Package ‘simctest’

November 4, 2019

Version 2.6
Date 2019-11-03
Title Safe Implementation of Monte Carlo Tests
Author Axel Gandy <a.gandy@imperial.ac.uk> with contributions from Patrick Rubin-Delanchy <patrick.rubin-delanchy@imperial.ac.uk>, Georg Hahn <ghahn@cantab.net>, Dong Ding <dong.ding10@imperial.ac.uk>
Maintainer Axel Gandy <a.gandy@imperial.ac.uk>
Depends R (>= 2.2.0), methods, stats
Suggests
Description Algorithms for the implementation and evaluation of Monte Carlo tests, as well as for their use in multiple testing procedures.
License GPL (>= 2)
URL http://www2.imperial.ac.uk/~agandy
Repository CRAN
NeedsCompilation yes
Date/Publication 2019-11-04 13:20:02 UTC

R topics documented:

confidenceLimits-methods ................................................... 2
confint-methods ............................................................. 3
cont-methods ................................................................. 4
getalgprecomp ............................................................... 4
getbounds-methods ........................................................... 5
getL-methods ................................................................. 6
getNumber-methods ............................................................ 6
getSamples-methods ........................................................... 7
getU-methods ................................................................. 7
hBH-methods ................................................................. 8
hBonferroni-methods .......................................................... 9
hPC-methods ................................................................. 9
Methods for class ‘mmctestres’ and ‘mmctest’, Package ‘simctest’

Description

Function which returns a list containing lower confidence limits (vector ‘lowerLimits’) and upper confidence limits (vector ‘upperLimits’).

Usage

certaintyLimits(obj)

Arguments

obj  object of type ‘mmctestres’ or ‘mmctest’.

Methods

certaintyLimits(obj) works with object of type mmctestres or mmctest.
Examples

```r
fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));
res <- confidenceLimits(a);
lower <- res$lowerLimits;
upper <- res$upperLimits;
```

Description

Computes a confidence interval for the p-value

Usage

```
confint(object, parm, level=0.95,...)
```

Arguments

- `object` An object of type `sampalgres` resulting from a previous call to `run` or `cont`.
- `parm` must be missing.
- `level` the desired coverage probability.
- `...` additional argument(s). Currently not used

Methods

- `object = "ANY", parm = "ANY"` Generic function: see `confint`.
- `object = "sampalgres", parm = "missing"` Computes a confidence interval for the p-value with the coverage probability given by `level`.

Examples

```
alg<-getalgonthefly()
res <- run(alg, function() runif(1)<0.05);
res
confint(res)
```
### cont-methods

**Methods for Function ‘cont’ in Package ‘simctest’**

#### Description
Continues the sampling for some more steps.

#### Usage
```r
cont(data, steps)
```

#### Arguments
- **data**: a result of a run of a sampling algorithm that has not come to a conclusion yet.
- **steps**: maximum number of further iterations to take.

#### Methods
- **data = "sampalgres"** works with the algorithm based on precomputation.
- **data = "sampalgontheflyres"** works with the on-the-fly algorithm.
- **data = "mmctestres"** works with object of type "mmctestres".

#### Examples
```r
res <- simctest(function() runif(1)>0.95, maxsteps=10);
res
res <- cont(res, 1000)
res
res <- cont(res, 1000)
res
```

### getalgprecomp

**Construct algorithms**

#### Description
Constructs classes of type `sampalgonthefly` and `sampalgPrecomp`.

#### Usage
```r
getalgonthefly(level = 0.05, epsilon = 0.001, halfspend = 1000)
getalgprecomp(level = 0.05, epsilon = 0.001, halfspend = 1000)
```
Arguments

- **level**: the threshold.
- **epsilon**: the bound on the resampling risk.
- **halfspend**: number of steps after which half the error has been spent.

Value

getalgonthefly returns an object of type `sampalgonthefly`. getalgprecomp returns an object of type `samplerPrecomp`.

Author(s)

Axel Gandy

References


Examples

```r
alg<-getalgprecomp()
run(alg, function() runif(1)<0.01)

alg<-getalgonthefly()
run(alg, function() runif(1)<0.01)
```

Description

returns bounds on the p.value if the algorithm has not stopped yet.

