Package 'binseqtest'

August 24, 2023

Type Package

Title Exact Binary Sequential Designs and Analysis

Version 1.0.4

Date 2023-08-24

Author Jenn Kirk, Michael P. Fay

Maintainer Michael P. Fay <mfay@niaid.nih.gov>

Description For a series of binary responses, create stopping boundary with exact results after stopping, allowing updating for missing assessments.

License GPL-3

Depends methods, graphics, stats, clinfun

Collate 'allFuncs.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2023-08-24 15:30:02 UTC

R topics documented:

`binseqtest-package` *Binary sequential tests*

Description

Design and analyze binary sequential tests

Details

The package creates designs for testing a series of binary responses sequentially. It allows checking after every response, or grouped sequential tests. Gives exact confidence intervals and p-values. Has an option for non-binding futility boundaries.

There are functions for creating the binary sequential boundaries or binary grouped sequential boundaries (see `designOBF`), creating tables of statistics (estimates, confidence intervals, and p-values) at specific stopping points in the boundary (see `link{stopTable}`), modifying the boundaries (see `modify`), and plotting the boundaries (`plot-methods`).

For details see Kirk and Fay (2014).

Author(s)

Jenn Kirk, Michael P. Fay

References

Kirk, JL, and Fay, MP (2014). An Introduction to Practical Sequential Inferences via Single Arm Binary Response Studies Using the binseqtest R Package. (to appear in American Statistician).

Examples

```
# create an O'Brien-Fleming-type design, with 2.5 percent error on each side with max N of 50
B<-designOBF(50)
# plot it
plot(B)
# create a table for N (total samples) values between 20 and 25
stopTable(B,Nrange=c(20,25))
# modify the boundary if you missed looks at N=30 through 35
Bmod<-modify(B,missN=30:35)
plot(Bmod)
```

`analyze` *Methods for calculating estimates, confidence intervals, and p-values from binary sequential boundaries*

Description

For routine use, these functions will not need to be called directly but are called from within the design functions (see `designOBF`). If needed, use `analyze` for any class representing a binary sequential boundary (see `bound-class`), and the appropriate function is called.

Usage

```
analyzeBound(object, theta0 = 0.5, stats = "all",
    alternative = "two.sided", conf.level = 0.95,
    tsalpha = NULL, ...)
analyzeBoundNBF(object, theta0 = 0.5, stats = "all",
    alternative = "two.sided", conf.level = 0.95,
    tsalpha = NULL, cipMatch = TRUE, ...)
```

Arguments

`object`	a binary sequential boundary (for classes see `bound-class`)
`theta0`	probability of success under the null
`stats`	character, either 'all' or 'pval'
`tsalpha`	vector of length 2, error on either side, if present overrides alternative and conf.level (see details)
`alternative`	character, either 'two.sided', 'less', or 'greater'
`conf.level`	confidence level
`cipMatch`	logical, for non-binding futility boundaries, should CI match the p-values on the binding boundary
`...`	further arguments to be passed

Value

if stats='all' returns an object of class 'boundEst', otherwise returns a numeric vector of p-values

See Also

See `analyze-methods`

`analyze-methods` *Calculate estimates, confidence intervals and p-values from binary sequential boundary*

Description

The method `analyze` calculates the estimate, confidence interval and p-values (both one-sided ones and the two-sided one) from a binary sequential boundary. The methods works on any of the classes that represent those boundaries (see `bound-class`).

Methods

`signature(object = "ANY")` Generic function: see `analyze`

`signature(object = "abparms")` Calculate estimates, confidence intervals and p-values from 'abparms' object.

`signature(object = "bound")` Calculate estimates, confidence intervals and p-values from 'bound' object.

`signature(object = "boundNBF")` Calculate estimates, confidence intervals and p-values from 'boundNBF' object.

