Package ‘npcp’

July 16, 2020

Type Package

Title Some Nonparametric CUSUM Tests for Change-Point Detection in Possibly Multivariate Observations

Version 0.2-2

Date 2020-07-16

Author Ivan Kojadinovic

Maintainer Ivan Kojadinovic <ivan.kojadinovic@univ-pau.fr>

Depends R (>= 3.5.0)

Imports stats

Suggests copula, sandwich

Description Provides nonparametric CUSUM tests for detecting changes in possibly serially dependent univariate or multivariate observations. Retrospective tests sensitive to changes in the expectation, the variance, the covariance, the autocovariance, the distribution function, Spearman's rho, Kendall's tau, Gini's mean difference, and the copula are provided, as well as a test for detecting changes in the distribution of independent block maxima (with environmental studies in mind). The package also contains a test sensitive to changes in the autocopula and a combined test of stationarity sensitive to changes in the distribution function and the autocopula. The latest additions are a closed-end sequential test based on empirical distribution functions that can be used for monitoring changes in the contemporary distribution of possibly serially dependent univariate or multivariate observations, and an open-end sequential test based on the retrospective CUSUM statistic that can be used for monitoring changes in the mean of possibly serially dependent univariate observations.

License GPL (>= 3) | file LICENCE

LazyLoad yes

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-07-16 15:00:03 UTC

R topics documented:

`bOptEmpProc` *Bandwidth Parameter Estimation*

Description

In the context of the standard CUSUM test based on the sample mean or in a particular empirical process setting, the following functions estimate the bandwidth parameter controlling the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the third reference. The function function `bOpt()` is called in the functions `cpMean()`, `cpVar()`, `cpGini()`, `cpAutocov()`, `cpCov()`, `cpTau()` and `detCpMean()` when `b` is set to `NULL`. The function function `bOptEmpProc()` is called in the functions `cpDist()`, `cpCopula()`, `cpAutocop()`, `stDistAutocop()` and `simCpDist()` when `b` is set to `NULL`.

Usage

```
bOpt(influ, weights = c("parzen", "bartlett"))

bOptEmpProc(x, m=5, weights = c("parzen", "bartlett"),
            L.method=c("max","median","mean","min"))
```

Arguments

`influ`	a numeric containing the relevant influence coefficients, which, in the case of the standard CUSUM test based on the sample mean, are simply the available observations; see also the last reference.
`x`	a data matrix whose rows are continuous observations.
`weights`	a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the third reference.

`m` a strictly positive integer specifying the number of points of the uniform grid on $(0,1)^d$ (where d is `ncol(x)`) involved in the estimation of the bandwidth parameter; see Section 5 of the third reference. The number of points of the grid is given by `m^ncol(x)` so that `m` needs to be decreased as d increases.

`L.method` a string specifying how the parameter `L` involved in the estimation of the bandwidth parameter is computed; see Section 5 of the third reference.

Details

The implemented approach results from an adaptation of the procedure described in the first two references (see also the references therein). The use of theses functions in a context different from that considered in the third or fourth reference may not be meaningful.

Acknowledgment: Part of the code of the function results from an adaptation of R code of C. Parmeter and J. Racine, itself an adaptation of Matlab code by A. Patton.

Value

A strictly positive integer.

References

D.N. Politis and H. White (2004), Automatic block-length selection for the dependent bootstrap, *Econometric Reviews* **23(1)**, pages 53-70.

D.N. Politis, H. White and A.J. Patton (2004), Correction: Automatic block-length selection for the dependent bootstrap, *Econometric Reviews* **28(4)**, pages 372-375.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

A. Bücher and I. Kojadinovic (2016), Dependent multiplier bootstraps for non-degenerate U-statistics under mixing conditions with applications, *Journal of Statistical Planning and Inference* **170** pages 83-105, http://arxiv.org/abs/1412.5875.