Usage

getbounds(data)

Arguments

data: an object of type `sampalgres` or `linkS4class(sampalgontheflyres)`. 
Methods for Function `getL` in Package `simctest`

**Description**

Returns the lower boundary for the stopping rule.

**Usage**

```r
## S4 method
getL(alg, ind)
```

**Arguments**

- `alg` the sampling algorithm
- `ind` a vector of indices at which the lower stopping boundary should be returned

**Methods**

- `alg = "sampalgPrecomp"` the sampling algorithm to be used

**Examples**

```r
getL(getalgprecomp(), 1:100)
```

Methods for Function ‘`cont`’ in class ‘`mmctestres`’, Package ‘`simctest`’

**Description**

Function to request number of hypotheses.

**Usage**

```r
ceturnber(obj)
```

**Arguments**

- `obj` object of type "mmctSampler" derived from class "mmctSamplerGeneric".

**Methods**

- `getNumber(obj)` works with object of type "mmctSampler" derived from class "mmctSamplerGeneric".
Examples

```r
fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
number <- getNumber(i);
```

Description

Function to request further samples from certain hypotheses.

Usage

```r
getSamples(obj, ind, n)
```

Arguments

- **obj** object of type "mmctSampler" derived from class "mmctSamplerGeneric".
- **ind** vector containing the indices of hypotheses for which further samples are requested.
- **n** vector containing number of further samples for each hypothesis in vector ‘ind’.

Methods

- **getSamples(obj, ind, n)** works with object of type "mmctSampler" derived from class "mmctSamplerGeneric".

Examples

```r
fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
samples <- getSamples(i, c(1,2), c(2,2));
```

Description

Returns the upper boundary for the stopping rule.

Usage

```r
getU(alg,ind)
```
Arguments

alg the sampling algorithm

ind a vector of indices at which the upper stopping boundary should be returned

Methods

alg = "sampalgPrecomp" the sampling algorithm to be used

Examples

getU(getalgprecomp(),1:100)

---

**hBH-methods**  
*Method for class ‘mcmtest’, Package ‘simctest’*

**Description**

Implementation of the multiple testing procedure by Benjamini-Hochberg.

**Usage**

hBH(p, threshold)

**Arguments**

p object of type "numeric".

threshold object of type "numeric".

**Methods**

hBH(p, threshold) applies the Benjamini-Hochberg procedure to p-values p with given threshold, returns rejected indices

**Examples**

hBH(runif(10), threshold=0.1)
**hBonferroni-methods**  
*Method for class ‘mcmtest’, Package ‘simctest’*

**Description**  
Implementation of independent (Bonferroni) multiple testing.

**Usage**  
hBonferroni(p, threshold)

**Arguments**  
p: object of type "numeric".  
threshold: object of type "numeric".

**Methods**  
hBonferroni(p, threshold) performs independent multiple testing using the Bonferroni correction at given threshold, returns rejected indices

**Examples**  
hBonferroni(runif(10), threshold=0.1)

---

**hPC-methods**  
*Method for class ‘mcmtest’, Package ‘simctest’*

**Description**  
Implementation of the multiple testing procedure by Pounds&Cheng.

**Usage**  
hPC(p, threshold)

**Arguments**  
p: object of type "numeric".  
threshold: object of type "numeric".

**Methods**  
hPC(p, threshold) applies the modification by Pounds&Cheng to p-values p with given threshold, returns rejected indices
Examples

hPC(runif(10), threshold=0.1)

mcp

Function mcp in package 'simctest'

Description

An algorithm for the computation of the power of Monte Carlo tests with guaranteed precision

Usage

mcp(genstream, alpha=0.05, delta="adaptive", cp=0.99, maxeffort=Inf, options = list())

Arguments

gencode: a function that returns a function that returns a random Bernoulli variable (each stream corresponds to a dataset. 0 = (T<t), 1 = (T>=t) where t is computed from the dataset and T is a resampled test-statistic from that dataset.)

alpha: the level of the test.

delta: the desired length of confidence interval, or "adaptive" if using adaptive delta. See details.