`binseqtest-internal` *Internal functions*

Description

Internal functions, not to be called by user

Usage

```
validAbparms(object)
validBound(object)
validBoundEst(object)
validBoundNBF(object)

abBindBothCalcK(object)
abtoBound(from)

pCalc(S,N,K,order,theta0=.5,alternative="two.sided",ponly=FALSE)
ciCalc(S,N,K,order,type="upper",alpha=0.025)

missNAbparms(ab,missN=NULL,...)
```

Arguments

object	object, usually a boundary of some class
from	an object of class abparms
S	vector of number of successes
N	vector of number of trials
K	vector of number of ways to reach each bounary point
order	vector of ordering of boundary points
theta0	null value of probability of success each binary random variable
alternative	character, either 'two.sided', 'less', or 'greater'
ponly	logical, should only the specific p-value type given by alternative be calculated
type	character, type of one-sided confidence interval to calculate, either 'upper' or 'lower'
alpha	numeric, amount of error to allow on the one side of the confidence interval
ab	object of class 'abparms'
missN	numeric vector, the N values where assessments are missed
...	arguments passed to other functions, not used

Details

The validXX functions check that the object is a valid member of the class XX. For example, validBound checks that a bound object is OK by sum the probability distribution using the N,S, and K values and checking that it is within computer error of 1. The validity checks are run automatically by the new() function as part of the S4 implementation.

The function abBindBothCalcK takes an abparms object and creates a bound object. It requires calculating K, which is the number of ways to reach each boundary point. It ignores the binding argument and assumes all boundaries are binding. The abtoBound function uses the binding argument to create either a bound object (for binding='both') or a boundNBF object otherwise. Users can use the as function to coerce an abparms object to a bound object.

The function pCalc takes a boundary and calculates p-values, and outputs a vector of p-values (ponly=TRUE) or list of 3 vectors (plower,pupper, pval).

The function cCalc takes a boundary and calculates one of the one sided confidence intervals as directed by the type argument (either 'upper' or 'lower').

The functions analyzeBound and analyzeBoundNBF take objects of the bound and boundNBF classes and create ones of the boundEst and boundNBFEst classes. This means basically that the confidence intervals and p-values are calculated that go with those bounds.

The functions getAlternative and getTSalpha get those parameters from the inputs.

The function missNAbparms modifies abparms objects to reflect missing assessments. This is the working function for the missN option in modify.

bound-class *Classes for binary sequential boundaries*

Description

There are several classes that represent binary sequential boundaries. The most simple is the abparms class, then comes the bounds class (which contains abparms), then comes boundNBF class (which contains bound class), or boundEst class (which contains bounds class), then comes boundNBFEst (which contains boundNBF). See details for which slots go with which classes.

Details

The simplest representation of a binary sequential boundary is the abparms class, represented by a vector of the total number of trials (Nk) where to stop, and denoting stopping when number of sucesses, S, is S>=b or S<= a. One sided boundaries can be represented by all NA values for either a or b. Often times a two-sided boundary treats one side as a superiority boundary which must be stopped if crossed (a binding boundary), while the other side of the boundary is a futility boundary which may be ignored (a non-binding boundary). For example when binding='upper', then p-values for the upper boundary are calculated as if the lower boundaries are ignored if crossed and stopping happens on the lower side at max(Nk) instead, while the p-values for the lower and end boundary points are calculated using all (lower,upper and end) boundaries.

Next is the bound class which adds the slots N (number of trials at each boundary point), S (number of sucesses at each point), K (number of ways to get to each point), order (ordering of points for p-value calculations), UL('upper','lower' or 'end').

Slots

`Nk`: vector of number of samples at boundary stopping points

`a`: vector for lower bound, stop if S out of Nk is less than or equal a. NA denotes do not stop.

`b`: vector for upper bound, stop if S out of Nk is greater than or equal b. NA denotes do not stop.

