Package 'seqmon'

September 4, 2020

Type Package

Title Group Sequential Design Class for Clinical Trials

Version 2.4

Date 2020-08-26

Author David A Schoenfeld and Hui Zheng

Maintainer Hui Zheng <hzheng1@mgh.harvard.edu>

Description S4 class object for creating and managing group sequential designs. It calculates the efficacy and futility boundaries at each look. It allows modifying the design and tracking the design update history.

License MIT + file LICENSE

Depends methods

Suggests

NeedsCompilation no

Repository CRAN

Date/Publication 2020-09-04 21:20:02 UTC

R topics documented:

`seqmon-package` *seqmon*

Description

a package for creating, monitoring and modifying a group sequential design

Details

The DESCRIPTION file: DESCRIPTION

Author(s)

David A Schoenfeld, PhD and Hui Zheng, PhD

References

Proschan, MA, Lan, KKG, Wittes, JT, Statistical Monitoring of Clinical Trials: A Unified Approach, Springer, 2006

Schoenfeld DA, "A Simple Algorithm for Designing Group Sequential Clinical Trials", Biometrics. 2001 Sep;57(3):972-4.

Examples

```
design1<-sequential.design()
design1<-calcBoundaries(design1)
print(design1)
summary(design1)
```

alphaspend *Function that calculates the upper boundaries for efficacy*

Description

Calculates the upper boundaries for efficacy at each look time

Usage

```
alphaspend(levels, t, int = rep(500, length(t)), tol = 0.005)
```

Arguments

levels	The cumulative alpha spending at each look time
t	Normalized look times
int	The number of intervals the solution space is partitioned into
tol	Tolerance of the solution using uniroot

Value

numeric

Examples

```
f<- function(t) 0.025*t^4
t<-c(0.33,0.67,1)
cum_probs<-f(t)
alphaspend(levels=cum_probs,t,int=rep(500, length(t)),tol=0.005)
```

alphaspendf *The default alpha spending function*

Description

The default alpha spending function

Usage

```
alphaspendf(t)
```

Arguments

t	The normalized look times

Value

numeric

Examples

```
t<-c(0.33,0.67,1)
alphas<-alphaspendf(t)

## The function is currently defined as
function (t)
0.025 * t^4
```

---

betaspend *Function that calculates the lower boundaries for futility*

Description

Calculates the lower boundaries for futility at each look

Usage

```
betaspend(levels, upperboundary, t, int = rep(500, length(t)), noncent, tol = 0.005)
```

Arguments

levels	The cumulative beta spending at each look time
upperboundary	The upper efficacy boundaries at each look
t	Normalized look times
int	The numbers of intervals the solution space is partitioned into
noncent	The noncentrality parameter
tol	Tolerance of the solution using uniroot

Value

numeric

Examples

```
f<- function(t) 0.025*t^4
g<- function(t) 0.15*t^3
t<-c(0.33,0.67,1)
cum_alphas<-f(t)
cum_betas<-g(t)
noncent<-qnorm(0.975)+qnorm(0.85)
upper_boundaries<-alphaspend(cum_alphas,t,int=rep(500, length(t)),tol=0.005)
lower_boundaries<-betaspend(cum_betas, upper_boundaries, t, int = rep(500,3), noncent, tol = 0.005)
```

betaspendf *The default beta spending function*

Description

The default beta spending function

Usage

```
betaspendf(t)
```

Arguments

t	The normalized look times

Value

numeric

Examples

```
t<-c(0.33,0.67,1)
betas<-betaspendf(t)

## The function is currently defined as
function (t)
0.15 * t^3
```

calcBoundaries *Function for calculating the efficacy and futility boundaries*

Description

Calculates the efficacy and futility boundaries. This only needs to be done once for a new design.

