

Package ‘gsDesign’

February 14, 2012

Version 2.6-04

Title Group Sequential Design

Author Keaven Anderson

Maintainer Keaven Anderson <keaven_anderson@merck.com>

Depends ggplot2, xtable

Description gsDesign is a package that derives group sequential designs and describes their properties.

License GPL (>= 2)

Copyright Copyright 2010, Merck Research Laboratories

Repository CRAN

Repository/R-Forge/Project gsdesign

Repository/R-Forge/Revision 331

Date/Publication 2012-02-13 14:51:32

R topics documented:

Binomial	2
checkScalar	7
gsBinomialExact	9
gsBound	11
gsBoundCP	13
gsBoundSummary	15
gsCP	18
gsDensity	22
gsDesign	24
gsDesign package overview	29
gsProbability	31
nNormal	33
normalGrid	35

nSurvival	37
plot.gsDesign	41
sfExponential	44
sfHSD	46
sfLDOF	48
sfLinear	50
sfLogistic	51
sfPoints	54
sfPower	56
sfTDist	58
sfTruncated	60
Spending functions	62
Wang-Tsiatis Bounds	64

Index	67
--------------	-----------

Binomial	<i>3.2: Testing, Confidence Intervals and Sample Size for Comparing Two Binomial Rates</i>
----------	--

Description

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

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

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

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

`simBinomial()` performs simulations to estimate the power for a Miettinen and Nurminen (1980) test comparing two binomial rates for superiority or non-inferiority. As noted in documentation for `bpower.sim()` in the `HMisc` package, by using `testBinomial()` you can see that the formulas without any continuity correction are quite accurate. In fact, Type I error for a continuity-corrected

test is significantly lower (Gordon and Watson, 1996) than the nominal rate. Thus, as a default no continuity corrections are performed.

Usage

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

Arguments

For `simBinomial()` and `ciBinomial()` all arguments must have length 1.
 For `testBinomial()`, `x1`, `x2`, `n1`, `n2`, `delta0`, `chisq`, and `adj` may be vectors.
 For `nBinomial()`, `p1`, `p2`, `beta`, `delta0` and `ratio` may be vectors.
 For `nBinomial()` or `testBinomial()`, when one or more arguments is a vector, the routines return a vector of sample sizes and powers, respectively. Where vector arguments are allowed, there may be a mix of scalar and vector arguments. All arguments specified using vectors must have the same length.

	event rate in group 1 under the alternative hypothesis
<code>p2</code>	event rate in group 2 under the alternative hypothesis
<code>alpha</code>	type I error; see <code>sided</code> below to distinguish between 1- and 2-sided tests
<code>beta</code>	type II error
<code>delta0</code>	A value of 0 (the default) always represents no difference between treatment groups under the null hypothesis. <code>delta0</code> is interpreted differently depending on the value of the parameter <code>scale</code> . If <code>scale="Difference"</code> (the default), <code>delta0</code> is the difference in event rates under the null hypothesis ($p_{10} - p_{20}$). If <code>scale="RR"</code> , <code>delta0</code> is the logarithm of the relative risk of event rates (p_{10} / p_{20}) under the null hypothesis. If <code>scale="LNOR"</code> , <code>delta0</code> is the difference in natural logarithm of the odds-ratio under the null hypothesis $\log(p_{10} / (1 - p_{10})) - \log(p_{20} / (1 - p_{20}))$.
<code>ratio</code>	sample size ratio for group 2 divided by group 1
<code>sided</code>	2 for 2-sided test, 1 for 1-sided test
<code>outtype</code>	<code>nBinomial</code> only; (default) returns total sample size; 2 returns sample size for each group (<code>n1</code> , <code>n2</code> ; 3 returns additional interim calculations); 3 and <code>delta0=0</code> returns a list with total sample size (<code>n</code>), sample size in each group (<code>n1</code> , <code>n2</code>), null and alternate hypothesis variance (<code>sigma0</code> , <code>sigma1</code>), input event rates (<code>p1</code> , <code>p2</code>) and null hypothesis event rates (<code>p10</code> , <code>p20</code>).
<code>x1</code>	Number of "successes" in the control group
<code>x2</code>	Number of "successes" in the experimental group
<code>n1</code>	Number of observations in the control group

n2	Number of observations in the experimental group
chisq	An indicator of whether or not a chi-square (as opposed to Z) statistic is to be computed. If $\delta_0=0$ (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 $\text{adj}=1$, 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 $\text{adj}=0$ since nominal Type I error is generally conservative with $\text{adj}=1$ (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 <code>simBinomial()</code>
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 p_1 and p_2 under the alternative hypothesis and a difference between these rates under the null hypothesis, δ_0 . From these values, actual rates under the null hypothesis are computed, which are labeled p_{10} and p_{20} when `outtype=3`. The rates p_1 and p_2 are used to compute a variance for a Z-test comparing rates under the alternative hypothesis, while p_{10} and p_{20} are used under the null hypothesis.

Sample size with `scale="Difference"` produces an error if $p_1-p_2=\delta_0$. Normally, the alternative hypothesis under consideration would be $p_1-p_2-\delta_0>0$. However, the alternative can have $p_1-p_2-\delta_0<0$.

Value

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

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

n	A vector with total samples size required for each event rate comparison specified
n1	A vector of sample sizes for group 1 for each event rate comparison specified
n2	A vector of sample sizes for group 2 for each event rate comparison specified
sigma0	A vector containing the variance of the treatment effect difference under the null hypothesis

sigma1	A vector containing the variance of the treatment effect difference under the alternative hypothesis
p1	As input
p2	As input
pbar	Returned only for superiority testing ($\delta_0=0$), the weighted average of p1 and p2 using weights n1 and n2

When $\delta_0=0$, instead of pbar, the following 2 vectors are returned (see details):

p10	group 1 event rate used for null hypothesis
p20	group 2 event rate used for null hypothesis

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

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

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

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

Miettinen, 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 50k simulations to test validity of the above
# asymptotic p-values
# (you may want to perform more to reduce standard error of estimate)
sum(as.real(x0) <=
  simBinomial(p1=.078, p2=.078, n1=500, n2=500, nsim=10000)) / 10000
sum(as.real(x0) <=
  simBinomial(p1=.052, p2=.052, n1=500, n2=500, nsim=10000)) / 10000
```

```

# Perform a non-inferiority test to see if p2=400 / 500 is within 5% of
# 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)

# provide 95% CI (Miettinen and Nurminen method)
ciBinomial(x1=410, x2=400, n1=500, n2=500)

# now simulate the z-statistic without continuity corrected variance
sum(qnorm(.975) <=
    simBinomial(p1=.8, p2=.8, n1=500, n2=500, nsim=100000)) / 100000

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

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

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

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

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

```

```
"40 pct reduction", "50 pct reduction"))
```

 checkScalar

 6.0 Utility functions to 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)), tol=0)
checkLengths(..., allowSingle=FALSE)
```

Arguments

x	any object.
isType	character string defining the class that the input object is expected to be.
length	integer specifying the expected length of the object in the case it is a vector. If length=NULL, the default, then no length check is performed.
interval	two-element numeric vector defining the interval over which the input object is expected to be contained. Use the inclusion argument to define the boundary behavior.
inclusion	two-element logical vector defining the boundary behavior of the specified interval. A TRUE value denotes inclusion of the corresponding boundary. For example, if interval=c(3,6) and inclusion=c(FALSE,TRUE), then all the values of the input object are verified to be on the interval (3,6].
varname	character string defining the name of the input variable as sent into the function by the caller. This is used primarily as a mechanism to specify the name of the variable being tested when checkRange is being called within a function.
tol	numeric scalar defining the tolerance to use in testing the intervals of the checkRange function.
...	For the checkScalar and checkVector functions, this input represents additional arguments sent directly to the checkRange function. For the checkLengths function, this input represents the arguments to check for equal lengths.
allowSingle	logical flag. If TRUE, arguments that are vectors comprised of a single element are not included in the comparative length test in the checkLengths function. Partial matching on the name of this argument is not performed so you must specify 'allowSingle' in its entirety in the call.