`binding`: character specifying which boundary section is binding, either 'both', 'upper', or 'lower'

`alternative`: character specifying alternative, 'two.sided', 'less', or 'greater'

`N`: vector of number of samples at boundary stopping points

`S`: vector of number of sucesses at boundary stopping points

`K`: vector of number of ways to get to each boundary point

`order`: vector of ordering of points

`UL`: character vector denoting part of boundary, either 'lower' or 'upper' or 'end'

`estimate`: vector of estimates of theta, probability of success

`lower`: vector of lower confidence intervals for theta

`upper`: vector of upper confidence intervals for theta

`conf.level`: confidence level associated with confidence intervals

`alpha`: error on either side

`theta0`: null value for theta

`plower`: vector of lower p-values

`pupper`: vector of upper p-values

`pval`: vector of p-values as directed by alternative slot

Methods

There is a `plot` and a `points` method for `boundEst` objects.

Author(s)

Jenn Kirk, Michael P. Fay

Examples

```
new("abparms",Nk=200)
```

`designOBF` *Design Sequential Binary Boundary*

Description

There are several functions that create binary sequential boundaries. The function `designAb` allows great flexibility in creating user defined boundaries. The functions `designOBF` and `designOBFpower` create boudaries of the O-Brien-Fleming type, extending those boundaries to allow looks after every observation. The former (`designOBF`) uses a user defined maximum number of observations (Nmax), while the latter (`designOBFpower`) uses the power argument to try to find a design with a smaller maximum that achieves the desired power. The functions `designFixed` and `designFixedpower` are analogous for fixed sample designs. The function `designSimon` uses the `ph2simon` from the **clinfun** package to create boundaries using Simon's (1989) two-stage design.

Usage

```
designAb(Nk, a = NULL, b = NULL, theta0 = NULL,
    tsalpha = NULL, alternative = "two.sided",
    conf.level = 0.95, binding = "both")

designOBF(Nmax,theta0 = 0.5,  k = Inf,  tsalpha = NULL,
    alternative = "two.sided", conf.level = 0.95,
    binding = "both")
designOBFpower(theta0 = 0.5, theta1=.6,  k=Inf,
    power=.9, tsalpha = NULL, alternative = "two.sided",
    conf.level = 0.95, binding = "both", allNgreater=FALSE,
    checkmax=10, maxNmax=2*ss)

designFixed(Nmax, theta0 = 0.5, tsalpha = NULL,
```

```
    alternative = "two.sided", conf.level = 0.95)
designFixedpower(theta0 = 0.5, theta1 = 0.6, power = 0.8,
    maxNmax = Inf, tsalpha = NULL, alternative = NULL,
    conf.level = 0.95, allNgreater = FALSE)

designSimon(theta0, theta1, alpha = 0.05, beta = 0.2,
    type = c("optimal", "minimax"), nmax=100)
```

Arguments

Nk	vector of unique N values where there is stopping
a	numeric vector with length(a)=length(Nk)-1, stop if number of successes out of Nk[i] is less than or equal to a[i] (see details)
b	numeric vector with length(a)=length(Nk)-1, stop if number of successes out of Nk[i] is greater than or equal to b[i] (see details)
Nmax	maximum number of observations for the design
maxNmax	maximum number for Nmax (see details)
k	number of looks at the data, Inf denotes looking after each observation
theta0	probability of success under the null
tsalpha	vector of length 2 with nominal significance levels for each side, if not NULL overrides conf.level and alternative (see getTSalpha)
conf.level	confidence level, ignored if tsalpha is not NULL
alternative	character, alternative hypothesis, either 'less', 'greater' or 'two.sided'
binding	character, which sides are binding: 'both', 'upper', or 'lower'
theta1	probability of success under alternative for power calculations
power	nominal power, boundary strives to have power under the alternative at least equal to power
allNgreater	logical, if TRUE max(N) will be at least as large as the fixed sample size for which all greater N have power>power
checkmax	integer, on the iteration checkmax, check that Nmax has power at least power
alpha	one sided alpha level for test theta>theta0
beta	1-power, for theta1
type	character, type of 2-stage design, either 'optimal' or 'minimax'
nmax	maximum total sample size, cannot be higher than 1000

Details

The tsalpha, alternative, and conf.level are input into the getTSalpha function to output a tsalpha vector. The tsalpha vector allows the nominal error to be different on each side. For details see getTSalpha.

