be made; these functions are implemented with the functions `plot.gsDesign()` and `plot.gsProbability()`, respectively. There are seven plot types, six of which are available for both classes, that are specified through the input variable `plottype`:

- `1` or `"Z"`: boundary plots (default if `class(x)="gsDesign"`)
- `2` or `"power"`: boundary crossing probability plots (default if `class(x)="gsProbability"`)
- `3` or `"thetahat"`: estimated treatment effect at boundaries
- `4` or `"CP"`: conditional power at boundaries
- `5` or `"sf"`: spending function plot (available only if `class(x)=="gsDesign"`)
- `6` or `"ASN"` or `"N"`: expected sample size plot
- `7"B"` or `"B-val"` or `"B-value"`: B-values at boundaries

Further details of the `print()`, and `plot()` functions are provided in Section 4, Detailed Examples, and in the help file for plotting in the gsDesign package. Since the plot functions have multiple arguments, the use of these arguments is formally specified here.

```
plot.gsDesign(x,plottype=1, main="Default", ylab="Default", xlab="Default")
plot.gsProbability(x,plottype=2, xval="Default", main="Default",
    ylab="Default", xlab="Default")
```

- `x`: for `plot.gsDesign()`, this is an object in the `gsDesign` class; for `plot.gsProbability()`, this is an object in the `gsProbability` class.
- `plottype`: described above.
- `theta`: used for `plottype` equal to 2, 4, or 6; normally defaults are adequate. See the help files for details.

- `ses=TRUE`: applies only when `plottype = 3` and `class(x) == "gsDesign"`; indicates that estimated standardized effect size at the boundary ($\hat{\theta}/\delta$) is to be plotted rather than the actual estimate ($\hat{\theta}$.
- `xval="Default"`: effective only when `plottype` equals 2 or 6. Appropriately scaled (reparameterized) values for x-axis for power and expected sample size graphs; see the help file for details.

8 Spending Functions

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. For users needing more background in spending functions, basic material is included in Section 9, Statistical Methods. 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.

8.1 Spending Function Basics

As noted previously, the default spending function for `gsDesign()` is `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. To change these to -3 (less conservative than an O'Brien-Fleming bound) and 1 (an aggressive Pocock-like bound), respectively, requires only the following:

```
> gsDesign(sfupar=-3, sflpar=1)
```

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) is specified as follows:

```
> gsDesign(sfu=sfPower, sfupar=3, sfl=sfPower, sflpar=0.75)}
```

O'Brien-Fleming, Pocock, and Wang-Tsiatis ($\Delta = 0.25$; between Pocock and O'Brien-Fleming) bounds can be obtained (only for `test.type = 1` or 2) as follows:

```
\texttt{%
> gsDesign(test.type=2, sfu="OF")
> gsDesign(test.type=2, sfu="Pocock")
> gsDesign(test.type=2, sfu="WT", sfupar=0.25)}
```

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 [4]. 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 5 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 [5], 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"))))
```

Users satisfied with these previously published and standard options may stop here!

8.2 Spending Function Details

Except for the "OF", "Pocock" and "WT" examples just given, a spending function passed to `gsDesign()` 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:

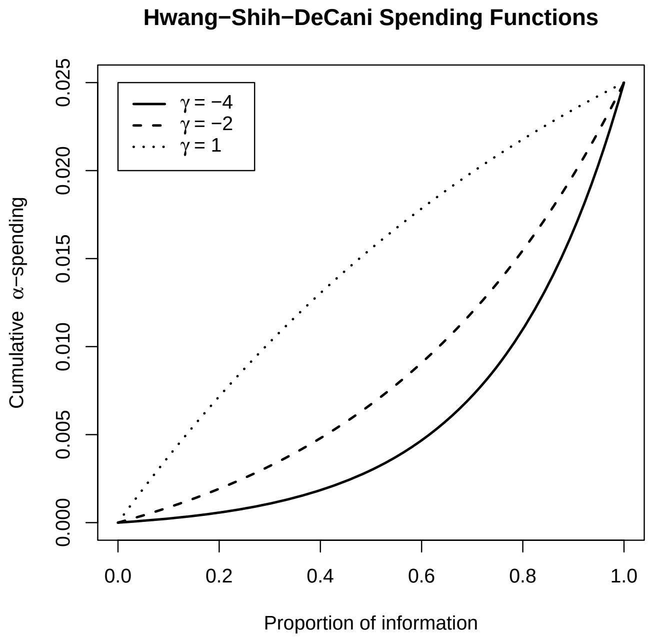

Figure 5: Hwang-Shih-DeCani spending function example

- `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 6.1, The `spendfn` Class.

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 by two investigational spending functions: `sfExponential()` and `sfLogistic()`. These spending functions are discussed further in Section 8.3, 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)
```

Table 1: 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
`sfPoints` (p$_1$,p$_2$,...,p$_K$)	Pointwise specification	Specified points $0<p_1\ldots<p_K=1$	Cumulative proportion of total boundary crossing probability for each analysis
`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 $\Delta=0$.