Details

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

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

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

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

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

Examples

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

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

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

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

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

# create faux function to check input variable
foo <- function(moo) checkVector(moo, "character")
foo(letters)
try(foo(1:5)) # expect error with function and argument name in message

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

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

Description

Computes power/Type I error and expected sample size for a group sequential design in a single-arm trial with a binary outcome. The print function has been extended using `print.gsBinomialExact` to print `gsBinomialExact` objects; see examples.

Usage

```
gsBinomialExact(k=2, theta=c(.1, .2), n.I=c(50, 100), a=c(3, 7), b=c(20,30))
```

Arguments

<code>k</code>	Number of analyses planned, including interim and final.
<code>theta</code>	Vector of possible underlying binomial probabilities for a single binomial sample.
<code>n.I</code>	Sample size at analyses (increasing positive integers); vector of length <code>k</code> .
<code>a</code>	Number of "successes" required to cross lower bound cutoffs for futility or harm at each analysis; vector of length <code>k</code> ; -1 means no lower bound.
<code>b</code>	Number of "successes" required to cross upper bound cutoffs for futility or harm at each analysis; vector of length <code>k</code> .

Details

Based on the book "Group Sequential Methods with Applications to Clinical Trials," Christopher Jennison and Bruce W. Turnbull, Chapter 12, Section 12.1.2 Exact Calculations for Binary Data. This computation is often used as an approximation for the distribution of the number of events in one treatment group out of all events when the probability of an event is small and sample size is large.

An object of class `gsBinomialExact` is returned. On output, the values of `theta` input to `gsBinomialExact` will be the parameter values for which the boundary crossing probabilities and expected sample sizes are computed.

Note that `a[1]` equal to -1 lower bound at `n.I[1]` means 0 successes continues at interim 1; `a[2]==0` at interim 2 means 0 successes stops trial for futility at 2nd analysis. For final analysis, set `a[k]` equal to `b[k]-1` to incorporate all possibilities into non-positive trial; see example.

Value

`gsBinomialExact()` returns a list of class `gsBinomialExact` and `gsProbability` (see example); when displaying one of these objects, the default function to print is `print.gsProbability()`. The object returned from `gsBinomialExact()` contains the following elements:

<code>k</code>	As input.
<code>theta</code>	As input.

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

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)

Jon Hartzel with modifications for gsDesign package by Yevgen Tymofyeyev and Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsProbability](#)

Examples

```
zz <- gsBinomialExact(k=3,theta=seq(0,1,0.1), n.I=c(12,24,36),
  a=c(-1, 0, 11), b=c( 5, 9, 12))

# let's see what class this is
class(zz)

# because of "gsProbability" class above, following is equivalent to
# print.gsProbability(zz)
zz
```

Description

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

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

Usage

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

Arguments

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

Details

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

Value

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

k	The length of vectors input; a scalar.
theta	As input in <code>gsBound1()</code> ; 0 for <code>gsBound()</code> .
I	As input.
a	For <code>gsbound1</code> , this is as input. For <code>gsbound</code> this is the derived lower boundary required to yield the input boundary crossing probabilities under the null hypothesis.
b	The derived upper boundary required to yield the input boundary crossing probabilities under the null hypothesis.
tol	As input.
r	As input.
error	Error code. 0 if no error; greater than 0 otherwise.

`gsBound()` also returns the following items:

rates	a list containing two items:
falsepos	vector of upper boundary crossing probabilities as input.
trueneg	vector of lower boundary crossing probabilities as input.

`gsBound1()` also returns the following items:

problo	vector of lower boundary crossing probabilities; computed using input lower bound and derived upper bound.
probhi	vector of upper boundary crossing probabilities as input.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsDesign package overview](#), [gsDesign](#), [gsProbability](#)

Examples

```

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

# use gsBound1 to set up boundary for a 1-sided test
x <- gsBound1(theta= 0, I=c(1, 2, 3) / 3, a=array(-20, 3),
              probhi=c(.001, .009, .015))
x$b

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

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

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

```

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 ($z_i/\sqrt{x\$n.I[i]}$) at analysis i, interim test statistic z_i and interim sample size/statistical information of $x\$n.I[i]$). Otherwise, conditional power is computed assuming the input scalar value theta.
r	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally r will not be changed by the user.

Details

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

Value

A list containing two vectors, CPlo and CPhi.

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

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

See Also

[gsDesign](#), [gsProbability](#), [gsCP](#)

Examples

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

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

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

gsBoundSummary

2.8: Bound Summary and Z-transformations

Description

Z-value test statistic transformations are commonly used as supplementary descriptions of bounds. `gsBoundSummary()` summarizes a bound for a group sequential design with by computing several of these design characteristics. `xtable.gsDesign` produces Latex output of boundary summary information; intended for use with `Sweave()`.

Individual transformation of z-value test statistics for interim and final analyses are obtained from `gsBValue()`, `gsDelta()`, `gsHR()` and `gsCPz()` for B-values, approximate treatment effect (see details), approximate hazard ratio and conditional power, respectively.

Usage

```
gsBValue(z, i, x, ylab="B-value", ...)
gsDelta(z, i, x, ylab=NULL, ...)
gsHR(z, i, x, ratio=1, ylab="Estimated hazard ratio", ...)
gsRR(z, i, x, ratio=1, ylab="Estimated risk ratio",...)
gsCPz(z, i, x, theta=NULL, ylab=NULL, ...)
gsBoundSummary(x, upper=TRUE, ratio=1)
## S3 method for class 'gsDesign'
xtable(x, caption=NULL, label=NULL, align=NULL, digits=c(0,0,3,4,4,4,3,3,3,3),
       display=NULL, upper=TRUE, rnames=NULL, cnames=NULL, ratio=1,
       sanitize.text.function=function(x){x},
       sanitize.rownames.function=function(x){x},...)
```

Arguments

x	A <code>gsDesign</code> object
upper	If TRUE, upper bound is summarized. If FALSE, lower bound is summarized
z	A vector of z-statistics
i	A vector containing the analysis for each element in z; each element must be in 1 to x\$k, inclusive

<code>theta</code>	A scalar value used for conditional power calculations; see <code>gsDesign</code> ; if <code>NULL</code> , conditional power is computed at the estimated interim treatment effect based on <code>z</code>
<code>ylab</code>	Used when functions are passed to <code>plot.gsDesign</code> to establish default y-axis labels
<code>ratio</code>	Used only for time-to-event studies; randomization ratio for experimental versus control group
<code>digits</code>	An integer vector containing the digits to be displayed for each row
<code>rnames</code>	Allows user to override default row names for output matrix
<code>cnames</code>	Allows user to override default column names for output matrix
<code>...</code>	This allows many optional arguments that are standard when calling <code>xtable</code> and <code>plot</code>
<code>caption</code>	See documentation for <code>xtable</code>
<code>label</code>	See documentation for <code>xtable</code>
<code>align</code>	See documentation for <code>xtable</code>
<code>display</code>	See documentation for <code>xtable</code>
<code>sanitize.text.function</code>	Should be able to use default; see documentation for <code>xtable</code> ,
<code>sanitize.rownames.function</code>	Should be able to use default; see documentation for <code>xtable</code> and <code>print</code>

Details

See examples and manual

Value

`gsBValue()`, `gsDelta()`, `gsHR()` and `gsCPz()` each returns a vector containing the B-values, approximate treatment effect (see details), approximate hazard ratio and conditional power, respectively, for each value specified by the interim test statistics in `z` at interim analyses specified in `i`.