For designAb, when you do not want to stop on the lower or upper boundary at any value of Nk, the associated value of a (lower) or b (upper) should be NA.

The designOBF function calculates a boundary that stops whenever the B-value (Lan and Wittes, 1988) is larger than one cutoff value or smaller than a different cutoff value. The cutoff values are chosen so that the probability of spending alpha on the appropriate side is almost all spent while still rejecting at at least one end value of the boundary.

The function designOBFpower repeatedly calls designOBF and finds the design that gives sufficient power under a given alternative. Specifically, by setting Nmax to Nmaxi in designOBF, where Nmaxi is increased by 1 at each iteration. The initial Nmaxi is either the first N that gives a large enough power in the fixed sample size design (allNgreater==FALSE) or the first N such that all larger N will give enough power for fixed samples (allNgreater==TRUE). On the (checkmax)th iteration, check that the power will be large enough when Nmaxi equals Nmax (from designOBFpower call). So if you set checkmax=1 then you will check the largest value of Nmax first, but this may be inefficient since larger values of Nmax in the obf call are slower.

See Kirk and Fay (2014) for an introductory paper about exact binary sequential tests using the binseqtest package.

Value

a object of class boundEst

References

Kirk, J, and Fay, MP (2014). An Introduction to Practical Sequential Inferences via Single Arm Binary Response Studies Using the binseqtest R Package. (to appear in American Statistician).

Lan, KKG, and Wittes, J (1988). The B-Value: A Tool for Monitoring Data. Biometrics 44:579-585.

Simon R. (1989). Optimal Two-Stage Designs for Phase II Clinical Trials. Controlled Clinical Trials 10, 1-10.

EN *Expected sample size for boundary.*

Description

Calculate expected sample size for bound object, after inputing theta.

Usage

```
EN(object, theta = 0.6)
```

Arguments

object	a object representing a binary sequential class (bound-class)
theta	a vector of parameters representing the probability of a success

Value

a vector of expected sample sizes associated with the theta argument.

See Also

See Also powerBsb

Examples

```
B<-designAb(Nk=c(20,40),a=c(5),b=c(15),theta0=.5)
En<-EN(B,theta=c(.1,.5,.6))
En
```

---

getTSalpha *Two-sided alpha, alternative, and confidence level*

Description

Two functions to find tsalpha and alternative.

Usage

```
getTSalpha(tsalpha = NULL, alternative = NULL, conf.level = NULL)
getAlternative(tsalpha)
```

Arguments

tsalpha	vector of length 2 with nominal significance levels for each side, if not NULL overrides conf.level and alternative (see details)
conf.level	confidence level, ignored if tsalpha is not NULL
alternative	character, alternative hypothesis, either 'less', 'greater' or 'two.sided'

Details

The tsalpha is a vector of length 2 giving the nominal error for each side of confidence intervals. The function getTSalpha creates a tsalpha vector, allowing its creation either directly (non-null input for the argument tsalpha simply outputs that same argument), or through the alternative and conf.level arguments. The element tsalpha[1] is the nominal error on the lower side, so for example if tsalpha=NULL, alternative='greater', andconf.level=.95, then getTSalpha outputs the vector c(0.05,0). In other words, if on rejection you want to conclude that $\theta > \theta_0$, then you want all the nominal error to be on the lower side. Similarly tsalpha[2] is the nominal error on the upper side, and tsalpha=NULL, alternative='less', andconf.level=.95, gives c(0,0.05). If tsalpha=NULL, alternative='greater', and conf.level=.95, then outputs the vector c(0.025,0.025). You must supply either tsalpha or both alternative and conf.level.