See also subsections below and online documentation of spending functions. Mathematical background is in Section 9, Statistical Methods.

8.3 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 [2], the GUSTO V trial [8] 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 8.6, 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:

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

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 (1) 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 [8]. 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 (1) we obtain the logistic spending function family:

$$\alpha(t; a, b) = \alpha \times \text{logit}^{-1}(\log(c) + b \times \text{logit}(u)) \tag{2}$$

$$= \alpha \frac{c\left(\frac{t}{1-t}\right)^b}{1 + c\left(\frac{t}{1-t}\right)^b} \tag{3}$$

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 $t > 1/2$.

```
> 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)
```

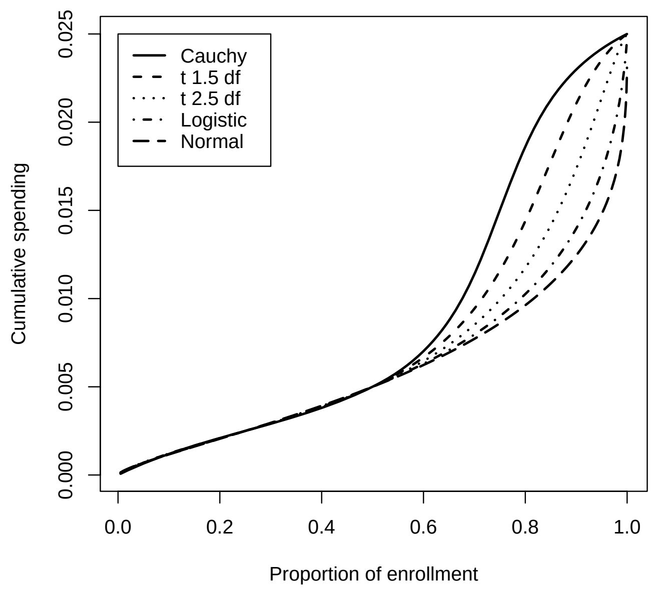

Figure 6: Example fitting two- and three-parameter spending functions

```
    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)
```

8.4 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.

8.5 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}
    x <- gsDesign(k=k, test.type=test.type, alpha=alpha, beta=beta,
                  astar=astar, delta=delta, n.fix=n.fix, timing=timing,
                  sfu=sfu,s fupar=x[1], sfl=sfl, sflpar=x[2], tol=tol, r=r)
    # compute boundary crossing probabilities for input theta
    x <- gsProbability(theta=theta, d=x)
    # compute sample sizes to the val power
    n <- x$n.I^val
    # compute expected values
    en <- n %*% (x$upper$prob +x$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=0.000000001)
> x$message
> y <- gsDesign(k=4, test.type=5, sfupar=x$par[1], sflpar=x$par[2])
> y
```

8.6 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)
}
```

