

Package ‘simctest’

March 19, 2012

Version 1.99-2

Date 2012-03-19

Title Safe implementation of Monte Carlo tests.

Author Axel Gandy <a.gandy@imperial.ac.uk> and Patrick Rubin-Delanchy
<patrick.rubin-delanchy@imperial.ac.uk>

Maintainer Axel Gandy <a.gandy@imperial.ac.uk>

Depends R (>= 2.2.0), methods, stats

Suggests

Description Monte Carlo computation of p-value with uniformly bounded
resampling risk, computation of the power of a Monte test.

License GPL (>= 2)

URL <http://www2.imperial.ac.uk/~agandy>

Repository CRAN

Date/Publication 2012-03-19 17:45:23

R topics documented:

confint-methods	2
cont-methods	3
getalgprecomp	3
getbounds-methods	4
getL-methods	5
getU-methods	5
mcp	6
mcpres-class	8
mkdeltamid	9
run-methods	10
sampalg-class	11

sampalgonthefly-class	11
sampalgontheflyres-class	12
sampalgPrecomp-class	13
sampalgres-class	14
simctest	15

Index	17
--------------	-----------

confint-methods *Methods for Function run in Package 'simctest'*

Description

Computes a confidence interval for the p-value

Usage

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

Arguments

object	An object of type <code>sampalgres</code> resulting from a previous call to <code>run</code> or <code>cont</code> .
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

```
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.

Examples

```
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

```
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 [sampalgPrecomp](#).

Author(s)

Axel Gandy

References

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Resampling Risk. To appear in JASA.

Examples

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

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

getbounds-methods *Methods for Function getbounds in Package 'simctest'*

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}`.

getL-methods

Methods for Function getL in Package 'simctest'

Description

Returns the lower boundary for the stopping rule

Usage

```
##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

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

getU-methods

Methods for Function getU in Package 'simctest'

Description

Returns the upper boundary for the stopping rule

Usage

```
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)
```

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

genstream	a function that returns a function that returns a random Bernoulli variable (each stream corresponds to a dataset. $0 = (T < t)$, $1 = (T \geq t)$ 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 $1/(20 + t)$ 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

Gandy, A. and Rubin-Delanchy, P. An algorithm to compute the power of Monte Carlo tests with guaranteed precision. arXiv:1110.1248v1

See Also

mkdeltamid

Examples

```
#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
```

mcpres-class

Class "mcpres"

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

Gandy, A. and Rubin-Delanchy, P. An algorithm to compute the power of Monte Carlo tests with guaranteed precision. [arXiv:1110.1248v1](https://arxiv.org/abs/1110.1248v1)

Examples

```
showClass("mcpres")
```

mkdeltamid *Function mkdeltamid in Package 'simctest'*

Description

Easy creation of adaptive delta function

Usage

```
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 reregion 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 $\text{deltamid}(M) \leq \text{length of CI}$.

Author(s)

Axel Gandy and Patrick Rubin-Delanchy

References

Gandy, A. and Rubin-Delanchy, P. An algorithm to compute the power of Monte Carlo tests with guaranteed precision. arXiv:1110.1248v1

Examples

```
## 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
```

run-methods

Methods for Function run in Package 'simctest'

Description

Starts a sampling algorithm

Usage

```
run(alg, gensample, maxsteps=1e4)
```

Arguments

alg	the sampling algorithm. An object of type sampalg.
gensample	a function returning the result of one resampling step (0=no rejection, 1=rejection of the null hypothesis)
maxsteps	the maximal number of steps to take

Methods

alg = "sampalgPrecomp" the algorithm to be used

alg = "sampalgonthefly" the algorithm to be used

Examples

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

sampalg-class *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](#)

sampalgonthefly-class *Class "smpalgonthefly"*

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 "[smpalg](#)", 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 Resampling Risk. To appear in JASA.

See Also

[sampalgPrecomp](#)

Examples

```
showClass("sampalgonthefly")
```

sampalgontheflyres-class

Class "sampalgontheflyres"

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"

preverr: 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 "[sompalgres](#)", directly.

Methods

contalg signature(data = "sompalgontheflyres"): ...

Author(s)

Axel Gandy

References

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Resampling Risk. To appear in JASA.

See Also

[simctest](#), [sompalgres](#)

Examples

```
showClass("sompalgontheflyres")
```

sompalgPrecomp-class *Class "sompalgPrecomp"*

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 "[sompalg](#)", directly.

Author(s)

Axel Gandy

References

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Resampling Risk. To appear in JASA.

Examples

```
showClass("sampalgPrecomp")
```

```
sampalgres-class      Class "sampalgres"
```

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

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Resampling Risk. To appear in JASA.

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

Gandy, A. (2009) Sequential Implementation of Monte Carlo Tests with Uniformly Bounded Re-sampling Risk. To appear in JASA.

Examples

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

Index

*Topic **classes**

- getalgprecomp, 3
- mcpres-class, 8
- sampalg-class, 11
- sampalgonthefly-class, 11
- sampalgontheflyres-class, 12
- sampalgPrecomp-class, 13
- sampalgres-class, 14

*Topic **methods**

- confint-methods, 2
- cont-methods, 3
- getbounds-methods, 4
- getL-methods, 5
- getU-methods, 5
- run-methods, 10

- confint, 2
- confint (confint-methods), 2
- confint, ANY, ANY-method
(confint-methods), 2
- confint, sampalgres, missing-method
(confint-methods), 2
- confint-methods, 2
- cont, 2
- cont (cont-methods), 3
- cont, sampalgontheflyres-method
(cont-methods), 3
- cont, sampalgres-method (cont-methods), 3
- cont-methods, 3
- contalg, sampalgontheflyres-method
(sampalgontheflyres-class), 12
- contalg, sampalgres-method
(sampalgres-class), 14
- getalgonthefly, 15
- getalgonthefly (getalgprecomp), 3
- getalgprecomp, 3, 13
- getbounds (getbounds-methods), 4
- getbounds, sampalgontheflyres-method
(getbounds-methods), 4

- getbounds, sampalgres-method
(getbounds-methods), 4
- getbounds-methods, 4
- getL (getL-methods), 5
- getL, sampalgPrecomp-method
(getL-methods), 5
- getL-methods, 5
- getU (getU-methods), 5
- getU, sampalgPrecomp-method
(getU-methods), 5
- getU-methods, 5
- mcp, 6
- mcpres-class, 8
- mkdeltamid, 9
- run, 2
- run (run-methods), 10
- run, sampalgonthefly-method
(run-methods), 10
- run, sampalgPrecomp-method
(run-methods), 10
- run-methods, 10
- sampalg, 11, 13
- sampalg-class, 11
- sampalgonthefly, 3, 4, 11
- sampalgonthefly
(sampalgonthefly-class), 11
- sampalgonthefly-class, 11
- sampalgontheflyres-class, 12
- sampalgPrecomp, 3, 4, 11, 12
- sampalgPrecomp (sampalgPrecomp-class),
13
- sampalgPrecomp-class, 13
- sampalgres, 2, 4, 13, 15
- sampalgres-class, 14
- show, mcpres-method (mcpres-class), 8
- show, sampalgres-method
(sampalgres-class), 14
- simctest, 13, 15