Value

getTSalpha returns a tsalpha vector (see details), and getAlternative gives the character vector for the appropriate alternative.

Examples

```
getTSalpha(conf.level=.95,alternative="two.sided")
getAlternative(c(0,.025))
```

`modify` *Modify binary sequential boundary*

Description

Modify several different aspects of a binary sequential boundary. Most modifications do not change the stopping boundaries. The exceptions are 'missN', which allows modifications for missing assessments, and 'closeout', which allows for early stopping of the trial for administrative reasons (i.e., reasons that do not depend on the responses in the trial). Other modifications possible: level of the confidence intervals (using tsalpha, conf.level, or alternative), which boundaries are binding (i.e., can change from a boundary with binding futility boundaries to one with non-binding futility boundaries), null hypothesis value (theta0), and whether the confidence intervals should match the non-binding futility p-values on the superiority boundaries (cipMatch).

Usage

```
modify(b, missN = NULL, theta0 = NULL, tsalpha = NULL,
    conf.level = NULL, alternative = NULL, cipMatch = TRUE,
    binding = NULL, closeout=NULL,  ...)
```

Arguments

`b`	an object of class `boundEst`
`missN`	a vector of missed assessments
`theta0`	null hypothesis probability of success
`tsalpha`	vector of length 2 with nominal significance level, if not NULL overrides `conf.level` and `alternative`
`conf.level`	confidence level, ignored if tsalpha is not NULL
`alternative`	character, alternative hypothesis, either 'less', 'greater' or 'two.sided'
`cipMatch`	logical, for non-binding futility boundaries, should CI match the p-values on the binding boundary
`binding`	character, which sides are binding: 'both', 'upper', or 'lower'
`closeout`	total number of trials at early closeout
`...`	other parameters passed

Value

an object of class `boundEst`

Examples

```
b<-designOBF(50)
bmod<-modify(b,missN=30:36)
par(mfrow=c(2,1))
plot(b)
plot(bmod)
```

---

plot-methods *Methods for Function* plot *and* points *in Package* **binseqtest**

Description

Plot binary sequential boundaries for "boundEst" objects.

Usage

```
## S4 method for signature 'boundEst,missing'
plot(x,
    rcol = c(orange = "#E69F00", blue = "#56B4E9", green = "#009E73"),
    rpch = c(openCircle=1, filledCircle=16, filledDiamond=18),
    bplottype = "NS",
    newplot = TRUE, dtext=NULL, grid=50, xlab=NULL, ylab=NULL, ...)

## S4 method for signature 'boundEst'
points(x, ...)
```

Arguments

x	an object of class "boundEst"
rcol	rejection color vector, rcol[1]=fail to reject, rcol[2]=reject, conclude theta>theta0, rcol[3]=reject, conclude theta< theta0 (see details)
rpch	rejection pch vector, correspond to same categories as rcol vector
bplottype	character, either 'NS' (default), 'FS', 'NB', 'NZ', or 'NE' (see details)
newplot	logical, should a new plot be started? if FALSE add to existing plot (only makes sense to add to plot with the same bplottype)
dtext	logical, add descriptive text? if NULL only adds text when newplot=TRUE (used for bplottype='NS' or 'FS')
grid	numeric, if maximum possible total trials<=grid then add gridlines (used for bplottype='NS' or 'FS')
xlab	title for x axis, if NULL value depends on bplottype
ylab	title for y axis, if NULL value depends on bplottype
...	other arguments to the plot function can be passed here.

Details

The default rcol vector are good colors for distinguishing for those with color blindness. Text is printed on the unused portion of the plot, which uses the color names taken from the rcol vector names.