maxeffort: maximum effort. Effort is total number of samples taken. Set to finite value if needed (the resulting confidence interval still has the guaranteed coverage probability, but may not be as ‘short’ as desired). Can also interrupt the algorithm during main loop and get a result of class "mcpres".

cp: the desired coverage probability.

options: Additional options. See details

Details

options$maxeffort: set to maximum allowable effort.

options$reports: set to FALSE if onscreen reports are not wanted.

options$file: optional file-name to save results to.

options$pilotn: number of streams in pilot (1000 by default).

options$pilotmaxsteps: maxsteps in pilot (1000 by default).

options$gammapilotprop: proportion of error spent on pilot CI (0.1 by default)

options$gammatestprop: proportion of error spent on testing remaining paths (default is 0.1)

options$spendgammatest: spending sequence for the testing procedure on the remaining streams. Must be a non-negative function of integers with positive limit 1 (t/(20 + t) by default).

options$eta: internal parameter to the testing procedure on the remaining streams (0.05 by default).
options$maxstepsbase: initial maximum number of steps (500 by default)
options$maxstepsinc: multiplier for the maximum number of steps thereafter (1.5 by default).
options$maxbatch: multiplier for the maximum number of steps thereafter (200000 by default).
options$deltamid: adaptive delta function. Describes the length of the confidence interval desired depending on the midpoint of the interval. By default the function requires 0.02 for intervals containing 0.05 or lower or 0.95 or higher, and 0.1 otherwise. If using non-default adaptive delta must also specify epsilon (below).
options$epsilon: error probability for each stream. Only set if using non-standard adaptive delta.

Value
An object of class "mcpres" with slots:

* int: confidence interval for power.
* cp: coverage probability.
* beta: Estimate of power.
* N: the number of streams started in main loop (or in pilot if stopped after pilot).
* effort: total number of samples generated.
* rescount: number of positive and negative outcomes.
* truncated: boolean indicating whether procedure was truncated by user-specified maxeffort.
* taccepted: boolean indicating whether the procedure stopped as a result of a hypothesis test or brute force (the confidence interval coverage probability is guaranteed in either case.)

Author(s)
Axel Gandy and Patrick Rubin-Delanchy

References

See Also
mkdeltamid

Examples
#The following example takes a bit of computing time
## Not run:
#Example where we know the power should be the level of the test
genstream <- function(){p <- runif(1); function(N){runif(N) <= p}}
res <- mcp(genstream, alpha=0.05, delta="adaptive", cp=0.99)
#should find confidence interval of length 0.02 centered around 0.05
res

```r
## End(Not run)
```

---

*Description*

Result returned by `mcp`

*Objects from the Class*

Objects can be created by calls of the form `new("mcpres", ...)`.

*Slots*

- `int`: Object of class "numeric"
- `cp`: Object of class "numeric"
- `beta`: Object of class "numeric"
- `N`: Object of class "numeric"
- `effort`: Object of class "numeric"
- `rescount`: Object of class "numeric"
- `truncated`: Object of class "logical"
- `taccepted`: Object of class "logical"

*Methods*

- `show` signature(object = "mcpres"): ...

*Author(s)*

Axel Gandy and Patrick Rubin-Delanchy

*References*


*Examples*

```
showClass("mcpres")
```
**Description**

Sequential implementation of the Monte Carlo test with p-value buckets.

Implementation of the Robbins-Lai (mctest.RL) and SIMCTEST (mctest.simctest) approaches to compute a decision interval (and decision) with respect to several thresholds/p-value buckets. The function “mctest” is a wrapper function for both the Robbins-Lai and the SIMCTEST approach which calls one of the two using an additional parameter "method" (method='simctest' for SIMCTEST and method='RL' for Robbins-Lai).

**Usage**

mctest(gen,J=Jstar,epsilon=0.001,batch=10,batchincrement=1.1,maxbatch=100, method=c("simctest","RL"))  
mctest.RL(gen,J=Jstar,epsilon=0.001,batch=10,batchincrement=1.1,maxbatch=100)  
mctest.simctest(gen,J=Jstar,epsilon=0.001,batch=10,batchincrement=1.1,maxbatch=100)

J

Jstar

## S3 method for class 'mctestres'

print(x,...)