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"))
```

9 Statistical Methods

Distributional assumptions. We begin with a sequence of normal random variates and later generalize to other situations. Let X_1, X_2,... be independent and identically distributed normal

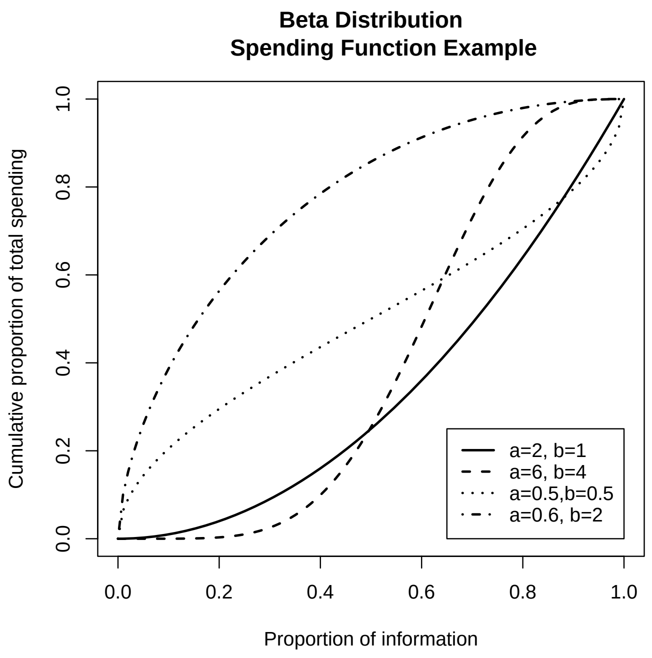

Figure 7: Example plotting user-written beta-distribution spending function

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

$$Z_i = \sum_{i=1}^{n_i} X_i / \sqrt{n_i}$$

and let $I_i = n_i$. The values $0 < I_1 < I_2 < \ldots < I_k$ represent the statistical information available at analyses 1 through k, respectively. The test statistics Z_1, Z_2,...,Z_k follow a multivariate normal distribution with, for $1 \leq j \leq i \leq k$,

$$E\{Z_i\} = \theta\sqrt{I_i} \tag{4}$$

$$Cov(Z_j, Z_i) = \sqrt{I_j/I_i} \tag{5}$$

Jennison and Turnbull [5] refer to (4) and (5) as the canonical form and present several types of outcomes (*e.g.*, binomial, time-to-event) for which.test statistics for comparison of treatment groups take this form asymptotically. Examples of how this is applied to a binomial trial are above in the DETAILED EXAMPLES section. Computational methods follow Chapter 19 of Jennison and Turnbull [5] and are not provided here. Note that other software programs such as EAST and the University of Wisconsin software use this distributional assumption as primary tools for group sequential design.

While we will work primarily with Z_1, Z_2,...,Z_k, we also define variables representing incremental sets of observations between analyses. 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

$$Y_i \tilde{} N(\sqrt{I_i - I_{i-1}}\theta, 1),\ i = 1, 2, ..., k. \tag{6}$$

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

$$Z_i = \frac{\sum_{j=1}^{i} \sqrt{I_j - I_{j-1}} Y_j}{\sqrt{I_i}} = \frac{\sum_{j=1}^{i} w_j Y_j}{\sqrt{\sum_{j=1}^{i} w_j^2}}. \tag{7}$$

Finally, we define notation for independent increments between arbitrary analyses. Select i and j with $1 \leq i < j \leq k$ 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

$$Z_j = \frac{\sqrt{I_i} Z_i + \sqrt{I_j - I_i} Z_{i,j}}{\sqrt{I_j}}. \tag{8}$$

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

$$Y_i = Z_{i-1,i}. \tag{9}$$

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

$$Z_{j,i} = \frac{\sqrt{I_i} Z_i - \sqrt{I_j} Z_j}{\sqrt{I_i - I_j}}. \tag{10}$$

The variables $Z_{j,i}$ and Z_j are independent, as before, for any $1 \leq j < i \leq k$. We use (9) to define Y_i, $i = 2, 3, ..., k$. As before $Y_i \tilde{} N(\sqrt{I_i - I_{i-1}}\theta, 1)$, $1 < i \leq k$, and these random variables are independent of each other.

Hypotheses and testing. We assume that the primary interest is to test the null hypothesis H_0: $\theta = 0$ against the alternative H_1: $\theta = \delta$ for a fixed $\delta > 0$. 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 $l_i < u_i$ are set; final cutoffs $l_K \leq u_K$ are also set. For $i = 1, 2, \ldots, K$, the trial is stopped at analysis i to reject H_0 if $l_j < Z_j < u_j$, $j = 1, 2, \ldots, i-1$ and $Z_i \geq u_i$. If the trial continues until stage i, H_0 is not rejected at stage i, and $Z_i \leq l_i$ then H_1 is rejected in favor of H_0, $i = 1, 2, \ldots, K$. Thus, $3K$ parameters define a group sequential design: l_i, u_i, and I_i, $i = 1, 2, \ldots, K$. Note that if $l_K < u_K$ there is the possibility of completing the trial without rejecting H_0 or H_1. We will generally restrict $l_K = u_K$ so that one hypothesis is rejected.

Sample size ratio for a group sequential design compared to a fixed design. Consider a trial with a fixed design 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

$$r = I_K / I_{fix} = N_K / N_{fix}. \tag{11}$$

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. Let $\Phi^{-1}()$ denote the inverse of the cumulative standard normal distribution function. In order for the fixed design with level α (1-sided), to have power 100(1–β)% to reject θ=0 when in fact $\theta = \delta$, we must have

$$I_{fix} = \left(\frac{\Phi^{-1}(1-\alpha) + \Phi^{-1}(1-\beta)}{\delta} \right)^2. \tag{12}$$

From (11) and (12), if we let $\delta = \Phi^{-1}(1-\alpha)+\Phi^{-1}(1-\beta)$ then $r = I_K$. Because of this relation, this is the value of δ that is used and displayed when information ratios are computed in `gsDesign()`.

Spending functions. For $i = 1, 2, \ldots, K$ denote the probability of stopping and rejecting the null hypothesis at analysis i as a function of θ by

$$\alpha_i(\theta) = P_\theta\{\{Z_i \geq u_i\} \cap_{j=1}^{i-1} \{l_j < Z_j < u_j\}\}. \tag{13}$$

The value $\alpha_i(0)$ is commonly referred to as the amount of α (Type I error rate) spent at analysis i, $1 \leq i \leq K$. The total Type I error rate for a trial is denoted by $\alpha \equiv \sum_{i=1}^{K} \alpha_i(0)$. We define a continuously increasing spending function $\alpha(t)$, $0 \leq t \leq 1$ with $\alpha\,(0) = 0$ and $\alpha\,(1) = \alpha$ that is used to set $\alpha_i(0)$, for $i = 1, 2, ...K$ using the relation