See Also

`cpDist()`, `cpCopula()`, `cpAutocop()`, `stDistAutocop()`, `cpMean()`, `cpVar()`, `cpGini()`, `cpAutocov()`, `cpCov()`, `cpTau()`, `seqCpMean` and `seqCpDist`.

`cpAutocop` *Test for Change-Point Detection in Univariate Observations Sensitive to Changes in the Autocopula*

Description

Nonparametric test for change-point detection particularly sensitive to changes in the autocopula of univariate continuous observations. Approximate p-values for the test statistic are obtained by means of a *multiplier* approach. Details can be found in the first reference.

Usage

```
cpAutocop(x, lag = 1, b = NULL, bivariate = FALSE,
          weights = c("parzen", "bartlett"), m = 5,
          N = 1000, init.seq = NULL, include.replicates = FALSE)
```

Arguments

`x`	a one-column matrix containing continuous observations.
`lag`	an integer specifying at which lag to consider the autocopula; the autocopula is a `(lag+1)`-dimensional copula.
`b`	strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. If set to `NULL`, `b` will be estimated using the function `bOptEmpProc()`; see the first reference.
`bivariate`	a logical specifying whether the test should focus only on the bivariate margin of the `(lag+1)`-dimensional autocopula obtained from the first and the last dimension.
`weights`	a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.
`m`	a strictly positive integer specifying the number of points of the uniform grid on $(0, 1)$ involved in the estimation of the bandwidth parameter; see Section 5 of the second reference.
`N`	number of multiplier replications.
`init.seq`	a sequence of independent standard normal variates of length `N * (nrow(x) -lag + 2 * (b -1))` used to generate dependent multiplier sequences.
`include.replicates`	a logical specifying whether the object of `class htest` returned by the function (see below) will include the multiplier replicates.

Details

The approximate p-value is computed as

$$(0.5 + \sum_{i=1}^{N} \mathbf{1}_{\{S_i \geq S\}})/(N + 1),$$

where S and S_i denote the test statistic and a multiplier replication, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments.

Value

An object of `class htest` which is a list, some of the components of which are

`statistic`	value of the test statistic.

`p.value`	corresponding approximate p-value.
`cvm`	the values of the `length(x)-lag-1` intermediate Cramér-von Mises change-point statistics; the test statistic is defined as the maximum of those.
`b`	the value of parameter `b`.

Note

This is a tests for a continuous univariate time series.

References

A. Bücher, J.-D. Fermanian and I. Kojadinovic (2019), Combining cumulative sum change-point detection tests for assessing the stationarity of univariate time series, *Journal of Time Series Analysis* **40**, pages 124-150, http://arxiv.org/abs/1709.02673.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

See Also

`cpAutocov()` for a related test based on the autocovariance.

Examples

```
## AR1 example
n <- 200
k <- n/2 ## the true change-point
x <- matrix(c(arima.sim(list(ar = -0.5), n = k),
              arima.sim(list(ar = 0.5), n = n - k)))
cp <- cpAutocop(x)
cp
## Estimated change-point
which(cp$cvm == max(cp$cvm))

## AR2 example
n <- 200
k <- n/2 ## the true change-point
x <- matrix(c(arima.sim(list(ar = c(0,-0.5)), n = k),
              arima.sim(list(ar = c(0,0.5)), n = n - k)))
cpAutocop(x)
cpAutocop(x, lag = 2)
cpAutocop(x, lag = 2, bivariate = TRUE)
```

`cpBlockMax` *Nonparametric Tests for Change-Point Detection in the Distribution of Independent Block Maxima*

Description

Nonparametric tests for change-point detection in the distribution of independent block maxima based either on the probability weighted moment method (see the second reference) or on the generalized probability weighted moment method (see the first reference) for estimating the parameters of the generalized extreme value (GEV) distribution. It is assumed that the block maxima are independent and that their unknown distribution functions (d.f.s) are continuous, but not necessarily that they are GEV distributed. Three statistics are computed. Under the assumption that the block maxima are GEV distributed, these are statistics particularly sensitive to changes in the location, scale and shape parameters of the GEV. Details can be found in third reference.