**Arguments**

- **gen**
  function that performs one sampling step. Returns 0 (sampled test statistic does not exceed the observation) or 1 (sampled test static exceeds the observation)

- **method**
  which method to use for stopping

- **J**
  p-value buckets to use. A matrix with two rows, each column describes an interval bucket. Column names give the code for the interval bucket. Defaults to Jstar.

- **epsilon**
  error bound

- **batch**
  initial number of samples to use before checking for stopping

- **batchincrement**
  factor by which the batch size gets multiplied after each step. 1 would mean no increment

- **maxbatch**
  maximum batch size

- **x**
  object of type "mctestres"

- **...**
  further arguments

**Value**

mctest, mctest.RL and mctest.simctest all return an object of class type mctestres, which has a print function (print.mctestres).
An object of class mctestres is a list with the following components: step (total batched number of samples drawn), decision.interval (interval for the p-value), decision (expressing significance), est.\( p \) (an estimate of the p-value) and realn (the actual number of samples taken without batching).

References


Examples

```r
#Example used in the above paper
dat <- matrix(nrow=5,ncol=7,byrow=TRUE, c(1,2,1,1,0,1, 2,0,0,2,3,0,0, 0,1,1,2,7,3, 1,1,2,0,0,0,1, 0,1,1,1,1,0,0))

loglikrat <- function(data){
  cs <- colSums(data)
  rs <- rowSums(data)
  mu <- outer(rs,cs)/sum(rs)
  2*sum(ifelse(data<=0.5, 0,data*log(data/mu)))
}

resample <- function(data){
  cs <- colSums(data)
  rs <- rowSums(data)
  n <- sum(rs)
  mu <- outer(rs,cs)/n/n
  matrix(rmultinom(1,n,c(mu)),nrow=dim(data)[1],ncol=dim(data)[2])
}

t <- loglikrat(dat);

# function to generate samples
gen <- function(){loglikrat(resample(dat))>=t}

#using simctest
mctest(gen)
mctest.simctest(gen)
mctest.RL(gen)
```

```R

## Function mkdeltamid in Package “simctest”

### Description

Easy creation of adaptive delta function

#### Usage

```r
mkdeltamid(mindelta=0.02, maxdelta=0.1, llim=0.05, rlim=0.95)
```
Arguments

- `mindelta`: desired length of CI for regions of interest, such as when the power is less than 0.05 or greater than 0.95.
- `maxdelta`: desired length of CI when power is not in region of interest, e.g. between 0.05 and 0.95.
- `llim`: change if want different left limit (i.e. not 0.05).
- `rlim`: change if want different right limit (i.e. not 0.95).

Value

A function, say `deltamid`, that specifies the user’s desired precision depending on the midpoint of the computed confidence interval. If the current confidence interval has a midpoint `M`, then the algorithm will stop if `deltamid(M) <= length of CI`.

Author(s)

Axel Gandy and Patrick Rubin-Delanchy

References


Examples

```r
## only care about powers around 0.9 or higher
## (e.g. if want to check that the test is powerful enough).

deltamid <- mkdeltamid(mindelta=0.02, maxdelta=1, llim=0, rlim=0.9)

genstream <- function(){p <- runif(1); function(N){runif(N) <= p}}

## The power is 0.05. The algorithm should stop as soon as it is clear
## that the power is not larger than 0.9. (Must specify epsilon
## if using non-standard delta.)

res <- mcp(genstream, alpha=0.05, delta="adaptive", cp=0.99,
options=list(deltamid = deltamid, epsilon = 0.0001))