$$\alpha(I_i/I_K) = \sum_{j=1}^{i} \alpha_j(0). \tag{14}$$

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

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

For one-sided testing (`test.type`=1) and for non-binding lower bounds (`test.type`=4 or 6), (13) is not used for computing Type I error. In these cases, $\alpha_i(\theta)$ is replaced for $i = 1, 2, ...K$ by

$$\alpha_i^+(\theta) = P_\theta\{\{Z_i \geq u_i\} \cap_{j=1}^{i-1} \{Z_j < u_j\}\} \tag{15}$$

and a spending function $\alpha^+(t)$, $0 \leq t \leq 1$ is used to set boundaries using the relation

$$\alpha^+(I_i/I_K) = \sum_{j=1}^{i} \alpha_j^+(0). \tag{16}$$

Next, we consider analogous notation for the lower bound. For $i = 1, 2, \ldots, K$ denote

$$\beta_i(\theta) = P_\theta\{\{Z_i \leq l_i\} \cap_{j=1}^{i-1} \{l_j < Z_j < u_j\}\}. \tag{17}$$

The total lower boundary crossing probability in this case is written as $\beta(\theta) = \sum\limits_{i=1}^{K} \beta_i(\theta)$. The total Type II error rate for a trial is denoted by $\beta \equiv \sum_{i=1}^{K} \beta_i(\delta)$. For a specified value of θ we define an increasing function $\beta(t;\theta)$ on the interval $[0,1]$ with $\beta(0;\theta) = 0$ and $\beta(1;\theta) = \beta(\theta)$ where $\beta(\theta)$ is the desired total probability for crossing a lower boundary at any analysis when θ is the true parameter value. We now wish to set lower bounds using this spending function to obtain

$$\beta(I_i/I_K;\theta) = \sum_{j=1}^{i} \beta_j(\theta). \tag{18}$$

There are two options for how to use (18) to set lower bounds. For `test.type`=2, 5 and 6, we set lower boundary values under the null hypothesis by specifying $\beta(t;0)$, $0 \leq t \leq 1$. For `test.type`=3 and 4, we compute lower boundary values under the alternative hypothesis by specifying $\beta(t;\delta)$, $0 \leq t \leq 1$. $\beta(t;\delta)$ 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, $1 \leq i \leq K$.

We now have four ways of specifying bounds using spending functions. For upper bounds, we can use spending functions $\alpha(t)$ or $\alpha^+(t)$ while for lower spending we can use spending functions $\beta(t;0)$ or $\beta(t;\theta)$.These are summarized in the following table:

Table 2. Spending functions by `test.type`.

Upper bound spending function	Lower bound spending function	`test.type`
$\alpha^+(t)$	None	1
$\alpha(t)$	$\beta(t;\delta)$	3
$\alpha^+(t)$	$\beta(t;\delta)$	4
$\alpha(t)$	$\beta(t;0)$	2, 5
$\alpha^+(t)$	$\beta(t;0)$	6

For `test.type`=1, 2 and 5, boundaries can be computed in a single step as outlined by Jennison and Turnbull [5], Chapter 19 just by knowing the proportion of the total planned sampling 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. In these cases, the total sample size is then set to obtain the desired power under the alternative hypothesis using a root finding algorithm.

For `test.type`=3 and 4 the sample size and bounds are all set simultaneously using an iterative algorithm. This computation is relatively straightforward, but 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.

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 probabilities 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

$$u_{m,j}(z_i) = \frac{u_j\sqrt{I_j} - z_i\sqrt{I_m}}{\sqrt{(I_j - I_m)}} \tag{19}$$

and

$$l_{m,j}(z_i) = \frac{l_j\sqrt{I_j} - z_i\sqrt{I_m}}{\sqrt{(I_j - I_m)}}. \tag{20}$$

Recall (8) and consider the conditional probabilities

$$\begin{aligned}
\alpha_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \geqslant u_j\} \cap_{m=i+1}^{j-1} \{l_m < Z_m < u_m\}|Z_i = z_i\} \qquad (21)\\
&= P_\theta\left\{\left\{\frac{\sqrt{I_i}z_i + \sqrt{I_j - I_i}Z_{i,j}}{\sqrt{I_j}} \geqslant u_j\right\} \cap_{m=i+1}^{j-1} \left\{l_m < \frac{\sqrt{I_i}z_i + \sqrt{I_j - I_i}Z_{i,m}}{\sqrt{I_j}}\right\} < u_m\right\}\\
&= P_\theta\{\{Z_{i,j} \geqslant u_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{l_{m,j}(z_i) < Z_{m,j} < u_{m,j}(z_i)\}\}.
\end{aligned}$$

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

$$\begin{aligned}
\alpha^+_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \geqslant u_j\} \cap_{m=i+1}^{j-1} \{Z_m < u_m\}|Z_i = z_i\} \qquad (22)\\
&= P_\theta\{\{Z_{i,j} \geqslant u_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{Z_{m,j} < u_{m,j}(z_i)\}\}.
\end{aligned}$$

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

$$\begin{aligned}
\beta_{i,j}(\theta|z_i) &= P_\theta\{\{Z_j \le l_j\} \cap_{m=i+1}^{j-1} \{l_m < Z_m < u_m\}|Z_i = z_i\} \qquad (23)\\
&= P_\theta\{\{Z_{i,j} \le l_{i,j}(z_i)\} \cap_{m=i+1}^{j-1} \{l_{m,j}(z_i) < Z_{m,j} < u_{m,j}(z_i)\}\}.
\end{aligned}$$

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.

10 Function and Class Reference

10.1 gsDesign Package

`gsDesign-package` *1.0 Group Sequential Design*

Description

gsDesign is a package that derives 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:	2.0
License:	GPL (version 2 or later)

Index:

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 controls 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)
```