Usage

```
calcBoundaries(theObject)
```

Arguments

theObject	The sequential design object

Value

numeric

Examples

```
design1<-sequential.design()
design1<-calcBoundaries(design1)
design1@lower.boundary
design1@upper.boundary
```

`curtail` *Generic function that calculates the probability to declare efficacy at the end of study given the Z value at the current look*

Description

Calculates the probability to declare efficacy at the end of study given the Z value at the current look

Usage

```
curtail(lower.boundary,upper.boundary,look,t,noncen,current=lower.boundary[look])
```

Arguments

`lower.boundary`	lower boundaries
`upper.boundary`	upper boundaries
`look`	current look number
`t`	time of looks
`noncen`	noncentrality parameter
`current`	current Z statistic

Value

numeric

Examples

```
t<-c(0.33,0.67,1)
f<- function(t) 0.025*t^4
g<-function(t) 0.20*t^3
a<-f(t)
b<-g(t)
noncen<-pnorm(0.975)+pnorm(0.8)
curtail(b,a,1,t,noncen)
```

curtailDesign — *Function for calculating the probability for efficacy given known information*

Description

calculates the probability for efficacy given the Z value

Usage

```
curtailDesign(theObject, current0)
```

Arguments

theObject	The sequential design object
current0	The current Z value

Value

numeric

Examples

```
design1<-sequential.design()
design1<-calcBoundaries(design1)
design1<-setCurrentLook(design1,1)
prob1<-curtailDesign(design1,1.5)
```

getProbabilities — *Function that calculates the cumulative probabilities to declare efficacy and futility*

Description

Calculates the cumulative probabilities to declare efficacy and futility under the null hypothesis and the alternative hypothesis. It also returns the p-values for declaring efficacy and futility.

Usage

```
getProbabilities(theObject)
```

Arguments

theObject	The sequential design object

Value

numeric

Examples

```
design1<-sequential.design()
probs<-getProbabilities(design1)
```

`plotBoundaries` *Function that plots the efficacy and futility boundaries*

Description

Plots the efficacy and futility boundaries

Usage

```
plotBoundaries(theObject)
```

Arguments

`theObject`	The sequential design object

Examples

```
design1<-sequential.design()
design1<-calcBoundaries(design1)
plotBoundaries(design1)
```

`print` *Function that displays the features of the design*

Description

'print' displays the look times, the base alpha and beta spending functions, and the noncentrality parameter

Usage

```
print(theObject)
```

Arguments

`theObject`	The sequential design object

Examples

```
design1<-sequential.design()
design1<-calcBoundaries(design1)
design1<-setAlphaspendfString(design1,"0.025*t^4")
design1<-setBetaspendfString(design1,"0.15*t^3")
print(design1)
```

`seqmon` *Generic function that calculates boundary crossing probabilities used for monitoring clinical trials*

Description

Finds the probability that a sequence of standard normal random variables $z_1, z_2, \ldots, z_m$ derived from a normal stochastic process with independent increments will cross a lower and and upper boundary.

Usage

```
seqmon(a, b, t, int = rep(500, length(t)))
```

Arguments

`a`	Lower boundary as a numeric vector of length m
`b`	Upper boundary as a numeric vector of length m
`t`	Information times as a numeric vector of length m
`int`	number of intervals that the Z-space is partitioned into for calculation purposes, increasing this will improve accuracy, this is also a numeric vector of length m

Value

Produces a numeric vector of length $2m$ the first m components are the probability that the z_k will be less than a_k for some $k \leq i$ and be less than b_k for all $k \leq i$. The second m components are the probability that the z_k will be greater than b_k for some $k \leq i$ and be greater than a_k for all $k \leq i$.

Note that the last probability in the sequence is the overall significance level of a sequential design that uses `a` and `b` as upper and lower boundaries. To get power you subtract the $\mu\sqrt{(t)}$ from `a` and `b` where μ is the mean of z_m under the alternative hypothesis.

References

Schoenfeld, David A. "A simple algorithm for designing group sequential clinical trials." Biometrics 57.3 (2001): 972-974.

Examples

```
seqmon(a=c(0,0,0), b=c(qnorm(1-0.005),qnorm(1-0.005),2.025),
         t=c(.33,.66,1), int = rep(500, 3))

t=c(.33,.66,1)
u=(qnorm(.8)+qnorm(1-0.025))
seqmon(a=c(0,0,0)-u*sqrt(t), b=c(qnorm(1-0.005),qnorm(1-0.005),2.025)-u*sqrt(t),
         t=c(.33,.66,1), int = rep(500, 3))
```

---

sequential.design *The sequential design class*

Description

The S4 sequential design class

Usage

```
sequential.design(...)
```

Arguments

```
...
```

Details

The sequential design class stores the information of a sequential design, including revision history.