#should stop early.
res
```
Class "mmctest"

Description

Class which creates an object of type "mmctestres".

Objects from the Class

Objects can be created by calls of the form mmctest(h=...).

Slots

  internal: Object of class "environment"

Methods

  run signature(alg = "mmctest", gensample = "mmctSamplerGeneric", maxsteps = "numeric"): ...

Author(s)

  Axel Gandy and Georg Hahn

References


Examples

  fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
  i <- mmctSampler(fun,num=500,data=runif(500));
  a <- mmctest(h=hBH);
  a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));

Methods for class 'mmctest', Package 'simctest'

Description

  Constructor for class 'mmctest'.

Usage

  mmctest(epsilon=0.01, threshold=0.1, r=10000, h, thompson=F, R=1000)
mmctestres-class

Arguments

epsilon probability of any misclassification one is willing to tolerate
threshold threshold for testing.
r parameter of the spending sequence, see vignette
h reference to a multiple testing function of the form function(p, threshold) which returns the set of rejected indices.
thompson if set to true, mmctest will use a Thompson strategy to draw further samples
R number of repetitions (=draws from the posterior distributions) used to calculate empirical probabilities of each hypothesis being rejected – used to calculate weights in QuickMMCTest (option thompson=TRUE in the mmctest constructor)

Methods

mmctest(epsilon=0.01, threshold=0.1, r=10000, h) returns object of type 'mmctest'.

Examples

fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[ind[i]])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));

mmctestres-class Class "mmctestres"

Description

Class which stores current result of type "mmctest".

Objects from the Class

Objects should not be created directly. Objects returned by calls of the form new("mmctest",...) are of type mmctestres.

Slots

internal: Object of class "environment"
etpsilon: Object of class "numeric"
threshold: Object of class "numeric"
r: Object of class "numeric"
R: Object of class "numeric"
h: Object of class "function"
genSample: Object of class "mmctSamplerGeneric"
g: Object of class "numeric"
num: Object of class "numeric"
A: Object of class "numeric"
B: Object of class "numeric"
C: Object of class "numeric"
thompson: Object of class "logical"
rejprob: Object of class "logical"

Methods

  mainalg signature(obj = "mmctestres", stopcrit = "numeric"): ...
  cont signature(data = "mmctestres", steps = "numeric"): ...
  show signature(object = "mmctestres"): ...
  pEstimate signature(obj = "mmctestres"): ...
  rejProb signature(obj = "mmctestres"): ...
  confidenceLimits signature(obj = "mmctestres"): ...
  testResult signature(obj = "mmctestres"): ...
  summary.mmctestres signature(object = "mmctestres"): ...

Author(s)

  Axel Gandy and Georg Hahn

References


Examples

  fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
  i <- mmctSampler(fun,num=500, data=runif(500));
  a <- mmctest(h=hBH);
  a <- run(a, i, maxsteps=list(maxnum=1000000, undecided=10));
  # a is object of type "mmctestres" now
Description

Wrapper-Class for "mmctestInterfaceGeneric", takes a function, the number of hypotheses and returns derived object of class "mmctestInterfaceGeneric". Class provides a slot for additional data. The function f(ind,n,data) has to return n[i] new samples for each hypothesis ind[i] in vector "ind", where i=1...length(ind). The data stored in the data slot of class "mmctSampler" is also passed on to "f".

Objects from the Class

Objects can be created by calls of the form mmctSampler(f=...,num=...,data=...).

Slots

- f: Object of class "function"
- num: Object of class "numeric"
- data: Object of class "numeric"

Methods

- **getSamples** signature(obj="mmctSampler",ind="numeric",n="numeric"): ...
- **getNumber** signature(obj="mmctSampler"): ...

Author(s)

Axel Gandy and Georg Hahn

References


Examples

```r
fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
```
Methods for class `mmctSampler`, Package 'simctest'

Description

Constructor for class 'mmctSampler'.

Usage

```r
mmctSampler(f, num, data=NULL)
```

Arguments

- `f`: a function `f(ind,n,data)` which for every hypothesis `ind[i]` in vector "ind" returns `n[i]` new samples and returns the number of exceedances, where `i=1...length(ind)`. The data stored in the data slot of class "mmctSampler" is also passed on to "f".
- `num`: number of hypotheses.
- `data`: additional slot for data.

Methods

- `mmctSampler(f, num, data)` returns object of type 'mmctSampler' (derived from class 'mmctSamplerGeneric').

Examples