10.2 gsDesign main functions

gsDesign *2.1: Design Derivation*

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 (`test.type=1, 2`), `sfu` is used to completely specify spending. The default value is `sfHSD` which is a Hwang-Shih-DeCani spending function. See details, Spending Functions, 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. `sfupar` 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`. 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). 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`	As input.
`tol`	As input.
`r`	As input.
`upper`	Upper bound spending function, boundary and boundary crossing probabilities under the NULL and alternate hypotheses. See Spending Functions and manual for further details.
`lower`	Lower bound spending function, boundary and boundary crossing probability 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 Functions 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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

gsDesign-package, Group Sequential Plots, `gsProbability`, Spending Functions, 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)
```

`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`	An 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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

Group Sequential Plots, `gsDesign`, gsDesign-package

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 in the `gsDesign` and `gsProbability` classes (that is, objects returned by the functions `gsDesign` and `gsProbability`, respectively). 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), 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: 1) `theta` which is used for `plottype=2`, 4, 6; normally defaults will be adequate; see details. 2) `ses=TRUE` which applies only when `plottype=3` and `class(x)=="gsDesign"`; indicates that estimated standardized effect size at the boundary ($hat(theta)/delta$)) is to be plotted rather than the actual estimate ($hat(theta)$ if `ses==FALSE`). 3) `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 in most cases, although 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 plots (`plottype=2,6`) for a `gsDesign` object. A call to `x<-gsDesign)()` produces power for only two *theta* values: 0 and *delta*. The call `plot(x,plottype=2,theta=theta)` (or `plot(x,plottype=6,theta=theta)` for a `gsDesign` object produces power (expected sample size) curves and returned a `gsDesign` object with *theta* values determined as follows. If `psi` is not specified, input values of `theta` are used (default is `theta=seq(0,2,.05)*x$delta`) and the default `xval=seq(0,2,.05)` so that standardized effect sizes are plotted on the x-axis. If `psi` is specified, then values are plotted for `theta=theta/psi`; that is, input values of `theta` are re-scaled; default x-values for plotting are on the . Rather than plotting the x-axis as *theta* values, values of $theta/delta$ are plotted in this case. See examples below.

Estimated treatment effects at boundaries are computed multiplying 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, 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/manual.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, `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
`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 Statistical Methods 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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

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
```