`gsBoundSummary` returns a matrix. The rows correspond to the analyses planned in `x`. For each interim, the columns provide 1) timing (%), 2) sample size ("N"), events ("Events") or sample size inflation relative to a fixed design ("r"), 3) Z, 4) nominal p-value (upper tail for upper bound, lower tail for lower bound), 5) cumulative boundary crossing probability (spend) under the null hypothesis, 6) cumulative spend under the alternate hypothesis, 7) conditional power at the bound assuming the alternate hypothesis treatment effect, 8) conditional power at the bound assuming the interim trend, 9) treatment effect at the boundary (hazard ratio for time-to-event endpoint studies), 10) B-value.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsDesign](#), [Plots for group sequential designs](#), [gsProbability](#)

Examples

```
# derive group sequential bound
x <- gsDesign(n.fix=100)

# upper bound summary
gsBoundSummary(x)

# lower bound summary
gsBoundSummary(x, upper=FALSE)

# print upper bound summary in latex format
xtable(x)

# print lower bound summary in latex format
xtable(x, upper=FALSE)

# now derive a design with a time-to-event endpoint
# note that by specifying endpoint="Survival", hazard ratio replaces treatment effect
ns <- nSurvival()
xs <- gsDesign(n.fix=ns$nEvents, nFixSurv=ns$n, endpoint="Survival")
gsBoundSummary(xs)
# generate some of the above summary statistics for the upper bound
z <- xs$upper$bound
# B-values
gsBValue(z=z, i=1:3, x=xs)
# hazard ratio
gsHR(z=z, i=1:3, x=xs)
# conditional power at observed treatment effect
gsCPz(z=z[1:2], i=1:2, x=xs)
# conditional power at H1 treatment effect
gsCPz(z=z[1:2], i=1:2, x=xs, theta=xs$delta)

# now look at a binomial endpoint; specify H0 treatment difference as p1-p2=.05
# now treatment effect at bound (say, thetahat) is transformed to
# xp$delta0 + xp$delta1*(thetahat-xp$delta0)/xp$delta
np <- nBinomial(p1=.15, p2=.10)
xp <- gsDesign(n.fix=np, endpoint="Binomial", delta1=.05)
gsBoundSummary(xp)
# estimate treatment effect at lower bound
```

```
# by setting delta0=0 (default) and delta1 above in gsDesign
# treatment effect at bounds is scaled to these differences
# in this case, this is the difference in event rates
gsDelta(z=xp$lower$bound, i=1:3, xp)
# binomial endpoint with risk ratio estimates
xrr <- gsDesign(n.fix=np, endpoint="Binomial", delta1=log(2/3))
gsRR(z=xp$lower$bound, i=1:3, xrr)
plot(xrr,plottype="RR")
```

gsCP

2.4: Conditional and Predictive Power, Overall and Conditional Probability of Success

Description

gsCP() computes conditional boundary crossing probabilities at future planned analyses for a given group sequential design assuming an interim z-statistic at a specified interim analysis. gsPP() averages conditional power across a posterior distribution to compute predictive power. gsPI() computes Bayesian prediction intervals for future analyses corresponding to results produced by gsPP(). gsPosterior() computes the posterior density for the group sequential design parameter of interest given a prior density and an interim outcome that is exact or in an interval. gsPOS() computes the probability of success for a trial using a prior distribution to average power over a set of theta values of interest. gsCPOS() assumes no boundary has been crossed before and including an interim analysis of interest, and computes the probability of success based on this event. Note that gsCP() and gsPP() take only the interim test statistic into account in computing conditional probabilities, while gsCPOS() conditions on not crossing any bound through a specified interim analysis.

Usage

```
gsCP(x, theta=NULL, i=1, zi=0, r=18)
gsPP(x, i=1, zi=0, theta=c(0,3), wgts=c(.5,.5), r=18, total=TRUE)
gsPI(x, i=1, zi=0, j=2, level=.95, theta=c(0,3), wgts=c(.5,.5))
gsPosterior(x=gsDesign(), i=1, zi=NULL, prior=normalGrid(), r=18)
gsPOS(x, theta, wgts)
gsCPOS(i, x, theta, wgts)
```

Arguments

x	An object of type gsDesign or gsProbability
theta	a vector with θ value(s) at which conditional power is to be computed; for gsCP() if NULL, an estimated value of θ based on the interim test statistic ($z_i/\sqrt{x\$n.I[i]}$) as well as at $x\$theta$ is computed. For gsPosterior, this may be a scalar or an interval; for gsPP and gsCP, this must be a scalar.
wgts	Weights to be used with grid points in theta. Length can be one if weights are equal, otherwise should be the same length as theta. Values should be positive, but do not need to sum to 1.

<code>i</code>	analysis at which interim z-value is given; must be from 1 to <code>x\$k-1</code>
<code>prior</code>	provides a prior distribution in the form produced by <code>normalGrid</code>
<code>zi</code>	interim z-value at analysis <code>i</code> (scalar)
<code>j</code>	specific analysis for which prediction is being made; must be <code>>i</code> and no more than <code>x\$k</code>
<code>r</code>	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 <code>r</code> will not be changed by the user.
<code>total</code>	The default of <code>total=TRUE</code> produces the combined probability for all planned analyses after the interim analysis specified in <code>i</code> . Otherwise, information on each analysis is provided separately.
<code>level</code>	The level to be used for Bayes credible intervals (which approach confidence intervals for vague priors). The default <code>level=.95</code> corresponds to a 95% credible interval. <code>level=0</code> provides a point estimate rather than an interval.

Details

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

For `gsPP()`, `gsPI()`, `gsPOS()` and `gsCPOS()`, the prior distribution for the standardized parameter `theta ()` for a group sequential design specified through a `gsDesign` object is specified through the arguments `theta` and `wgts`. This can be a discrete or a continuous probability density function. For a discrete function, generally all weights would be 1. For a continuous density, the `wgts` would contain integration grid weights, such as those provided by `normalGrid`.

For `gsPosterior`, a prior distribution in `prior` must be composed of the vectors `z` `density`. The vector `z` contains points where the prior is evaluated and `density` the corresponding density or, for a discrete distribution, the probabilities of each point in `z`. Densities may be supplied as from `normalGrid()` where grid weights for numerical integration are supplied in `gridwgts`. If `gridwgts` are not supplied, they are defaulted to 1 (equal weighting). To ensure a proper prior distribution, you must have `sum(gridwgts * density)` equal to 1; this is NOT checked, however.

Value

`gsCP()` returns an object of the class `gsProbability`. Based on the input design and the interim test statistic, the output `gsDesign` object has bounds for test statistics computed based on solely on observations after interim `i`. Boundary crossing probabilities are computed for the input θ values. See manual and examples.

`gsPP()` if `total==TRUE`, returns a real value indicating the predictive power of the trial conditional on the interim test statistic `zi` at analysis `i`; otherwise returns vector with predictive power for each future planned analysis.

`gsPI()` returns an interval (or point estimate if `level=0`) indicating 100`level`% credible interval for the z-statistic at analysis `j` conditional on the z-statistic at analysis `i<j`. The interval does not consider intervening interim analyses. The probability estimate is based on the predictive distribution used for `gsPP()` and requires a prior distribution for the group sequential parameter `theta` specified in `theta` and `wgts`.

gsPosterior() returns a posterior distribution containing the the vector z input in prior\$z, the posterior density in density, grid weights for integrating the posterior density as input in prior\$gridwgts or defaulted to a vector of ones, and the product of the output values in density and gridwgts in wgts.

gsPOS() returns a real value indicating the probability of a positive study weighted by the prior distribution input for theta.

gsCPOS() returns a real value indicating the probability of a positive study weighted by the posterior distribution derived from the interim test statistic and the prior distribution input for theta conditional on an interim test statistic.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

Proschan, Michael A., Lan, KK Gordon and Wittes, Janet Turk (2006), *Statistical Monitoring of Clinical Trials*. NY: Springer.

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

[normalGrid](#), [gsDesign](#), [gsProbability](#), [gsBoundCP](#)

Examples

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

# set up a prior distribution for the treatment effect
# that is normal with mean .75*x$delta and standard deviation x$delta/2
mu0 <- .75*x$delta
sigma0 <- x$delta/2
prior <- normalGrid(mu=mu0, sigma=sigma0)

# compute POS for the design given the above prior distribution for theta
gsPOS(x=x, theta=prior$z, wgts=prior$wgts)

# assume POS should only count cases in prior where theta >= x$delta/2
gsPOS(x=x, theta=prior$z, wgts=prior$wgts*(prior$z>=x$delta/2))
```

```

# assuming a z-value at lower bound 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=x$lower$bound[2])
CP

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

# compute predictive probability based on above assumptions
gsPP(x, i=2, zi=x$lower$bound[2], theta=prior$z, wgts=prior$wgts)

# if it is known that boundary not crossed at interim 2, use
# gsCPOS to compute conditional POS based on this
gsCPOS(x=x, i=2, theta=prior$z, wgts=prior$wgts)

# 2-stage example to compare results to direct computation
x<-gsDesign(k=2)
z1<- 0.5
n1<-x$n.I[1]
n2<-x$n.I[2]-x$n.I[1]
thetahat<-z1/sqrt(n1)
theta<-c(thetahat, 0 , x$delta)