Their are several different types of plots, selected by the `bplottype` argument, where the value is a character string with 2 characters, the first representing the x-axis and the second representing the y-axis. For example `bplottype='NS'` denotes N=total number of trials on the horizontal axis, and S=number of successes on the vertical axis. Other plots are: 'FS'=failure by successes; 'NB'=total by B-values; 'NZ'=total by Z-scores; 'NE'=total by estimates and confidence intervals. The type 'NE' is only defined if there are only 1 value for each N on the upper and 1 value for each N on the lower part of the boundary. Otherwise, the confidence intervals would overlap and be uninformative. For 'NE' the end of the boundary is not plotted because of that overlapping.

For some examples, see plot section of the vignette. The method points just calls `plot(x,newPlot=FALSE,...)`.

Methods

`signature(x = "ANY", y = "ANY")` Generic function: see `plot`.

`signature(x = "boundEst", y = "missing")` Plot binary sequential boundaries for `x`.

`signature(x = "ANY")` Generic function: see `points`.

`signature(x = "boundEst")` Add points associated with the binary sequential boundaries for `x` to a plot.

Examples

```
b<-designOBF(50,theta0=.5)
plot(b,bplottype="NE")
plot(b)
b2<-designFixed(49,theta0=.5)
points(b2,rpch=c(17,17,17))
```

---

powerBsb *Power for binary sequential boundary*

Description

Calculate power from `boundEst` object for vector of alternatives

Usage

```
powerBsb(object, theta = 0.6, alternative = NULL)
```

Arguments

object	a 'boundEst' object
theta	vector of theta values, probability of success
alternative	character, either 'two.sided', 'less', or 'greater'

Details

Power to reject. For alternative='greater' reject when pU<tsalpha['alphaUpper'], and for alternative='less' reject when pL<tsalpha['alphaLower']. For alternative='two.sided' if theta[i]>theta0 reject when pU<tsalpha['alphaUpper'], if theta[i]<theta0 reject when pL<tsalpha['alphaLower'], if theta[i]==theta0 and tsalpha['alphaUpper']<=tsalpha['alphaLower'] reject when pU<tsalpha['alphaUpper'], and if theta[i]==theta0 and tsalpha['alphaUpper']>tsalpha['alphaLower'] reject when pL<tsalpha['alphaLower'].

Value

a vector with power associated with the parameter vector theta

Examples

```
B<-designAb(Nk=c(20,40),a=10,theta0=.4)
powerBsb(B,theta=c(.1,.4,.8),alternative="less")
```

---

prStop *Probabilty of Stopping*

Description

Calculates the probability of stopping at any point in a binary sequential boundary.

Usage

```
prStop(object, theta = NULL)
```

Arguments

object	an object of class boundEst
theta	probability of a positive response

Value

A list with the following elements

B	the boundEst object inputted
Nupper	vector of N values for stopping on upper boundary
dStopUpper	vector of probabilities for stopping at each value of Nupper
pStopUpper	vector of cumulative probabilities for stopping by each value of Nupper

Nlower	vector of N values for stopping on lower boundary
dStopLower	vector of probabilities for stopping at each value of NLower
pStopLower	vector of cumulative probabilities for stopping by each value of NLower
Nend	N value at the end of the boundary, i.e., max(N)
dStopEnd	probability of stopping at the end of the boundary
check	check value, should be 1 or very close to it

See Also

powerBsb

Examples

```
b<-designOBF(20,theta0=.5)
prStop(b,theta=.5)
```

---

stopTable *Create data frame with statistics for stopping boundary*

Description

Takes boundEst object and creates data frame with p-values and confidence intervals for stopping points between Nmin and Nmax.