```r
fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
```

Generic class, has to be implemented as "mmctSampler".

Objects from the Class

This is a virtual class - no objects should be derived from it.

Methods

- `getSamples` signature(obj = "mmctSamplerGeneric",ind = "numeric",n = "numeric"): ...
- `getNumber` signature(obj = "mmctSamplerGeneric"): ...
Author(s)

Axel Gandy and Georg Hahn

References


Description

Function which shows current estimates of p-values.

Usage

pEstimate(obj)

Arguments

obj object of type ‘mmctestres’ or ‘mmctest’.

Methods

pEstimate(obj) works with object of type mmctestres or mmctest.

Examples

fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));
pEstimate(a);
Methods for class 'mmctestres' and 'mmctest', Package 'simctest'

Description

Function which returns empirical rejection probabilities. Threshold against e.g. 0.5 to obtain rejections (all rejProb>0.5 are rejected). Important: For usage in connection with thompson=TRUE (see the mmctest constructor).

Usage

rejProb(obj)

Arguments

obj

object of type 'mmctestres' or 'mmctest'.

Methods

rejProb(obj) works with object of type mmctestres or mmctest.

Examples

fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));
rejProb(a);

Methods for Function run in Package 'simctest'

Description

Starts a sampling algorithm

Usage

run(alg,gensample,maxsteps)

Arguments

alg

the sampling algorithm. An object of type "sampalg" or "mmctest".

gensample

a function returning the result of one resampling step (0=no rejection, 1= rejection of the null hypothesis), or an object of type "mmctSamplerGeneric" if alg="mmctest".

maxsteps

the maximal number of steps to take
Methods

alg = "sampalgPrecomp"  the algorithm to be used
alg = "sampalgonthefly"  the algorithm to be used
alg = "mmctest", gensample = "mmctSamplerGeneric"  the algorithm to be used

Examples

alg<-getalgonthefly()
res <- run(alg, function() runif(1)<0.2);
res

---

class"sampalg"

Description

Virtual base class for several sequential sampling algorithms.

Objects from the Class

This is a virtual class - no objects should be derived from it.

Slots

internal: Internal status data of the algorithm. Object of class "environment"

Methods

No methods defined with class "sampalg" in the signature.

Author(s)

Axel Gandy

See Also

sampalgonthefly, sampalgPrecomp
Description

A sequential sampling algorithm that creates its boundaries on the fly.

Objects from the Class

Objects can be created by calls of the form getalgonthefly(level, epsilon, halfspend).

Slots

  internal: Object of class "environment". Internal state of the algorithm. Do not access.

Extends

Class "sampalg", directly.

Methods

  run signature(alg = "sampalgonthefly"): ...

  getboundaryandprob signature(alg = "sampalgonthefly"): ...

Author(s)

Axel Gandy

References

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Re-
sampling Risk. JASA 104(488):1504-1511.

See Also

  sampalgPrecomp

Examples

  showClass("sampalgonthefly")
Description

Class returned as result from simctest and run.

Objects from the Class

Objects can be created by calls of the form new("sampalgontheflyres",...).

Slots

- porig: Object of class "numeric"
- U: Object of class "numeric"
- L: Object of class "numeric"
- ind: Object of class "numeric"
- preerr: Object of class "numeric"
- p.value: Object of class "numeric"
- steps: Object of class "numeric"
- pos: Object of class "numeric"
- alg: Object of class "sampalg"
- gen: Object of class "function"

Extends

Class "sampalgres", directly.

Methods

- `contalg` signature(data = "sampalgontheflyres"): ...

Author(s)

Axel Gandy

References


See Also

simctest, sampalgres
Class "sampalgPrecomp"

Description

A sampling algorithm that precomputes the boundaries

Objects from the Class

Objects can be created by calls to `getalgprecomp`

Slots

`internal`: internal state of the object. Do not access.

Extends

Class "sampalg", directly.

Author(s)

Axel Gandy

References


Examples

```
showClass("sampalgPrecomp")
```
**sampalgres-class**  

---

**Description**

Results returned by `run` - Internal.

**Objects from the Class**

Objects can be created by calls of the form `new("sampalgres",...).

**Slots**

- `p.value`: Object of class "numeric"
- `steps`: Object of class "numeric"
- `pos`: Object of class "numeric"
- `alg`: Object of class "sampalg"
- `gen`: Object of class "function"

**Methods**

- `confint` signature(object = "sampalgres",parm = "missing"): ...
- `contalg` signature(data = "sampalgres"): ...
- `getbounds` signature(data = "sampalgres"): ...
- `show` signature(object = "sampalgres"): ...

**Author(s)**

Axel Gandy

**References**


**Examples**

`showClass("sampalgres")`
simctest

Sequential implementation of Monte Carlo tests

Description

Wrapper function for convenient use of the sequential implementation of the Monte Carlo test.

Usage