# conditional power direct computation - comparison w gsCP
pnorm((n2*theta+z1*sqrt(n1)-x$upper$bound[2]*sqrt(n1+n2))/sqrt(n2))

gsCP(x=x, zi=z1, i=1)$upper$prob

# predictive power direct computation - comparison w gsPP
# use same prior as above
mu0 <- .75 * x$delta * sqrt(x$n.I[2])
sigma2 <- (.5 * x$delta)^2 * x$n.I[2]
prior <- normalGrid(mu=.75 * x$delta, sigma=x$delta/2)
gsPP(x=x, zi=z1, i=1, theta=prior$z, wgts=prior$wgts)
t <- .5
z1 <- .5
b <- z1 * sqrt(t)
# direct from Proschan, Lan and Wittes eqn 3.10
# adjusted drift at n.I[2]
pnorm(((b - x$upper$bound[2]) * (1 + t * sigma2) +
(1 - t) * (mu0 + b * sigma2)) /
sqrt((1 - t) * (1 + sigma2) * (1 + t * sigma2)))

# plot prior then posterior distribution for unblinded analysis with i=1, zi=1
xp <- gsPosterior(x=x,i=1,zi=1,prior=prior)
plot(x=xp$z, y=xp$density, type="l", col=2, xlab=expression(theta), ylab="Density")
points(x=x$z, y=x$density, col=1)

```

```
# add posterior plot assuming only knowlede that interim bound has
# not been crossed at interim 1
xpb <- gsPosterior(x=x,i=1,zi=1,prior=prior)
lines(x=xpb$z,y=xpb$density,col=4)

# prediction interval based in interim 1 results
# start with point estimate, followed by 90% prediction interval
gsPI(x=x, i=1, zi=z1, j=2, theta=prior$z, wgts=prior$wgts, level=0)
gsPI(x=x, i=1, zi=z1, j=2, theta=prior$z, wgts=prior$wgts, level=.9)
```

 gsDensity

 2.6: Group sequential design interim density function

Description

Given an interim analysis i of a group sequential design and a vector of real values z_i , `gsDensity()` computes an interim density function at analysis i at the values in z_i . For each value in z_i , this interim density is the derivative of the probability that the group sequential trial does not cross a boundary prior to the i -th analysis and at the i -th analysis the interim Z-statistic is less than that value. When integrated over the real line, this density computes the probability of not crossing a bound at a previous analysis. It corresponds to the subdistribution function at analysis i that excludes the probability of crossing a bound at an earlier analysis.

The initial purpose of this routine was as a component needed to compute the predictive power for a trial given an interim result; see [gsPP](#).

Usage

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

Arguments

<code>x</code>	An object of type <code>gsDesign</code> or <code>gsProbability</code>
<code>theta</code>	a vector with θ value(s) at which the interim density function is to be computed.
<code>i</code>	analysis at which interim z-values are given; must be from 1 to <code>x\$K</code>
<code>zi</code>	interim z-value at analysis i (scalar)
<code>r</code>	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 <code>r</code> will not be changed by the user.

Details

See Jennison and Turnbull (2000) for details on how these computations are performed.

Value

zi	The input vector zi.
theta	The input vector theta.
density	A matrix with length(zi) rows and length(theta) columns. The subdensity function for z[j], theta[m] at analysis i is returned in density[j,m].

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsDesign](#), [gsProbability](#), [gsBoundCP](#)

Examples

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

# set theta values where density is to be evaluated
theta <- x$theta[2] * c(0, .5, 1, 1.5)

# set zi values from -1 to 7 where density is to be evaluated
zi <- seq(-3, 7, .05)

# compute subdensity values at analysis 2
y <- gsDensity(x, theta=theta, i=2, zi=zi)

# plot sub-density function for each theta value
plot(y$zi, y$density[,3], type="l", xlab="Z",
      ylab="Interim 2 density", lty=3, lwd=2)
lines(y$zi, y$density[,2], lty=2, lwd=2)
lines(y$zi, y$density[,1], lwd=2)
lines(y$zi, y$density[,4], lty=4, lwd=2)
title("Sub-density functions at interim analysis 2")
legend(x=c(3.85,7.2), y = c(.27,.385), lty=1:5, lwd=2, cex=1.5,
       legend=c(
         expression(paste(theta,"=0.0")),
         expression(paste(theta,"=0.5", delta)),
         expression(paste(theta,"=1.0", delta)),
```

```

expression(paste(theta,"=1.5", delta)))

# add vertical lines with lower and upper bounds at analysis 2
# to demonstrate how likely it is to continue, stop for futility
# or stop for efficacy at analysis 2 by treatment effect
lines(array(x$upper$bound[2],2), c(0,.4),col=2)
lines(array(x$lower$bound[2],2), c(0,.4), lty=2, col=2)

# Replicate part of figures 8.1 and 8.2 of Jennison and Turnbull text book
# O'Brien-Fleming design with four analyses

x <- gsDesign(k=4, test.type=2, sfu="OF", alpha=.1, beta=.2)

z <- seq(-4.2, 4.2, .05)
d <- gsDensity(x=x, theta=x$theta, i=4, zi=z)

plot(z, d$density[,1], type="l", lwd=2, ylab=expression(paste(p[4],"(z,",theta,")")))
lines(z, d$density[,2], lty=2, lwd=2)
u <- x$upper$bound[4]
text(expression(paste(theta,"=",delta)),x=2.2, y=.2, cex=1.5)
text(expression(paste(theta,"=0")),x=.55, y=.4, cex=1.5)

```

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, nFixSurv=0, endpoint=NULL, delta1=1, delta0=0)

```

```

## S3 method for class 'gsDesign'
print(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 <code>test.type=5</code> or <code>6</code> , <code>astar</code> 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 <code>astar</code> is not specified by the user.
delta	Standardized effect size for theta under alternative hypothesis. See details and examples.
n.fix	Sample size for fixed design with no interim; used to find maximum group sequential sample size. For a time-to-event outcome, input number of events required for a fixed design rather than sample size and enter fixed design sample size (optional) in <code>nFixSurv</code> . 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 <code>k</code> or <code>k-1</code> . The values should satisfy $0 < \text{timing}[1] < \text{timing}[2] < \dots < \text{timing}[k-1] < \text{timing}[k]=1$.
sfu	A spending function or a character string indicating a boundary type (that is, “WT” for Wang-Tsiatis bounds, “OF” for O’Brien-Fleming bounds and “Pocock” for Pocock bounds). For one-sided and symmetric two-sided testing is used to completely specify spending (<code>test.type=1, 2</code>), <code>sfu</code> . The default value is <code>sfHSD</code> which is a Hwang-Shih-DeCani spending function. See details, Spending function overview , manual and examples.
sfupar	Real value, default is -4 which is an O’Brien-Fleming-like conservative bound when used with the default Hwang-Shih-DeCani spending function. This is a real-vector for many spending functions. The parameter <code>sfupar</code> specifies any parameters needed for the spending function specified by <code>sfu</code> ; this will be ignored for spending functions (<code>sfLDOF</code> , <code>sfLDPocock</code>) or bound types (“OF”, “Pocock”) that do not require parameters.
sf1	Specifies the spending function for lower boundary crossing probabilities when asymmetric, two-sided testing is performed (<code>test.type = 3, 4, 5, or 6</code>). Unlike the upper bound, only spending functions are used to specify the lower bound. The default value is <code>sfHSD</code> which is a Hwang-Shih-DeCani spending function. The parameter <code>sf1</code> is ignored for one-sided testing (<code>test.type=1</code>) or symmetric 2-sided testing (<code>test.type=2</code>). See details, spending functions, manual and examples.
sf1par	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 <code>r</code> will not be changed by the user.