Usage

```
stopTable(object, Nrange = c(0,Inf),
    Srange=c(0,Inf),output="all",file="stopTableOutput.csv")

## S3 method for class 'stopTable'
print(x,digits=c(3,5),maxnprint=Inf,...)
```

Arguments

object	object of class boundEst
Nrange	numeric vector, range of total trials to output boundary points (see details)
Srange	numeric vector, range of successes to output boundary points (see details)
output	character, type of output, 'all','csv', or 'print'
file	character, name of file to output comma separated file
x	object of class 'stopTable'
digits	vector of length 1 or 2, first element is digits to print for estimates and confidence intervals (and p-values if length(digits)=1), second element is digits for p-values
maxnprint	number of rows to print from the table
...	additional arguments to be passed to print function, not needed

Details

Create table with statistics at selected stopping points. The arguments `Nrange` and `Srange` select which points to output. If `Nrange` and `Srange` is of length 2, then the function selects output within the ranges of the associated N and S values. If `Nrange` and `Srange` is of length 1, then the function selects only output with exactly the specified values. Note that because `Nrange` is the only argument that starts with 'N', then `stopTable(object,N=41)` and `stopTable(oject,Nrange=41)` are equivalent. Output is a `stopTable` object (default), a data.frame (output='data.frame'), or a .csv file (output='csv'). The `print.stopTable` function is an S3 print function for `stopTable` objects, i.e., it determines the default printing to screen for those objects.

Value

Either a `stopTable` object, a data.frame, or a .csv file. The `stopTable` object is a list with elements

`conf.level`	confidence level=1-lowerError-upperError
`lowerError`	c.i. error on lower side
`upperError`	c.i. error on upper side
`nullTheta`	probability of success under null hypothesis
`binding`	character, which boundaries are binding: 'both', 'lower', or 'upper'
`alternative`	character, alternative hypothesis either 'two.sided', 'less', or 'greater'
`table`	a data.frame with variables, S,N, estimate, lower (confidence limit), upper (confidence limit), pL (lower one-sided p-value), pU (upper one-sided p-value), pts (two-sided p-value), and UL (character giving part of the boundary: 'lower', 'upper', or 'end')

The data.frame (or .csv file) returns the table element of the `stopTable` with the other elements added as variables (or columns).

`summary-methods` *Methods for Function* `summary` *in Package* **binseqtest**

Description

Objects of class `boundEst` have a `summary` method. It basically calls stopTable(object)

Methods

`signature(object = "ANY")` Gives a summary of object, usually a little more calculations than associated with print or show

`signature(object = "boundEst")` calls `stopTable(object, output='print')`

`unirootDiscrete` *Identify where a non-increasing function changes sign*

Description

Let f be a non-increasing (or non-decreasing) function that changes sign within the interval specified. If 'pos.side'=TRUE (or FALSE) then `unirootDiscrete` finds the value x such that f(x) is closest to the sign change and is positive (or negative).

Usage

```
unirootDiscrete(f, interval, lower = min(interval),
    upper = max(interval), tol = 10^-5, pos.side = FALSE,
    print.steps = FALSE, maxiter = 1000, ...)
```

Arguments

`f`	function for which a root is needed
`interval`	an interval giving minimum and maximum allowable values for root
`lower`	lower bound for root
`upper`	upper bound for root
`tol`	absolute tolerance, abs(true root-estimated root)<= tol
`pos.side`	if TRUE finds value x closest to the sign change in f, such that f(x)>0
`print.steps`	if true prints interations
`maxiter`	maximum number of iterations
`...`	additional arguments to f

Details

The algorithm evaluates f(x) iteratively, and the change in 'x' is halved each iteration until the change in 'x' is less than tol. Then the root is returned according to the pos.side parameter.

Value

A list with the folllowing elements,

`iter`	number of iterations (times f is evaluated)
`f.root`	value of f(x), where x is the root
`root`	the root x, where f(x)>=0 if pos.side=TRUE

...

Author(s)

M.P. Fay

Examples

```
test<-function(x,parm=10.987654321){ ifelse(x>=parm,1,-1) }
unirootDiscrete(test,lower=0,upper=100,tol=10^-4,pos.side=FALSE,print.steps=TRUE)
```

Index