Usage

```
cpBlockMax(x, method = c("pwm", "gpwm"), r=10)
```

Arguments

`x`	a numeric vector representing independent block maxima whose unknown d.f.s are assumed continuous.
`method`	a string specifying how statistics will be defined; can be either "pwm" (the probability weighted moment method) or "gpwm" (the generalized probability weighted moment method). The method "pwm" is suggested for climate block maxima that are typically not too heavy tailed, more precisely, whose distributions are in the maximum domains of attraction of GEV distributions with shape parameters smaller than a half. The method "gpwm" should be preferred otherwise.
`r`	strictly positive integer specifying the set of breakpoints that will be tested; more precisely, starting from the initial sample of block maxima, the tests compare subsamples formed by the `k` first maxima and `n-k` last maxima for `k` in the set `{r,...,n-r}`, where `n` is the sample size.

Details

Approximate p-values are computed from the estimated asymptotic null distributions, which involve the Kolmogorov distribution. The latter is dealt with reusing code from the `ks.test()` function; credit to RCore.

Value

An object of `class` `htest` which is a list, some of the components of which are

`statistic`	value of the three test statistics.
`pvalues`	corresponding approximate p-values.

stats.loc	the values of the n-(2 * r -1) intermediate change-point statistics sensitive to changes in the location; the first test statistic is defined as the maximum of those.
stats.scale	the values of the n-(2 * r -1) intermediate change-point statistics sensitive to changes in the scale; the second test statistic is defined as the maximum of those.
stats.shape	the values of the n-(2 * r -1) intermediate change-point statistics sensitive to changes in the shape; the third test statistic is defined as the maximum of those.

Note

The tests were derived under the assumption of block maxima with continuous d.f., which implies that ties occur with probability zero. A way to deal with ties based on randomization is proposed in the third reference.

References

J. Diebolt, A. Guillou, P. Naveau and P. Ribereau (2008), Improving probability-weighted moment methods for the generalized extreme-value distribution, *REVSTAT* **6**, pages 33-50.

J.R.M. Hosking, J.R. Wallis and E.F. Wood (1985), Estimation of the generalized extreme-value distribution by the method of probability-weighted moments, *Technometrics* **27**, pages 251-261.

I. Kojadinovic and P. Naveau (2017), Nonparametric tests for change-point detection in the distribution of block maxima based on probability weighted moments, *Extremes* **20:2**, pages 417-450, http://arxiv.org/abs/1507.06121.

See Also

cpDist() for a related test based on the empirical d.f.

Examples

```
## Not run:
require(evd)
n <- 100
k <- 50 ## the true change-point

## Change in the shape parameter of a GEV
x <- rgev(k,loc=0,scale=1,shape=-0.8)
y <- rgev(k,loc=0,scale=1,shape=0.4)
cp <- cpBlockMax(c(x,y))
cp
## Estimated change-point
which(cp$stats.shape == max(cp$stats.shape))

## Change in the scale parameter of a GEV
x <- rgev(k,loc=0,scale=0.5,shape=0)
y <- rgev(k,loc=0,scale=1,shape=0)
cp <- cpBlockMax(c(x,y))
cp
## Estimated change-point
which(cp$stats.scale == max(cp$stats.scale))
```

```
## Change in the location parameter of a GEV
x <- rgev(k,loc=0,scale=1,shape=0)
y <- rgev(k,loc=0.5,scale=1,shape=0)
cp <- cpBlockMax(c(x,y))
cp
## Estimated change-point
which(cp$stats.loc == max(cp$stats.loc))
## End(Not run)
```

---

cpCopula *Test for Change-Point Detection in Multivariate Observations Sensitive to Changes in the Copula*

Description

Nonparametric test for change-point detection particularly sensitive to changes in the copula of multivariate continuous observations. The observations can be serially independent or dependent (strongly mixing). Approximate p-values for the test statistic are obtained by means of a *multiplier* approach. Details can be found in the first reference.