<code>n.I</code>	Used for re-setting bounds when timing of analyses changes from initial design; see examples.
<code>maxn.IPlan</code>	Used for re-setting bounds when timing of analyses changes from initial design; see examples.
<code>nFixSurv</code>	If a time-to-event variable is used, <code>nFixSurv</code> computed as the sample size from <code>nSurvival</code> may be entered to have <code>gsDesign</code> compute the total sample size required as well as the number of events at each analysis that will be returned in <code>n.fix</code> ; this is rounded up to an even number.
<code>x</code>	In <code>print.gsDesign</code> this is an object of class <code>gsDesign</code> .
<code>...</code>	This should allow optional arguments that are standard when calling <code>print</code> .
<code>endpoint</code>	An optional character string that should represent the type of endpoint used for the study. This may be used by output functions. Types most likely to be recognized initially are "TTE" for time-to-event outcomes with fixed design sample size generated by <code>nSurvival()</code> and "Binomial" for 2-sample binomial outcomes with fixed design sample size generated by <code>nBinomial()</code> .
<code>delta1</code>	<code>delta1</code> and <code>delta0</code> may be used to store information about the natural parameter scale compared to <code>delta</code> that is a standardized effect size. <code>delta1</code> is the alternative hypothesis parameter value on the natural scale (e.g., the difference in two binomial rates).
<code>delta0</code>	The parameter value under the null hypothesis on the natural parameter scale. Default is 0, which might normally be interpreted as no difference between two treatment groups. If non-zero, this would normally represent a non-inferiority margin.

Details

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

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

`delta` and `n.fix` are used together to determine what sample size output options the user seeks. The default, `delta=0` and `n.fix=1`, results in a 'generic' design that may be used with any sampling situation. Sample size ratios are provided and the user multiplies these times the sample size for a fixed design to obtain the corresponding group sequential analysis times. If `delta>0`, `n.fix` is ignored, and `delta` is taken as the standardized effect size - the signal to noise ratio for a single observation; for example, the mean divided by the standard deviation for a one-sample normal problem. In this case, the sample size at each analysis is computed. When `delta=0` and `n.fix>1`, `n.fix` is assumed to be the sample size for a fixed design with no interim analyses. See examples below.

Following are further comments on the input argument `test.type` which is used to control what type of error measurements are used in trial design. The manual may also be worth some review in

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

Value

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

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

```

maxn.IPlan      As input.
endpoint        As input.
delta1          As input.
delta0          As input.

```

Note: `print.gsProbability()` returns the input `x`.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsDesign package overview](#), [Plots for group sequential designs](#), [gsProbability](#), [Spending function overview](#), [Wang-Tsiatis Bounds](#)

Examples

```

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

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

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

```

gsDesign package overview

1.0 Group Sequential Design

Description

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

Details

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

Index:

gsDesign	2.1: Design Derivation
gsProbability	2.2: Boundary Crossing Probabilities
plot.gsDesign	2.3: Plots for group sequential designs
gsCP	2.4: Conditional Power Computation
gsBoundCP	2.5: Conditional Power at Interim Boundaries
gsbound	2.6: Boundary derivation - low level
normalGrid	3.1: Normal Density Grid
binomial	3.2: Testing, Confidence Intervals and Sample Size for Comparing Two Binomial Rates
Survival sample size	3.3: Time-to-event sample size calculation (Lachin-Foulkes)
Spending function overview	4.0: Spending functions
sfHSD	4.1: Hwang-Shih-DeCani Spending Function
sfPower	4.2: Kim-DeMets (power) Spending Function
sfExponential	4.3: Exponential Spending Function
sfLDPocock	4.4: Lan-DeMets Spending function overview
sfPoints	4.5: Pointwise Spending Function
sfLogistic	4.6: 2-parameter Spending Function Families
sfTDist	4.7: t-distribution Spending Function
Wang-Tsiatis Bounds	5.0: Wang-Tsiatis Bounds

checkScalar 6.0: Utility functions to verify variable properties

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

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

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

Author(s)

Keaven Anderson

Maintainer: Keaven Anderson <keaven_anderson@merck.com>

References

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

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

See Also

[gsDesign](#), [gsProbability](#)

Examples

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

Description

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)
## S3 method for class 'gsProbability'
print(x,...)
```

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 <code>gsDesign</code> returned by a call to <code>gsDesign()</code> . 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.
x	An item of class <code>gsProbability</code> .
...	Not implemented (here for compatibility with generic print input).

Details

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

Value

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

Note: `print.gsProbability()` returns the input x .

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Plots for group sequential designs, gsDesign, gsDesign package overview](#)

Examples

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

# take a look at it
x

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

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

```

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

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

# the plot does not change from before since this is a
# gsDesign object; note that theta/delta is on x axis
plot(y, plotype=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 plotype is now 2
# this is the same range for theta, but plot now has theta on x axis
plot(z)

```

nNormal

Normal distribution sample size (2-sample)

Description

nNormal() computes a fixed design sample size for comparing 2 means where variance is known. The function allows computation of sample size for a non-inferiority hypothesis. Note that you may wish to investigate other R packages such as the pwr package which uses the t-distr

Usage

```
nNormal(delta1=1, sigma=1.7, sigalt=NULL, alpha=.025, beta=.1, ratio=1, sided=1,
n=NULL, delta0=0)
```

Arguments

delta1	difference between sample means under the alternate hypothesis.
delta0	difference between sample means under the null hypothesis; normally this will be left as the default of 0.
ratio	randomization ratio of experimental group compared to control.
sided	1 for 1-sided test (default), 2 for 2-sided test.
sigma	Standard deviation for the control arm.

sigalt	Standard deviation of experimental arm; this will be set to be the same as the control arm with the default of NULL.
alpha	type I error rate. Default is 0.025 since 1-sided testing is default.
beta	type II error rate. Default is 0.10 (90% power). Not needed if n is provided.
n	Sample size; may be input to compute power rather than sample size. If NULL (default) then sample size is computed.

Details

nNormal() computes sample size for comparing two normal means when the variance for observations in

Value

If n is NULL (default), total sample size (2 arms combined) is computed. Otherwise, power is c

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

Lachin JM (1981), Introduction to sample size determination and power analysis for clinical trials. *Controlled Clinical Trials* 2:93-113.

Snedecor GW and Cochran WG (1989), Statistical Methods. 8th ed. Ames, IA: Iowa State University Press.

See Also

[gsDesign package overview](#)

Examples

```
# EXAMPLES
# equal variances
nNormal(delta1=.5,sigma=1.1,alpha=.025,beta=.2)
# unequal variances
nNormal(delta1=.5,sigma=1.1,sigalt=2,alpha=.025,beta=.2)
# unequal sample sizes
nNormal(delta1=.5,sigma=1.1,alpha=.025,beta=.2, ratio=2)
# non-inferiority assuming a better effect than null
nNormal(delta1=.5,delta0=-.1,sigma=1.2)
```

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.

Details

This is a utility function to provide a normal density function and a grid to integrate over as described by Jennison and Turnbull (2000), Chapter 19. While integration can be performed over the real line or over any portion of it, the numerical integration does not extend beyond 6 standard deviations from the mean. The grid used for integration uses equally spaced points over the middle of the distribution function, and spreads points further apart in the tails. The values returned in gridwghts may be used to integrate any function over the given grid, although the user should take care that the function integrated is not large in the tails of the grid where points are spread further apart.

Value

z	Grid points for numerical integration.
density	The standard normal density function evaluated at the values in z; see examples.
gridwghts	Simpson's rule weights for numerical integration on the grid in z; see examples.
wgts	Weights to be used with the grid in z for integrating the normal density function; see examples. This is equal to density * gridwghts.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

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

Examples

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

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

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

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

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

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

# this can also be done by combining gridwgts and density
sqrt(sum((x$z - 2)^2 * x$gridwgts * x$density))

# integrate normal density and compare to built-in function
# to compute probability of being within 1 standard deviation
# of the mean
pnorm(1)-pnorm(-1)
x <- normalGrid(bounds=c(-1, 1))
sum(x$wgts)
sum(x$gridwgts * x$density)

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

# set a prior distribution for theta
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\n prior distribution for theta with \n 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")
lines(y$z, z$en)

```

nSurvival

3.4: Time-to-event sample size calculation (Lachin-Foulkes)

Description

nSurvival() is used to calculate the sample size for a clinical trial with a time-to-event endpoint. The Lachin and Foulkes (1986) method is used. nEvents uses the Schoenfeld (1981) approximation to provide sample size and power in terms of the underlying hazard ratio and the number of events observed in a survival analysis. The functions hrz2n(), hrn2z() and zn2hr() also use the Schoenfeld approximation to provide simple translations between hazard ratios, z-values and the number of events in an analysis; input variables can be given as vectors.