`gsBoundCP` *2.5: Conditional Power at Interim Boundaries*

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 Statistical Methods 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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

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)
```

10.3 Binomial trial functions

`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/manual.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

`gsProbability`, `gsDesign`, Group Sequential Plots.

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$z)
cat("Expected sample size averaged over normal prior distribution for theta with mu=",
   x$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")
```

`nBinomial` *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, computations based on risk-ratio and odds-ratio are also provided.

`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 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()`.

`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()`, 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()` all arguments must have length 1. In general, arguments for `nBinomial()`, `testBinomial()` and `ciBinomial` may be scalars or vectors, in which case they return a vector of sample sizes and powers, respectively. There can 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. If `scale="RR"`, `delta0` is the relative risk of event rates under the null hypothesis minus 1 (`p2 / p1 - 1`). If `scale="LNOR"`, `delta0` is the difference in natural logarithm of the odds-ratio under the null hypothesis `log(p2 / (1 - p2)) - log(p1 / (1 - p1))`.
`ratio`	sample size ratio for group 2 divided by group 1
`sided`	2 for 2-sided test, 1 for 1-sided test

`outtype`	1 (default) returns total sample size; 2 returns sample size for each group (`n1, n2`);
`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 character string (vector of character strings If 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, which `p10` and `p20` are used under the null hypothesis.

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)` or `qchisq(.95,1)`).

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 treatment effect used for null hypothesis
`p20`	group 2 treatment effect 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 500k simulations to test validity of the above asymptotic p-values
sum(as.real(x0) <= simBinomial(p1=.078, p2=.078, n1=500, n2=500, nsim=500000)) / 500000
sum(as.real(x0) <= simBinomial(p1=.052, p2=.052, n1=500, n2=500, nsim=500000)) / 500000

# 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=1000000)) / 1000000

# compute a sample size to show non-inferiority with 5
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 time-to-event endpoint. The Lachin and Faulkes (1986) method is used.

Usage

```
nSurvival(lambda.0, lambda.1, eta = 0, rand.ratio = 1, Ts, Tr,
      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 duration.
`alpha`	type I error rate. Default is 0.025 for one-sided.
`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 Faulkes 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 = 0.025 (one-sided)
# power = 0.9

h0 <- 0.2
h1 <- 0.1
eta <- 0.1
rand.ratio <- 1
tm1 <- 2
tm2 <- 0.5
gamma <- NA
alpha <- 0.05
beta <- 0.1

ss <- nSurvival(h0, h1, eta = eta, rand.ratio = rand.ratio,
             Ts = tm1, Tr = tm2,
             alpha = alpha, beta = beta, sided = 1,
             type = "rd", entry = "unif", gamma = gamma)

#  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 = ss$Sample.size)
        x
        plot(x)
        plot(x, plottype = 2)
```

10.4 Spending Functions

`Spending Functions` *4.0: Spending functions*

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. Another somewhat flexible approach to 1-sided and symmetric 2-sided designs is using Wang-Tsiatis boundaries which include O'Brien-Fleming and Pocock designs.

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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