Usage

```
cpCopula(x, method = c("seq", "nonseq"), b = NULL,
         weights = c("parzen", "bartlett"), m = 5,
         L.method=c("max","median","mean","min"),
         N = 1000, init.seq = NULL, include.replicates = FALSE)
```

Arguments

`x` a data matrix whose rows are multivariate continuous observations.

`method` a string specifying the simulation method for generating multiplier replicates of the test statistic; can be either `"seq"` (the 'check' approach in the first reference) or `"nonseq"` (the 'hat' approach in the first reference). The 'check' approach appears to lead to better behaved tests in the case of samples of moderate size. The 'hat' approach is substantially faster.

`b` strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. The value 1 will create i.i.d. multiplier sequences suitable for serially independent observations. If set to `NULL`, `b` will be estimated from `x` using the function `bOptEmpProc()`; see the procedure described in Section 5 of the second reference.

`weights` a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.

m
: a strictly positive integer specifying the number of points of the uniform grid on $(0,1)^d$ (where d is `ncol(x)`) involved in the estimation of the bandwidth parameter; see Section 5 of the third reference. The number of points of the grid is given by `m^ncol(x)` so that `m` needs to be decreased as d increases.

L.method
: a string specifying how the parameter L involved in the estimation of the bandwidth parameter is computed; see Section 5 of the second reference.

N
: number of multiplier replications.

init.seq
: a sequence of independent standard normal variates of length `N * (nrow(x) + 2 * (b -1))` used to generate dependent multiplier sequences.

include.replicates
: a logical specifying whether the object of `class` `htest` returned by the function (see below) will include the multiplier replicates.

Details

The approximate p-value is computed as

$$(0.5 + \sum_{i=1}^{N} \mathbf{1}_{\{S_i \geq S\}})/(N+1),$$

where S and S_i denote the test statistic and a multiplier replication, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments.

Value

An object of `class` `htest` which is a list, some of the components of which are

statistic
: value of the test statistic.

p.value
: corresponding approximate p-value.

cvm
: the values of the `nrow(x)-1` intermediate Cramér-von Mises change-point statistics; the test statistic is defined as the maximum of those.

b
: the value of parameter `b`.

Note

These tests were derived under the assumption of continuous margins.

References

A. Bücher, I. Kojadinovic, T. Rohmer and J. Segers (2014), Detecting changes in cross-sectional dependence in multivariate time series, *Journal of Multivariate Analysis* **132**, pages 111-128, http://arxiv.org/abs/1206.2557.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

See Also

cpRho() for a related test based on Spearman's rho, cpTau() for a related test based on Kendall's tau, cpDist() for a related test based on the multivariate empirical d.f., bOptEmpProc() for the function used to estimate b from x if b = NULL.

Examples

```
## Not run:
require(copula)
n <- 100
k <- 50 ## the true change-point
u <- rCopula(k, gumbelCopula(1.5))
v <- rCopula(n - k, gumbelCopula(3))
x <- rbind(u,v)
cp <- cpCopula(x, b = 1)
cp
## Estimated change-point
which(cp$cvm == max(cp$cvm))
## End(Not run)
```

---

cpDist *Test for Change-Point Detection in Possibly Multivariate Observations Sensitive to Changes in the Distribution Function*

Description

Nonparametric test for change-point detection based on the (multivariate) empirical distribution function. The observations can be continuous univariate or multivariate, and serially independent or dependent (strongly mixing). Approximate p-values for the test statistics are obtained by means of a *multiplier* approach. The first reference treats the serially independent case while details about the serially dependent case can be found in second and third references.