Usage

```

nSurvival(lambda1=1/12, lambda2=1/24, Ts=24, Tr=12, eta = 0, ratio = 1,
          alpha = 0.025, beta = 0.10, sided = 1, approx = FALSE,
          type = c("rr", "rd"), entry = c("unif", "expo"), gamma = NA)
## S3 method for class 'nSurvival'
print(x,...)
nEvents(hr = .6, alpha = .025, beta = .1, ratio = 1, sided = 1, hr0 = 1, n = 0, tbl = FALSE)
hrn2z(hr, n, ratio=1)
hrz2n(hr, z, ratio=1)
zn2hr(z, n, ratio=1)

```

Arguments

lambda1, lambda2	event hazard rate for placebo and treatment group respectively.
eta	equal dropout hazard rate for both groups.
ratio	randomization ratio between placebo and treatment group. Default is balanced design, i.e., randomization ratio is 1.
Ts	maximum study duration.
Tr	accrual (recruitment) duration.
alpha	type I error rate. Default is 0.025 since 1-sided testing is default.
beta	type II error rate. Default is 0.10 (90% power). Not needed for nEvents() if n is provided.
sided	one or two-sided test? Default is one-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).
x	An object of class "nSurvival" returned by nSurvival() (optional: used for output; "months" or "years" would be the 'usual' choices).
hr	Hazard ratio. For nEvents, this is the hazard ratio under the alternative hypothesis (>0).
hr0	Hazard ratio under the null hypothesis (>0, != hr).
n	Number of events. For nEvents may be input to compute power rather than sample size.
tbl	Indicator of whether or not scalar (vector) or tabular output is desired for nEvents().
z	A z-statistic.
...	Allows additional arguments for print.nSurvival().

Details

nSurvival() produces an object of class "nSurvival" with the number of subjects and events for a set of pre-specified trial parameters, such as accrual duration and follow-up period. The calculation is based on Lachin and Foulkes (1986) method and can be used for risk ratio or risk difference. The function also consider non-uniform (exponential) 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.

nEvents() uses the Schoenfeld (1981) method to approximate the number of events n (given beta) or the power (given n). Arguments may be vectors or scalars, but any vectors must have the same length.

The functions hrz2n, hrn2z and zn2hr also all apply the Schoenfeld approximation for proportional hazards modelling. This approximation is based on the asymptotic normal distribution of the logrank statistic as well as related statistics are asymptotically normal. Let λ denote the underlying hazard ratio (λ_1/λ_2 in terms of the arguments to nSurvival). Further, let n denote the number of events observed when computing the statistic of interest and r the ratio of the sample size in an experimental group relative to a control. The estimated natural logarithm of the hazard ratio from a proportional hazards ratio is approximately normal with a mean of $\log\lambda$ and variance $(1+r)^2/nr$. Let z denote a logrank statistic (or a Wald statistic or score statistic from a proportional hazards regression model). The same asymptotic theory implies z is asymptotically equivalent to a normalized estimate of the hazard ratio λ and thus z is asymptotically normal with variance 1 and mean

$$\frac{\log\lambda r}{(1+r)^2}.$$

Plugging the estimated hazard ratio into the above equation allows approximating any one of the following based on the other two: the estimate hazard ratio, the number of events and the z-statistic. That is,

$$\hat{\lambda} = \exp(z(1+r)/\sqrt{rn})$$

$$z = \log(\hat{\lambda})\sqrt{nr}/(1+r)$$

$$n = (z(1+r)/\log(\hat{\lambda}))^2/r.$$

Value

nSurvival produces a list with the following component returned:

type	As input.
entry	As input.
n	Sample size required (computed).
nEvents	Number of events required (computed).
lambda1	As input.
lambda2	As input.
eta	As input.
ratio	As input.
gamma	As input.
alpha	As input.
beta	As input.
sided	As input.
Ts	As input.
Tr	As input.

nEvents produces a scalar or vector of sample sizes (or powers) when tbl=FALSE or, when tbl=TRUE a matrix of values with the following columns:

hr	As input.
n	If n[1]=0 on input (default), output contains the number of events need to obtain the input Type I and II error. If n[1]>0 on input, the input value is returned.
alpha	As input.
beta	If n[1]=0 on input (default), beta is output as input. Otherwise, this is the computed Type II error based on the input n.
Power	One minus the output beta. When tbl=FALSE, n[1]>0, this is the value or vector of values returned.
delta	Standardized effect size represented by input difference between null and alternative hypothesis hazard ratios.
ratio	Ratio of experimental to control sample size where 'experimental' is the same as the group with hazard represented in the numerator of the hazard ratio.
se	Estimated standard error for the observed log(hazard ratio) with the given sample size.

hrz2n outputs a number of events required to approximately have the input hazard ratio, z-statistic and sample size correspond. hrn2z outputs an approximate z-statistic corresponding to an input hazard ratio and number of events. zn2hr outputs an approximate hazard ratio corresponding to an input z-statistic and number of events.

Author(s)

Shanhong Guan <shanhong.guan@gmail.com>, Keaven Anderson <keaven_anderson@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.

Schoenfeld D (1981), The Asymptotic Properties of Nonparametric Tests for Comparing Survival Distributions. *Biometrika*, 68, 316-319.

See Also

[gsDesign package overview](#), [Plots for group sequential designs](#), [gsDesign](#), [gsHR](#)

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 (1-sided)
# power = 0.9 (default beta=.1)

ss <- nSurvival(lambda1=.2 , lambda2=.1, eta = .1, Ts = 2, Tr = .5,
               sided=1, alpha=.025)

ss

# symmetric, 2-sided design with O'Brien-Fleming-like boundaries
# sample size is computed based on a fixed design requiring n=100
x<-gsDesign(k = 5, test.type = 2, n.fix=ss$nEvents, nFixSurv=ss$n)
# boundary plot
plot(x)
# effect size plot
plot(x, plotype = "hr")
# total sample size
  x$nSurv
# number of events at analyses
  x$n.I
# print the design
x

# approximate number of events required using Schoenfeld's method
# for 2 different hazard ratios
nEvents(hr=c(.5, .6), tbl=TRUE)
# vector output
nEvents(hr=c(.5, .6))

# approximate power using Schoenfeld's method
```

```

# given 2 sample sizes and hr=.6
nEvents(hr=.6, n=c(50, 100), tbl=TRUE)
# vector output
nEvents(hr=.6, n=c(50, 100))

# approximate hazard ratio corresponding to 100 events and z-statistic of 2
zn2hr(n=100,z=2)

# approximate number of events corresponding to z-statistic of 2 and
# estimated hazard ratio of .5 (or 2)
hrz2n(hr=.5,z=2)
hrz2n(hr=2,z=2)

# approximate z statistic corresponding to 75 events
# and estimated hazard ratio of .6 (or 1/.6)
# assuming 2-to-1 randomization of experimental to control
hrn2z(hr=.6,n=75,ratio=2)
hrn2z(hr=1/.6,n=75,ratio=2)

```

plot.gsDesign

2.3: Plots for group sequential designs

Description

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

Usage