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

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 function 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
plot(0:100/100, sfbdist(1, 0:100/100, c(2, 1))$spend, type="l",
    xlab="Proportion of information",
    ylab="Cumulative proportion of total spending",
    main="Beta distribution Spending Function Example")
lines(0:100/100, sfbdist(1, 0:100/100, c(6, 4))$spend, lty=2)
lines(0:100/100, sfbdist(1, 0:100/100, c(.5, .5))$spend, lty=3)
lines(0:100/100, sfbdist(1, 0:100/100, c(.6, 2))$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"))
```

`sfHSD` *4.1: Hwang-Shih-DeCani Spending Function*

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

Details

A Hwang-Shih-DeCani spending function takes the form

$$\alpha(1 - e^{-\gamma t})/(1 - e^{-\gamma})$$

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 Functions for further details.

Note

The manual is not linked to this help file, but is available in library/gsdesign/doc/manual.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 Functions, `gsDesign`, gsDesign-package

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
plot(0:100/100,  sfHSD(0.025, 0:100/100, -4)$spend, xlab="Proportion of final sample size",
   ylab="Cumulative Type I error spending",
   main="Hwang-Shih-DeCani Spending Function Example", type="l")
lines(0:100/100, sfHSD(0.025, 0:100/100, -2)$spend, lty=2)
lines(0:100/100, sfHSD(0.025, 0:100/100, 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

Details

A Kim-DeMets spending function takes the form

$$\alpha t^{\rho}$$

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 Functions for further details.

Note

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

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

Spending Functions, `gsDesign`, gsDesign-package

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
plot(0:100/100,  sfPower(0.025, 0:100/100, 3)$spend, xlab="Proportion of sample size",
    ylab="Cumulative Type I error spending",
    main="Kim-DeMets (rho) versus Hwang-Shih-DeCani (gamma) Spending",
    type="l", cex.main=.9)
lines(0:100/100, sfPower(0.025, 0:100/100, 2)$spend, lty=2)
lines(0:100/100, sfPower(0.025, 0:100/100, 0.75)$spend, lty=3)
lines(0:100/100, sfHSD(0.025, 0:100/100, 1)$spend, lty=3, col=2)
lines(0:100/100, sfHSD(0.025, 0:100/100, -2)$spend, lty=2, col=2)
lines(0:100/100, sfHSD(0.025, 0:100/100, -4)$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 an investigational (unpublished) spending function called the exponential spending function. 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.

Details

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

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

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

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

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()`),

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

Value

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

Note

The manual shows how to closely approximate an O'Brien-Fleming design using `sfExponential()`. An example is given below. The manual is not linked to this help file, but is available in library/gsdesign/doc/manual.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 Functions, `gsDesign`, gsDesign-package

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
plot(0:100/100, sfExponential(0.025, 0:100/100, 0.8)$spend, xlab="Proportion of final sample size",
   ylab="Cumulative Type I error spending",
   main="Exponential Spending Function Example", type="l")
lines(0:100/100, sfExponential(0.025, 0:100/100, 0.5)$spend, lty=2)
lines(0:100/100, sfExponential(0.025, 0:100/100, 0.3)$spend, lty=3)
lines(0:100/100, sfExponential(0.025, 0:100/100, 0.2)$spend, lty=4)
lines(0:100/100, sfExponential(0.025, 0:100/100, 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 Functions*

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 the gsDesign package.

Details

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

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

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

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

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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

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 Functions, `gsDesign`, gsDesign-package

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. Normally `sfpoint` 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. Given the wide variety of flexible spending functions provided in this package, `sfPoints` is not generally recommended since no interpolation function is provided to adjust boundaries if timing of interims changes from the original plan (see example).

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/manual.pdf in the directory where R is installed.

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

Spending Functions, `gsDesign`, gsDesign-package

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
lines(0:100/100, sfTDist(1, 0:100/100, 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")
```

`sfLogistic` *4.6: 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

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

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

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

For the extreme value distribution with

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

this simplifies to

$$\alpha \exp(-e^a(-\ln t)^b).$$

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 Functions` for further details.

Note

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

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

Spending Functions, `gsDesign`, gsDesign-package

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
plot(0:100/100,  sfLogistic(1, 0:100/100, c(.1, .4, .01, .1))$spend,
    xlab="Proportion of final sample size",
    ylab="Cumulative Type I error spending",
    main="Logistic Spending Function Examples",
    type="l", cex.main=.9)
lines(0:100/100, sfLogistic(1, 0:100/100, c(.01, .1, .1, .4))$spend, lty=2)
```

```
# now just give a=0 and b=1 as 3rd parameters for sfLogistic
lines(0:100/100, sfLogistic(1, 0:100/100, c(0, 1))$spend, lty=3)