Usage

```
cpDist(x, statistic = c("cvmmax", "cvmmean", "ksmax", "ksmean"),
       method = c("nonseq", "seq"), b = NULL, gamma = 0,
       delta = 1e-4, weights = c("parzen", "bartlett"),
       m = 5, L.method=c("max","median","mean","min"),
       N = 1000, init.seq = NULL, include.replicates = FALSE)
```

Arguments

x	a data matrix whose rows are continuous observations.
statistic	a string specifying the statistic whose value and p-value will be displayed; can be either "cvmmax" or "cvmmean" (the maximum or average of the nrow(x)-1 intermediate Cramér-von Mises statistics), or "ksmax" or "ksmean" (the maximum or average of the nrow(x)-1 intermediate Kolmogorov-Smirnov statistics); see Section 3 in the first reference. The four statistics and the corresponding p-values are computed at each execution.

method	a string specifying the simulation method for generating multiplier replicates of the test statistic; can be either "nonseq" (the 'check' approach in the first reference) or "seq" (the 'hat' approach in the first reference). The 'check' approach appears to lead to better behaved tests and is recommended.
b	strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. The value 1 will create i.i.d. multiplier sequences suitable for serially independent observations. If set to NULL, b will be estimated from x using the function bOptEmpProc(); see the procedure described in Section 5 of the second reference.
gamma	parameter between 0 and 0.5 appearing in the definition of the weight function used in the detector function.
delta	parameter between 0 and 1 appearing in the definition of the weight function used in the detector function.
weights	a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.
m	a strictly positive integer specifying the number of points of the uniform grid on $(0,1)^d$ (where d is ncol(x)) involved in the estimation of the bandwidth parameter; see Section 5 of the third reference. The number of points of the grid is given by m^ncol(x) so that m needs to be decreased as d increases.
L.method	a string specifying how the parameter L involved in the estimation of the bandwidth parameter is computed; see Section 5 of the second reference.
N	number of multiplier replications.
init.seq	a sequence of independent standard normal variates of length N * (nrow(x) + 2 * (b -1)) used to generate dependent multiplier sequences.
include.replicates	a logical specifying whether the object of class htest returned by the function (see below) will include the multiplier replicates.

Details

The approximate p-value is computed as

$$(0.5 + \sum_{i=1}^{N} \mathbf{1}_{\{S_i \geq S\}})/(N+1),$$

where S and S_i denote the test statistic and a multiplier replication, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments.

Value

An object of class htest which is a list, some of the components of which are

statistic	value of the test statistic.

p.value	corresponding approximate p-value.
cvm	the values of the nrow(x)-1 intermediate Cramér-von Mises change-point statistics.
ks	the values of the nrow(x)-1 intermediate Kolmogorov-Smirnov change-point statistics.
all.statistics	the values of all four test statistics.
all.p.values	the corresponding p-values.
b	the value of parameter b.

Note

Note that when the observations are continuous univariate and serially independent, independent realizations of the tests statistics under the null hypothesis of no change in the distribution can be obtained by simulation; see Section 4 in the first reference.

References

M. Holmes, I. Kojadinovic and J-F. Quessy (2013), Nonparametric tests for change-point detection à la Gombay and Horváth, *Journal of Multivariate Analysis* **115**, pages 16-32.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

A. Bücher, J.-D. Fermanian and I. Kojadinovic (2019), Combining cumulative sum change-point detection tests for assessing the stationarity of univariate time series, *Journal of Time Series Analysis* **40**, pages 124-150, http://arxiv.org/abs/1709.02673.

See Also

cpCopula() for a related test based on the empirical copula, cpRho() for a related test based on Spearman's rho, cpTau() for a related test based on Kendall's tau, bOptEmpProc() for the function used to estimate b from x if b = NULL, seqCpDist for the corresponding sequential test.