```

## S3 method for class 'gsProbability'
plot(x, plottype=2, base=FALSE, ...)
## S3 method for class 'gsDesign'
plot(x, plottype=1, base=FALSE, ...)

```

Arguments

<code>x</code>	Object of class <code>gsDesign</code> for <code>plot.gsDesign()</code> or <code>gsProbability</code> for <code>plot.gsProbability()</code> .
<code>plottype</code>	1=boundary plot (default for <code>gsDesign</code>), 2=power plot (default for <code>gsProbability</code>), 3=estimated treatment effect at boundaries, 4=conditional power at boundaries, 5=spending function plot (only available if <code>class(x)=="gsDesign"</code>),

6=expected sample size plot, and
 7=B-values at boundaries.

Character values for plot type 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.

base Default is FALSE, which means ggplot2 graphics are used. If true, base graphics are used for plotting.

... This allows many optional arguments that are standard when calling plot. Other arguments include:

theta which is used for plottype=2, 4, 6; normally defaults will be adequate; see details.

ses=TRUE which applies only when plottype=3 and

class(x)="gsDesign"; indicates that estimated standardized effect size at the boundary is to be plotted rather than the actual estimate.

xval="Default" which is only effective when plottype=2 or 6. Appropriately scaled (reparameterized) values for x-axis for power and expected sample size graphs; see details.

Details

The intent is that many standard plot() parameters will function as expected; exceptions to this rule exist. In particular, main, xlab, ylab, lty, col, lwd, type, pch, cex have been tested and work for most values of plottype; one exception is that type="1" cannot be overridden when plottype=2. Default values for labels depend on plottype and the class of x.

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

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

Spending functions are plotted for a continuous set of values from 0 to 1. This option should not be used if a boundary is used or a pointwise spending function is used (sfu or sf1="WT", "OF", "Pocock" or sfPoints).

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

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

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

Value

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

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

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

See Also

[gsDesign package overview](#), [gsDesign](#), [gsProbability](#)

Examples

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

# the following translate to calls to plot.gsDesign since x was
# returned by gsDesign; run these commands one at a time
plot(x)
plot(x, plottype=2)
plot(x, plottype=3)
plot(x, plottype=4)
plot(x, plottype=5)
plot(x, plottype=6)
plot(x, plottype=7)

# choose different parameter values for power plot
```

```
# start with design in x from above
y <- gsProbability(k=5, theta=seq(0, .5, .025), x$n.I,
                  x$lower$bound, x$upper$bound)

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

sfExponential

4.3: Exponential Spending Function

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

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

Value

An object of type spendfn.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

- Anderson KM and Clark JB (2009), Fitting spending functions. *Statistics in Medicine*; 29:321-327.
- Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.
- Lan, KKG and DeMets, DL (1983), Discrete sequential boundaries for clinical trials. *Biometrika*; 70:659-663.

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

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

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

# plot exponential spending function across a range of values of interest
t <- 0:100/100
plot(t, sfExponential(0.025, t, 0.8)$spend,
     xlab="Proportion of final sample size",
```

```

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

```

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; allowable range is $[-40, 40]$

Details

A Hwang-Shih-DeCani spending function takes the form

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

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

Value

An object of type `spendfn`. See [Spending function overview](#) for further details.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

# print the design
x

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

# print again
x

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

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 to the standard for spending functions used with <code>gsDesign()</code> .

Details

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

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

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

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

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

Value

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

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

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

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

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

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

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

# use 'best' exponential approximation for k=6; see manual for details
gsDesign(k=6, sfu=sfExponential, sfupar=0.7849295,
```

```
test.type=2)$upper$bound
```

sfLinear

4.6: Piecewise Linear Spending Function

Description

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

Usage

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

Arguments

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

Value

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

# now do an example where there is no lower-spending at interim 1
# and no upper spending at interim 2
sflpar<-c(1/3,2/3,0,.25)
sfupar<-c(1/3,2/3,.1,.1)
x <- gsDesign(sfu=sfLinear, sfl=sfLinear, sfupar=sfupar, sflpar=sflpar)
plot(x, plottype="sf")
x
```

sfLogistic

4.7: Two-parameter Spending Function Families

Description

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

Usage

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

Arguments

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

Details

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

The other spending functions take the form

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

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

For the logistic spending function this simplifies to

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

For the extreme value distribution with

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

this simplifies to

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

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

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

Value

An object of type spendfn. See [Spending function overview](#) for further details.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

# print the design
x

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

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

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

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

# set up a function to plot comparisons of all
# 2-parameter spending functions
plotsf <- function(alpha, t, param)
{
```

```

plot(t, sfCauchy(alpha, t, param)$spend,
     xlab="Proportion of enrollment",
     ylab="Cumulative spending", type="l", lty=2)
lines(t, sfExtremeValue(alpha, t, param)$spend, lty=5)
lines(t, sfLogistic(alpha, t, param)$spend, lty=1)
lines(t, sfNormal(alpha, t, param)$spend, lty=3)
lines(t, sfExtremeValue2(alpha, t, param)$spend, lty=6, col=2)
lines(t, sfBetaDist(alpha, t, param)$spend, lty=7, col=3)
legend(x=c(.05, .475), y=.025*c(.55, .9),
       lty=c(1, 2, 3, 5, 6, 7),
       col=c(1, 1, 1, 1, 2, 3),
       legend=c("Logistic", "Cauchy", "Normal", "Extreme value",
               "Extreme value 2", "Beta distribution"))
}
# do comparison for a design with conservative early spending
# note that Cauchy spending function is quite different
# from the others
param <- c(.25, .5, .05, .1)
plotsf(.025, t, param)

```

sfPoints

4.5: Pointwise Spending Function

Description

The function `sfPoints` implements a spending function with values specified for an arbitrary set of specified points. It is now recommended to use `sfLinear` rather than `sfPoints`. Normally `sfPoints` will be passed to `gsDesign` in the parameter `sfu` for the upper bound or `sf1` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence, just the points they wish to specify. If using `sfPoints()` in a design, it is recommended to specify how to interpolate between the specified points (e.g., linear interpolation); also consider fitting smooth spending functions; see [Spending function overview](#).

Usage

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

Arguments

alpha	Real value > 0 and no more than 1. Normally, $\alpha=0.025$ for one-sided Type I error specification or $\alpha=0.1$ for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size/information.
t	A vector of points with increasing values from >0 and ≤ 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 <code>t</code> specifying the cumulative proportion of spending to corresponding to each point in <code>t</code> ; must be ≥ 0 and ≤ 1 .

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/gsdDesignManual.pdf` in the directory where R is installed.

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#), [sfLogistic](#)

Examples

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

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

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

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

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

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

```
# example without lower spending at initial interim or
# upper spending at last interim
x <- gsDesign(k=3, sfu=sfPoints, sfupar=c(.25, .25),
             sfl=sfPoints, sflpar=c(0,.25))
x
```

sfPower

4.2: Kim-DeMets (power) Spending Function

Description

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

Usage

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

Arguments

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

Details

A Kim-DeMets spending function takes the form

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

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

Value

An object of type `spendfn`. See [Spending function overview](#) for further details.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

# print the design
x

# plot the spending function using many points to obtain a smooth curve
# show rho=3 for approximation to O'Brien-Fleming and rho=.75 for
# approximation to Pocock design.
# Also show rho=2 for an intermediate spending.
# Compare these to Hwang-Shih-DeCani spending with gamma=-4, -2, 1
t <- 0:100/100
plot(t, sfPower(0.025, t, 3)$spend, xlab="Proportion of sample size",
     ylab="Cumulative Type I error spending",
     main="Kim-DeMets (rho) versus Hwang-Shih-DeCani (gamma) Spending",
     type="l", cex.main=.9)
lines(t, sfPower(0.025, t, 2)$spend, lty=2)
lines(t, sfPower(0.025, t, 0.75)$spend, lty=3)
lines(t, sfHSD(0.025, t, 1)$spend, lty=3, col=2)
lines(t, sfHSD(0.025, t, -2)$spend, lty=2, col=2)
lines(t, sfHSD(0.025, t, -4)$spend, lty=1, col=2)
legend(x=c(.0, .375), y=.025*c(.65, 1), lty=1:3,
       legend=c("rho= 3", "rho= 2", "rho= 0.75"))
legend(x=c(.0, .357), y=.025*c(.65, .85), lty=1:3, bty="n", col=2,
       legend=c("gamma= -4", "gamma= -2", "gamma=1"))
```

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 `sf1` 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 parameter (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 <code>gsDesign()</code> is called with a t -distribution spending function, this is the parameterization printed. The five parameter specification is $c(t_1, t_2, u_1, u_2, df)$ where the objective is that the resulting cumulative proportion of spending at t represented by $sf(t)$ satisfies $sf(t_1)=\alpha*u_1$, $sf(t_2)=\alpha*u_2$. The t -distribution used has df degrees of freedom. In this parameterization, all the first four values must be between 0 and 1 and $t_1 < t_2$, $u_1 < u_2$. 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 <code>param</code> is $c(t_1, t_2, t_3, u_1, u_2, u_3)$ where the objective is that $sf(t_1)=\alpha*u_1$, $sf(t_2)=\alpha*u_2$, and $sf(t_3)=\alpha*u_3$. See examples to see what happens when points are specified that cannot be fit.