# try a couple with unconventional shapes again using the xy form in the 3rd parameter
lines(0:100/100, sfLogistic(1, 0:100/100, c(.4, .6, .1, .7))$spend, lty=4)
lines(0:100/100, sfLogistic(1, 0:100/100, c(.1, .7, .4, .6))$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=seq(0, 1, .01), param)
```

---

sfTDist *4.7: 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

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

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/manual.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 Functions, `gsDesign`, gsDesign-package

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
# final parameter is df
sfTDist(1, 1:3/4, c(.25, .5, .1, .2, 4))$spend

# 6-parameter specification fits 3 points,  in this case all of the interims
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
plot(0:100/100, sfTDist(0.025, 0:100/100, c(.25, .5, .1, .2, 1))$spend,
    xlab="Proportion of final sample size",
    ylab="Cumulative Type I error spending",
    main="t-Distribution Spending Function Examples", type="l")
lines(0:100/100, sfTDist(0.025, 0:100/100, c(.25, .5, .1, .2, 1.5))$spend, lty=2)
lines(0:100/100, sfTDist(0.025, 0:100/100, c(.25, .5, .1, .2, 3))$spend, lty=3)
lines(0:100/100, sfTDist(0.025, 0:100/100, c(.25, .5, .1, .2, 10))$spend, lty=4)
lines(0:100/100, sfTDist(0.025, 0:100/100, 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"))
```

10.5 Other Files

`Wang-Tsiatis Bounds` *5.0: Wang-Tsiatis Bounds*

Description

`gsDesign` offers the option of using Wang-Tsiatis bounds as an alternative to the spending function approach to group sequential design. Wang-Tsiatis bounds include both Pocock and O'Brien-Fleming designs. Wang-Tsiatis bounds are currently only available for 1-sided and symmetric 2-sided designs. Wang-Tsiatis bounds are typically used with equally spaced timing between analyses, but the option is available to use them with unequal spacing.

Details

Wang-Tsiatis bounds are defined as follows. Assume k analyses and let Z_i represent the upper bound and t_i the proportion of the total planned sample size for the i-th analysis, $i = 1, 2, \ldots, k$.

Let Δ be a real-value. Typically Δ will range from 0 (O'Brien-Fleming design) to 0.5 (Pocock design). The upper boundary is defined by

$$ct_i^{\Delta-0.5}$$

for $i = 1, 2, \ldots, k$ where c depends on the other parameters. The parameter Δ is supplied to `gsDesign()` in the parameter `sfupar`. For O'Brien-Fleming and Pocock designs there is also a calling sequence that does not require a parameter. See examples.

Note

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

Author(s)

Keaven Anderson ⟨keaven_anderson@merck.com⟩, Jennifer Sun, John Zhang

References

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

See Also

`Spending Functions`, `Spending Functions`, `gsProbability`

Examples

```
# Pocock design
gsDesign(test.type=2, sfu="Pocock")

# alternate call to get Pocock design specified using Wang-Tsiatis option and Delta=0.5
gsDesign(test.type=2, sfu="WT", sfupar=0.5)

# this is how this might work with a spending function approach
# Hwang-Shih-DeCani spending function with gamma=1 is often used to approximate Pocock design
gsDesign(test.type=2, sfu=sfHSD, sfupar=1)

# unequal spacing works,  but may not be desirable
gsDesign(test.type=2, sfu="Pocock", timing=c(.1, .2))

# spending function approximation to Pocock with unequal spacing
# is quite different from this
gsDesign(test.type=2, sfu=sfHSD, sfupar=1, timing=c(.1, .2))

# One-sided O'Brien-Fleming design
gsDesign(test.type=1, sfu="OF")

# alternate call to get O'Brien-Fleming design specified using Wang-Tsiatis option and Delta=0
gsDesign(test.type=1, sfu="WT", sfupar=0)
```

`checkScalar` *verify variable properties*

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)), to
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
try(checkVector(x, "numeric", c(1, 10), c(TRUE, FALSE), length=2)) # expect error

# 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)
```

`gsBound` *Boundary derivation*

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

`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 print 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

Forthcoming...

Value

list(k=xx[[1]],theta=0.,I=xx[[2]],a=xx[[3]],b=xx[[4]],rates=rates,tol=xx[[7]], r=xx[[8]],error=xx[[9]])
y¡-list(k=xx[[1]],theta=xx[[2]],I=xx[[3]],a=xx[[4]],b=xx[[5]], problo=xx[[6]],probhi=xx[[7]],tol=xx[[8]],r=xx[[9]],er
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`	
`b`	
`tol`	As input.
`r`	As input.
`error`	Error code. 0 if no error; ¿ 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.com⟩, Jennifer Sun, John Zhang

References

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

See Also

gsDesign-package, `gsDesign`, `gsProbability`

Examples

```
#  set boundaries so that probability is .01 of first crossing each upper boundary
#  and .02 of crossing each lower boundary under 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)
```

References

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

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

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

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

11 Contacts

Keaven M. Anderson: keaven_anderson@merck.com