Examples

```
## A univariate example
n <- 100
k <- 50 ## the true change-point
y <- rnorm(k)
z <- rexp(n-k)
x <- matrix(c(y,z))
cp <- cpDist(x, b = 1)
cp

## All statistics
cp$all.statistics
## Corresponding p.values
cp$all.p.values
```

```
## Estimated change-point
which(cp$cvm == max(cp$cvm))
which(cp$ks == max(cp$ks))

## A very artificial trivariate example
## with a break in the first margin
n <- 100
k <- 50 ## the true change-point
y <- rnorm(k)
z <- rnorm(n-k, mean = 2)
x <- cbind(c(y,z),matrix(rnorm(2*n), n, 2))
cp <- cpDist(x, b = 1)
cp

## All statistics
cp$all.statistics
## Corresponding p.values
cp$all.p.values

## Estimated change-point
which(cp$cvm == max(cp$cvm))
which(cp$ks == max(cp$ks))
```

---

cpRho *Test for Change-Point Detection Based on Spearman's Rho*

Description

Nonparametric test for change-point detection particularly sensitive to changes in Spearman's rho in multivariate time series. The observations can be serially independent or dependent (strongly mixing). Approximate p-values for the test statistic are obtained by means of a *multiplier* approach or by estimating the asymptotic null distribution. Details can be found in first reference.

Usage

```
cpRho(x, method = c("mult", "asym.var"),
      statistic = c("pairwise", "global"),
      b = NULL, weights = c("parzen", "bartlett"),
      N = 1000, init.seq = NULL, include.replicates = FALSE)
```

Arguments

`x`	a data matrix whose rows are multivariate continuous observations.
`method`	a string specifying the method for computing the approximate p-value for the test statistic; can be either `"mult"` (the multiplier approach 'tilde' in the first reference) or `"asym.var"` (the approach based on the estimation of the asymptotic null distribution of the test statistic described in the first reference). The 'mult' approach appears to lead to better behaved tests.

statistic
: a string specifying the test statistic; can be either "pairwise" (the statistic $S_{n,3}$ in the first reference) or "global" (the statistic $S_{n,1}$ in the first reference).

b
: strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. The value 1 will create i.i.d. multiplier sequences suitable for serially independent observations. If set to NULL, b will be estimated from x using the procedure described in the first reference.

weights
: a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.

N
: number of multiplier replications.

init.seq
: a sequence of independent standard normal variates of length N * (nrow(x) + 2 * (b -1)) used to generate dependent multiplier sequences.

include.replicates
: a logical specifying whether the object of class htest returned by the function (see below) will include the multiplier replicates, if generated.

Details

When method == "mult", the approximate p-value is computed as

$$(0.5 + \sum_{i=1}^{N} \mathbf{1}_{\{S_i \geq S\}})/(N + 1),$$

where S and S_i denote the test statistic and a multiplier replication, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments.

When method == "asym.var", the approximate p-value is computed from the estimated asymptotic null distribution, which involves the Kolmogorov distribution. The latter is dealt with reusing code from the ks.test() function; credit to RCore.

Value

An object of class htest which is a list, some of the components of which are

statistic
: value of the test statistic.

p.value
: corresponding approximate p-value.

rho
: the values of the nrow(x)-1 intermediate change-point statistics; the test statistic is defined as the maximum of those.

b
: the value of parameter b.

Note

These tests were derived under the assumption of continuous margins.

References

I. Kojadinovic, J-F. Quessy and T. Rohmer (2016), Testing the constancy of Spearman's rho in multivariate time series, *Annals of the Institute of Statistical Mathematics* **68:5**, pages 929-954, http://arxiv.org/abs/1407.1624.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

See Also

`cpTau()` for a related test based on Kendall's tau, `cpDist()` for a related test based on the multivariate empirical d.f., `cpCopula()` for a related test based on the empirical copula.