```
simctest(gensample, level=0.05, epsilon=1e-3, maxsteps=1e4)
```

Arguments

- `gensample`: function that performs one sampling step. Returns 0 (sampled test statistic does not exceed the observation) or 1 (sampled test static exceeds the observation).
- `level`: level passed to `getalgonthefly`.
- `epsilon`: error bound `epsilon` passed to `getalgonthefly`.
- `maxsteps`: maximal number of steps to take.

Value

An object of class `sampalgres`.

Author(s)

Axel Gandy

References


Examples

```
# Example used in the above paper
dat <- matrix(nrow=5,ncol=7,byrow=TRUE,
              c(1,2,2,1,0,1,2,0,0,2,3,0,0,0,1,1,1,1,2,7,3,1,1,2,0,0,0,1,0,1,1,1,1,0,0))
loglikrat <- function(data){
  cs <- colSums(data)
  rs <- rowSums(data)
  mu <- outer(rs,cs)/sum(rs)
  2*sum(ifelse(data<=0.5, 0, data*log(data/mu)))
}
resample <- function(data){
  cs <- colSums(data)
  rs <- rowSums(data)
  n <- sum(rs)
  mu <- outer(rs,cs)/n/n
```
matrix(rmultinom(1,n,c(mu)),nrow=dim(data)[1],ncol=dim(data)[2])
}  
t <- loglikrat(dat);

# function to generate samples
gen <- function(){loglikrat(resample(dat))>=t}

#using simctest
simctest(gen,maxsteps=10000)

#now trying simctest.cont
res <- simctest(gen,maxsteps=500)
res
cont(res,20000)

summary.mmctestres-methods

Methods for class 'mmctestres' and 'mmctest', Package 'simctest'

Description

Function which shows current estimates of p-values.

Usage

## S3 method for class 'mmctestres'
summary(object,...)

Arguments

object object of type 'mmctestres' or 'mmctest'.
...
No further arguments needed. Listed only for compatibility with generic 'summary' method.

Methods

summary.mmctestres(object) works with object of type mmctestres or mmctest.

Examples

fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));
summary.mmctestres(a);
Description

Function which returns a list containing indices of rejected hypotheses (vector ‘rejected’), nonrejected hypotheses (vector ‘nonrejected’) and undecided hypotheses (vector ‘undecided’).

Usage

testResult(obj)

Arguments

obj 
object of type ‘mmctestres’ or ‘mmctest’.

Methods

testResult(obj) works with object of type mmctestres or mmctest.

Examples

fun <- function(ind,n,data) sapply(1:length(ind), function(i) sum(runif(n[i])<=data[ind[i]]));
i <- mmctSampler(fun,num=500,data=runif(500));
a <- mmctest(h=hBH);
a <- run(a, i, maxsteps=list(maxnum=1000000,undecided=10));
res <- testResult(a);
rejected <- res$rejected;
nonrejected <- res$nonrejected;
undecided <- res$undecided;
Index

*Topic classes
  getalgprecomp, 4
  mcpres-class, 12
  mmctest-class, 16
  mmctestres-class, 17
  mmctSampler-class, 19
  mmctSamplerGeneric-class, 20
  sampalg-class, 23
  sampalgonthefly-class, 24
  sampalgontheflyres-class, 25
  sampalgPrecomp-class, 26
  sampalgres-class, 27

*Topic methods
  confidenceLimits-methods, 2
  cont-methods, 4
  getbounds-methods, 5
  getL-methods, 6
  getSamples-methods, 7
  getU-methods, 7
  hBH-methods, 8
  hBonferroni-methods, 9
  hPC-methods, 9
  mmctest-methods, 16
  mmctSampler-methods, 20
  pEstimate-methods, 21
  rejProb-methods, 22
  run-methods, 22
  summary.mmctestres-methods, 29
  testResult-methods, 30

confidenceLimits
  (confidenceLimits-methods, 2