Value

an object of the class "sequential.design"

Author(s)

David A. Schoendfeld, PhD and Hui Zheng, PhD

References

Proschan, MA; Lan, KKG; Wittes JT,"Statistical Monitoring of Clinical Trials: A Unified Approach", Chapter 6, Springer 2006.

Schoenfeld DA, "A Simple Algorithm for Designing Group Sequential Clinical Trials", Biometrics. 2001 Sep;57(3):972-4.

Examples

```
design1<-sequential.design()
```

sequential.design-class
Class "sequential.design"

Description

The sequential design class

Objects from the Class

Objects can be created by calls of the form `sequential.design(...)`.

Slots

lower.boundary: Object of class "numeric"

upper.boundary: Object of class "numeric"

times: Object of class "numeric"

noncentrality: Object of class "numeric"

base.alpha.spend: Object of class "function"

base.beta.spend: Object of class "function"

base.alpha.spend.string: Object of class "character"

base.beta.spend.string: Object of class "character"

current.look: Object of class "numeric"

current.alpha.spend: Object of class "numeric"

current.beta.spend: Object of class "numeric"

times.history: Object of class "numeric"

alpha.spent.history: Object of class "numeric"

beta.spent.history: Object of class "numeric"

alpha.func.history: Object of class "numeric"

beta.func.history: Object of class "numeric"

date.stamp: Object of class "POSIXct"

Methods

calcBoundaries signature(theObject = "sequential.design"): ...

curtailDesign signature(theObject = "sequential.design"): ...

getProbabilities signature(theObject = "sequential.design"): ...

plotBoundaries signature(theObject = "sequential.design"): ...

print signature(theObject = "sequential.design"): ...

summary signature(theObject = "sequential.design"): ...

setAlphaspendfString signature(theObject = "sequential.design"): ...

setBaseAlphaspendf signature(theObject = "sequential.design"): ...

setBaseBetaspendf signature(theObject = "sequential.design"): ...

setBetaspendfString signature(theObject = "sequential.design"): ...

setCurrentLook signature(theObject = "sequential.design"): ...

setDatestamp signature(theObject = "sequential.design"): ...

setNoncentrality signature(theObject = "sequential.design"): ...

setTimes signature(theObject = "sequential.design"): ...

updateDesign signature(theObject = "sequential.design"): ...

Examples

```
showClass("sequential.design")
```

`setAlphaspendfString` *Function that Sets the expression of the base alpha spending function as a string*

Description

Sets the expression of the base alpha spending function as a string. This function is only used if one needs to display the base alpha spending function as a string. This function DOES NOT update the base alpha spending function. One can use setBaseAlphaspendf() to change the base alpha spending function. The spending functions and their string expressions should be defined only once per object. They should not be updated during any interim update to the design.

Usage

```
setAlphaspendfString(theObject, string0)
```

Arguments

`theObject`	The sequential design object
`string0`	The string of the expression of the base alpha spending function. Its argument need to be 't'.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
design1<-setAlphaspendfString(design1,'0.025*t^4')
```

setBaseAlphaspendf *Function that sets the base alpha spending function*

Description

Sets the base alpha spending function.

Usage

```
setBaseAlphaspendf(theObject, funct0)
```

Arguments

theObject	The sequential design object
funct0	The base alpha spending function. It needs to be defined before this method is called.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
f1<-function (t) 0.025*t^3.5
design1<-setBaseAlphaspendf(design1,f1)
```

setBaseBetaspendf *Function that sets the base beta spending function*

Description

Sets the base beta spending function.