Examples

```
## Not run:
require(copula)
n <- 100
k <- 50 ## the true change-point
u <- rCopula(k,gumbelCopula(1.5))
v <- rCopula(n-k,gumbelCopula(3))
x <- rbind(u,v)
cp <- cpRho(x, b = 1)
cp
## Estimated change-point
which(cp$rho == max(cp$rho))
## End(Not run)
```

---

cpU *Some CUSUM Tests for Change-Point Detection Based on U-statistics*

Description

Nonparametric CUSUM tests for change-point detection particularly sensitive to changes in certain quantities that can be estimated using one-sample U-statistics of order one or two. So far, the quantities under consideration are the expectation (thus corresponding to the standard CUSUM test based on the sample mean), the variance, Gini's mean difference, the autocovariance at a specified lag, the covariance for bivariate data and Kendall's tau for multivariate data. The observations can be serially independent or dependent (strongly mixing). Approximate p-values for the test statistic are obtained by means of a *multiplier* approach or by estimating the asymptotic null distribution. Details can be found in the first reference.

Usage

```
cpMean(x, method = c("nonseq", "seq", "asym.var"),
       b = NULL, weights = c("parzen", "bartlett"),
       N = 1000, init.seq = NULL, include.replicates = FALSE)
```

```
cpVar(x, method = c("nonseq", "seq", "asym.var"),
      b = NULL, weights = c("parzen", "bartlett"),
      N = 1000, init.seq = NULL, include.replicates = FALSE)

cpGini(x, method = c("nonseq", "seq", "asym.var"),
       b = NULL, weights = c("parzen", "bartlett"),
       N = 1000, init.seq = NULL, include.replicates = FALSE)

cpAutocov(x, lag = 1, method = c("nonseq", "seq", "asym.var"),
          b = NULL, weights = c("parzen", "bartlett"),
          N = 1000, init.seq = NULL, include.replicates = FALSE)

cpCov(x, method = c("nonseq", "seq", "asym.var"),
      b = NULL, weights = c("parzen", "bartlett"),
      N = 1000, init.seq = NULL, include.replicates = FALSE)

cpTau(x, method = c("seq", "nonseq", "asym.var"),
      b = NULL, weights = c("parzen", "bartlett"),
      N = 1000, init.seq = NULL, include.replicates = FALSE)
```

Arguments

`x` a numeric vector or a data matrix containing continuous observations.

`lag` an integer specifying at which lag to consider the autocovariance.

`method` a string specifying the method for computing the approximate p-value for the test statistic; can be either `"seq"` (the 'check' approach in the first reference), `"nonseq"` (the 'hat' approach in the first reference), or `"asym.var"` (the approach based on the estimation of the asymptotic null distribution of the test statistic described in the first reference). The 'seq' approach appears overall to lead to better behaved tests for `cpTau()`. More experiments are necessary for the other functions.

`b` strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. The value 1 will create i.i.d. multiplier sequences suitable for serially independent observations. If set to `NULL`, `b` will be estimated from `x` using the procedure described in the first reference.

`weights` a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.

`N` number of multiplier replications.

`init.seq` a sequence of independent standard normal variates of length `N * (nrow(x) + 2 * (b -1))` used to generate dependent multiplier sequences.

`include.replicates`
a logical specifying whether the object of `class` `htest` returned by the function (see below) will include the multiplier replicates, if generated.

Details

When `method` is either `"seq"` or `"nonseq"`, the approximate p-value is computed as

$$(0.5 + \sum_{i=1}^{N} \mathbf{1}_{\{S_i \geq S\}})/(N+1),$$

where S and S_i denote the test statistic and a multiplier replication, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments.

When `method = "asym.var"`, the approximate p-value is computed from the estimated asymptotic null distribution, which involves the Kolmogorov distribution. The latter is dealt with reusing code from the `ks.test()` function; credit to RCore.

Value

An object of `class` `htest` which is a list, some of the components of which are

`statistic`	value of the test statistic.
`p.value`	corresponding approximate p-value.
`u`	the values of the `nrow(x)-3` intermediate change-point statistics; the test statistic is defined as the maximum of those.
`b`	the value of parameter `b`.