confidenceLimits,mmctest-method
  (confidenceLimits-methods, 2
confidenceLimits,mmctestres-method
  (confidenceLimits-methods, 2
confidenceLimits-methods, 2
confint, 3

confint (confint-methods, 3
confint,ANY,ANY-method
  (confint-methods, 3
confint,sampalgres,missing-method
  (confint-methods, 3
confint-methods, 3
cont, 3
cont (cont-methods, 4
cont,mmctestres-method (cont-methods, 4
cont,sampalgontheflyres-method
  (cont-methods, 4
cont,sampalgres-method (cont-methods, 4
cont-methods, 4
contalg,sampalgontheflyres-method
  (sampalgontheflyres-class, 25
contalg,sampalgres-method
  (sampalgres-class, 27

getalgonthefly, 28
getalgonthefly (getalgprecomp, 4
getalgprecomp, 4, 26
getbounds (getbounds-methods, 5
getbounds,sampalgontheflyres-method
  (getbounds-methods, 5
getbounds,sampalgres-method
  (getbounds-methods, 5
getbounds-methods, 5
getL (getL-methods, 6
getL,sampalgPrecomp-method
  (getL-methods, 6
getL-methods, 6
getNumber (getNumber-methods, 6
getNumber,mmctSampler-method
  (getNumber-methods, 6
getNumber,mmctSamplerGeneric-method
  (getNumber-methods, 6
getNumber-methods, 6
getSamples (getSamples-methods, 7
getSamples,mmctSampler-method
  (getSamples-methods, 7
getSamples, mmctSamplerGeneric-method
  (getSamples-methods), 7
getSamples-methods, 7
gU, (getU-methods), 7
getU, sampalgPrecomp-method
  (getU-methods), 7
getU-methods, 7
hBH, (hBH-methods), 8
hBH-methods, 8
hBonferroni, (hBonferroni-methods), 9
hBonferroni-methods, 9
hPC, (hPC-methods), 9
hPC-methods, 9
J, (mctest), 13
Jstar, (mctest), 13
mcp, 10
mcpres-class, 12
mctest, 13
mkdeltamid, 14
mmctest, (mmctest-methods), 16
mmctest-class, 16
mmctest-methods, 16
mmctestres-class, 17
mmctSampler, (mmctSampler-methods), 20
mmctSampler-class, 19
mmctSampler-methods, 20
mmctSamplerGeneric-class, 20
pEstimate, (pEstimate-methods), 21
pEstimate, mmctest-method
  (pEstimate-methods), 21
pEstimate, mmctestres-method
  (pEstimate-methods), 21
pEstimate-methods, 21
print.mctestres, (mctest), 13
rejProb, (rejProb-methods), 22
rejProb, mmctest-method
  (rejProb-methods), 22
rejProb, mmctestres-method
  (rejProb-methods), 22
rejProb-methods, 22
run, 3
run, (run-methods), 22
run, mmctest, mmctSamplerGeneric-method
  (run-methods), 22
run, sampalgonthefly, ANY-method
  (run-methods), 22
run, sampalgPrecomp, ANY-method
  (run-methods), 22
run-methods, 22
sampalg, 24, 26
sampalg-class, 23
sampalgonthefly, 4, 5, 23
sampalgonthefly
  (sampalgonthefly-class), 24
sampalgonthefly-class, 24
sampalgontheflyres-class, 25
sampalgPrecomp, 4, 5, 23, 24
sampalgPrecomp, (sampalgPrecomp-class), 26
sampalgPrecomp-class, 26
sampalgres, 3, 5, 28
sampalgres-class, 27
show, mcpres-method, (mcpres-class), 12
show, mmctestres-method
  (mmctestres-class), 17
show, sampalgres-method
  (sampalgres-class), 27
simctest, 25, 28
summary.mmctestres
  (summary.mmctestres-methods), 29
summary.mmctestres-mmctest-method
  (summary.mmctestres-methods), 29
summary.mmctestres-mmctestres-method
  (summary.mmctestres-methods), 29
summary.mmctestres-methods, 29
testResult, (testResult-methods), 30
testResult, mmctest-method
  (testResult-methods), 30
testResult, mmctestres-method
  (testResult-methods), 30
testResult-methods, 30