Usage

```
setBaseBetaspendf(theObject, funct0)
```

Arguments

theObject	The sequential design object
funct0	The base beta spending function. It needs to be defined before this method is called.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
f2<-function (t) 0.15*t^2.5
design1<-setBaseBetaspendf(design1,f2)
```

`setBetaspendfString` *Function that sets the expression of the base beta spending function as a string*

Description

Sets the expression of the base beta spending function as a string. This function is only used if one needs to display the base beta spending function as a string. This function DOES NOT update the base beta spending function. One can use setBaseBetaspendf() to change the base beta spending function. The spending functions and their string expressions should be defined only once per object. They should not be updated during any interim update to the design.

Usage

```
setBetaspendfString(theObject, string0)
```

Arguments

`theObject`	The sequential design object
`string0`	The string of the expression of the base beta spending function. Its argument need to be 't'.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
design1<-setBetaspendfString(design1,'0.15*t^3.5')
```

setCurrentLook *Function that sets the current look number*

Description

Sets the current look number. The curent look is the one that last took place.

Usage

```
setCurrentLook(theObject, look0)
```

Arguments

theObject	The sequential design object
look0	The curent look number

Details

The curent look is the one that last took place. One can only set the current look forward. If the new current look number attempted is less than the old current look number, no action will take place and the current look number will not be updated.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
design1<-setCurrentLook(design1,2)
```

setDatestamp *Function that sets the date stamp of the design object*

Description

Sets the date stamp of the design object

Usage

```
setDatestamp(theObject, date0)
```

Arguments

theObject	The sequential design object
date0	The date value.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
design1<-setDatestamp(design1,as.POSIXct("2018-10-30"))
```

`setNoncentrality` *Function that sets the noncentrality parameter*

Description

Sets the noncentrality parameter.

Usage

```
setNoncentrality(theObject, noncent)
```

Arguments

`theObject`	The sequential design object
`noncent`	The noncentrality parameter

Details

The noncentrality paraeter is the expected drift at the end of the study. For example, if the study has a power of 80% using a one sided Z-test with 2.5% type 1 error, the noncentrality parameter is q(0.975)+q(0.8), where q() is the percentile function of the standard normal distribution.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
noncent<-qnorm(0.975,0,1)+qnorm(0.8,0,1)
design1<-setNoncentrality(design1,noncent)
```

setTimes *Function that sets the look times*

Description

Sets the look times. It is to be called only for the inital design, not for updating the design.

Usage

```
setTimes(theObject, time0)
```

Arguments

theObject	The sequential design object
time0	The look times.

Value

an object of class "sequential.design"

Examples

```
design1<-sequential.design()
design1<-setTimes(design1,c(1,2,3))
```

summary *Function that shows the cumulative probabilities for efficacy and futility*

Description

Shows the cumulative probability for efficacy and futility under the null and alternative hypotheses, the corresponding p-values, and the boundaries for Z at each look

Usage

```
summary(theObject)
```

Arguments

theObject	The sequential design object

Examples

```
design2<-calcBoundaries(sequential.design())
summary(design2)
```

`updateDesign` *Function that updates the design*

Description

Updates the design. This can be done in the process of the study, when the future look times need to be changed from those originally planned.

Usage

```
updateDesign(theObject, futureTimes)
```

Arguments

`theObject`	The sequential design object
`futureTimes`	The future look times.

Details

The efficacy and futility boundaries will be updated according to the new future look times. If the new final look is before the planned final look, the efficacy and futility boundaries will be updated, but the alpha and beta spending functions need not be updated. If the new final look is after the planned final look, the efficacy and futility boundaries will be updated, as well as the alpha and beta spending functions. The details are given in Proschan, Lan, and Wittes(2006) and Schoenfeld (2001). No historical information such as the past look times, the past alpha and beta spent, or the baseline spending function is updated.

Value

an object of class "sequential.design"

Author(s)

David A Schoenfeld, PhD and Hui Zheng, PhD

References

Proschan, MA; Lan, KKG; Wittes JT,"Statistical Monitoring of Clinical Trials: A Unified Approach", Chapter 6, Springer 2006.

Schoenfeld DA, "A Simple Algorithm for Designing Group Sequential Clinical Trials", Biometrics. 2001 Sep;57(3):972-4.

Examples

```
design1<-sequential.design()
design1<-setTimes(design1,c(1,2))
design1<-calcBoundaries(design1)
design1<-setCurrentLook(design1,1)
design2<-updateDesign(design1,c(3))
```

Index