Details

The t -distribution spending function takes the form

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

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

Value

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

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

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

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

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

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

```

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

```

sfTruncated

4.7a: Truncated spending functions

Description

The function `sfTruncated()` applies any other spending function over a restricted range. This allows eliminating spending for early interim analyses when you desire not to stop for the bound being specified. The truncation can come late in the trial. if you desire to stop a trial any time after, say, 90 percent of information is available and an analysis is performed.

Usage

```
sfTruncated(alpha, t, param)
```

Arguments

alpha	Real value > 0 and no more than 1. Normally, $\alpha=0.025$ for one-sided Type I error specification or $\alpha=0.1$ for Type II error specification. However, this could be set to 1 if for descriptive purposes you wish to see the proportion of spending as a function of the proportion of sample size or information.
t	A vector of points with increasing values from 0 to 1, inclusive. Values of the proportion of sample size or information for which the spending function will be computed.
param	a list containing the elements <code>sf</code> (a <code>spendfn</code> object such as <code>sfHSD</code>), <code>trange</code> (the range over which the spending function increases from 0 to 1; $0 \leq \text{trange}[1] < \text{trange}[2] \leq 1$), and <code>param</code> (null for a spending function with no parameters or a scalar or vector of parameters needed to fully specify the spending function in <code>sf</code>).

Details

sfTruncated compresses spending into a sub-range of [0,1]. The parameter param\$trange specifies the range over which spending is to occur. Within this range, spending is spent according to the spending function specified in param\$sf along with the corresponding spending function parameter(s) in param\$param. See example using sfLinear that spends uniformly over specified range.

Value

An object of type spendfn. See [Spending function overview](#) for further details.

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [gsDesign package overview](#)

Examples

```
# Eliminate spending for any interim at or before 20 percent of information.
# Complete spending at first interim at or after 80 percent of information.
s<-sfLinear(alpha=.05,t=(0:100)/100,param=c(.5,.5))
plot((0:100)/100,s$spend,type="l",main="Accelerating spending with sfTruncated",
xlab="Proportion of information",ylab="Spending")
s<-sfTruncated(alpha=.05,t=(0:100)/100,param=list(sf=sfLinear,trange=c(.2,.8),param=c(.5,.5)))
lines(col=2,(0:100)/100,s$spend)
text("Accelerated (red) spending over interval (.2,.8)",x=.03,y=.045,pos=4)

# now apply this in gsDesign
# note how sfupar is set up to do as above

# 1st version produces an error next to last interim must be before final spend
# x<-gsDesign(k=5, sfu=sfTruncated, sfupar=list(sf=sfLinear, param=c(.5,.5),
# trange=c(.2,.8)))

# now final spend occurs at > next to last interim
x<-gsDesign(k=5, sfu=sfTruncated, sfupar=list(sf=sfLinear, param=c(.5,.5), trange=c(.2,.95)))
x
```

```
# The above means if final analysis is done a little early, all spending can occur
# Suppose we skip 4th interim due to fast enrollment and set calendar date
# based on estimated full information, but come up with only 97 pct of plan
xA <- gsDesign(k=x$k-1,n.I=c(x$n.I[1:3],.97*x$n.I[5]),test.type=x$test.type,
              maxn.IPlan=x$n.I[x$k],sfu=sfTruncated,
              sfupar=list(sf=sfLinear, param=c(.5,.5), trange=c(.2,.95)))
xA
```

Spending functions 4.0: *Spending function overview*

Description

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

Usage

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

Arguments

alpha	Real value > 0 and no more than 1. Defaults in calls to <code>gsDesign()</code> 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 `sf1` for the lower bound to specify a spending function family for a design. In this case, the user does not need to know the calling sequence - only how to specify the parameter(s) for the spending function. The calling sequence is useful when the user wishes to plot a spending function as demonstrated below in examples. In addition to using supplied spending functions, a user can write code for a spending function. See examples.

Value

An object of type `spendfn`.

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

Note

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

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[gsDesign](#), [sfHSD](#), [sfPower](#), [sfLogistic](#), [sfExponential](#), [sfTruncated](#), [Wang-Tsiatis Bounds](#), [gsDesign package overview](#)

Examples

```
# Example 1: simple example showing what most 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, plotype=5)

# Example 2: advance example: writing a new spending function
```

```

# Most users may ignore this!

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

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

  class(x) <- "spendfn"

  x
}

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

```

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, $i = 1, 2, \dots, k$.

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 $i = 1, 2, \dots, k$ 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/gsdDesignManual.pdf` in the directory where R is installed.

Author(s)

Keaven Anderson <keaven_anderson@merck.>

References

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

See Also

[Spending function overview](#), [gsDesign](#), [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)
```

Index

*Topic **design**

- Binomial, 2
- gsBinomialExact, 9
- gsBound, 11
- gsBoundCP, 13
- gsCP, 18
- gsDensity, 22
- gsDesign, 24
- gsDesign package overview, 29
- gsProbability, 31
- nNormal, 33
- normalGrid, 35
- nSurvival, 37
- plot.gsDesign, 41
- sfExponential, 44
- sfHSD, 46
- sfLDOF, 48
- sfLinear, 50
- sfLogistic, 51
- sfPoints, 54
- sfPower, 56
- sfTDist, 58
- sfTruncated, 60
- Spending functions, 62
- Wang-Tsiatis Bounds, 64

*Topic **programming**

- checkScalar, 7

Binomial, 2

- checkLengths, 7
- checkLengths (checkScalar), 7
- checkRange, 7
- checkRange (checkScalar), 7
- checkScalar, 7, 7
- checkVector, 7
- checkVector (checkScalar), 7
- ciBinomial (Binomial), 2

gsBinomialExact, 9

- gsBound, 11
- gsBound1 (gsBound), 11
- gsBoundCP, 13, 20, 23
- gsBoundSummary, 15
- gsBValue (gsBoundSummary), 15
- gsCP, 14, 18
- gsCPOS (gsCP), 18
- gsCPz (gsBoundSummary), 15
- gsDelta (gsBoundSummary), 15
- gsDensity, 22
- gsDesign, 12, 14, 17, 20, 23, 24, 30–32, 40, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65
- gsDesign package overview, 12, 28, 29, 32, 34, 40, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63
- gsHR, 40
- gsHR (gsBoundSummary), 15
- gsPI (gsCP), 18
- gsPOS (gsCP), 18
- gsPosterior (gsCP), 18
- gsPP, 22
- gsPP (gsCP), 18
- gsProbability, 10, 12, 14, 17, 20, 23, 28, 30, 31, 43, 65
- gsRR (gsBoundSummary), 15

hrn2z (nSurvival), 37

hrz2n (nSurvival), 37

is.integer, 8

isInteger (checkScalar), 7

nBinomial (Binomial), 2

nEvents (nSurvival), 37

nNormal, 33

normalGrid, 19, 20, 35

nSurvival, 37

O'Brien-Fleming Bounds (Wang-Tsiatis Bounds), 64

plot.gsDesign, 41
plot.gsProbability (plot.gsDesign), 41
Plots for group sequential designs, 17,
28, 32, 40
Plots for group sequential designs
(plot.gsDesign), 41
Pocock Bounds (Wang-Tsiatis Bounds), 64
print.gsBinomialExact
(gsBinomialExact), 9
print.gsDesign (gsDesign), 24
print.gsProbability (gsProbability), 31
print.nSurvival (nSurvival), 37

sfBetaDist (sfLogistic), 51
sfCauchy (sfLogistic), 51
sfExponential, 44, 46, 48, 63
sfExtremeValue (sfLogistic), 51
sfExtremeValue2 (sfLogistic), 51
sfHSD, 46, 63
sfLDOF, 48
sfLDPocock (sfLDOF), 48
sfLinear, 50
sfLogistic, 51, 55, 63
sfNormal (sfLogistic), 51
sfPoints, 54
sfPower, 56, 63
sfTDist, 58
sfTruncated, 60, 63
simBinomial (Binomial), 2
Spending function overview, 25, 27, 28,
45, 47, 49–57, 59, 61, 65
Spending function overview (Spending
functions), 62
Spending functions, 62
spendingFunction (Spending functions),
62
Survival sample size (nSurvival), 37

testBinomial (Binomial), 2

Wang-Tsiatis Bounds, 28, 63, 64

xtable.gsDesign (gsBoundSummary), 15

zn2hr (nSurvival), 37