References

A. Bücher and I. Kojadinovic (2016), Dependent multiplier bootstraps for non-degenerate U-statistics under mixing conditions with applications, *Journal of Statistical Planning and Inference* **170**, pages 83-105, http://arxiv.org/abs/1412.5875.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

See Also

`cpDist()` for a related test based on the multivariate empirical d.f., `cpCopula()` for a related test based on the empirical copula, `cpAutocop()` for a related test based on the empirical autocopula, `cpRho()` for a related test based on Spearman's rho, `bOpt()` for the function used to estimate `b` from `x` if `b = NULL` and `seqCpMean` for related sequential tests that can be used for online monitoring.

Examples

```
## The standard CUSUM test based on the sample mean
cp <- cpMean(c(rnorm(50), rnorm(50, mean=1)), b=1)
cp
## Estimated change-point
which(cp$statistics == cp$statistic)

## Testing for changes in the autocovariance
```

```
n <- 200
k <- n/2 ## the true change-point
x <- c(arima.sim(list(ar = -0.5), n = k),
       arima.sim(list(ar = 0.5), n = n - k))
cp <- cpAutocov(x)
cp
## Estimated change-point
which(cp$u == cp$statistic)
## Another example
x <- c(arima.sim(list(ar = c(0,-0.5)), n = k),
       arima.sim(list(ar = c(0,0.5)), n = n - k))
cpAutocov(x)
cp <- cpAutocov(x, lag = 2)
cp
## Estimated change-point
which(cp$u == cp$statistic)

## Not run:
## Testing for changes in Kendall's tau
require(copula)
n <- 100
k <- 50 ## the true change-point
u <- rCopula(k,gumbelCopula(1.5))
v <- rCopula(n-k,gumbelCopula(3))
x <- rbind(u,v)
cp <- cpTau(x)
cp
## Estimated change-point
which(cp$u == cp$statistic)

## Testing for changes in the covariance
cp <- cpCov(x)
cp
## Estimated change-point
which(cp$u == cp$statistic)
## End(Not run)
```

`quantiles` *Estimated Quantiles for the Open-end Nonparametric Sequential Change-Point Detection Tests Sensitive to Changes in the Mean*

Description

Estimated quantiles for the open-end nonparametric sequential change-point detection tests in `seqCpMean` sensitive to changes in the mean. More details can be found in the references below.

Usage

```
data("quantiles")
```

Format

`list` of 5 arrays containing the estimated 90%, 95% and 99% quantiles necessary for carrying out the sequential tests in `seqCpMean` described in the references below.

References

J. Gösmann, T. Kley and H. Dette (2020), A new approach for open-end sequential change point monitoring, 49 pages, http://arxiv.org/abs/1906.03225.

M. Holmes and I. Kojadinovic (2020), Open-end nonparametric sequential change-point detection based on the retrospective CUSUM statistic, 41 pages, available on the arXiv.

L. Horváth, M. Hušková, P. Kokoszka and J. Steinebach (2004). Monitoring changes in linear models. *Journal of Statistical Planning and Inference* **126**, pages 225-251.

Examples

```
data("quantiles")
str(quantiles)
```

---

seqCpDist — *Closed-end Sequential Test for Change-Point Detection in Possibly Multivariate Time Series Sensitive to Changes in the Contemporary Distribution Function*

Description

Closed-end nonparametric sequential test for change-point detection based on the (multivariate) empirical distribution function. The observations can be continuous univariate or multivariate, and serially independent or dependent (strongly mixing). To carry out the test, four steps are required. The first step consists of simulating under the null many trajectories of the detector function. The second step consists of estimating a piecewise constant threshold function from these trajectories. The third step consists of computing the detector function from the data to be monitored. The fourth and last step consists of comparing the detector function with the estimated threshold function. Each of these steps corresponds to one of the functions in the usage section below. The current implementation is preliminary and not optimized for real-time monitoring (but could still be used for that). If the observations to be monitored are univariate and can be assumed serially independent, the simulation of the trajectories of the detector functions can be carried out using Monte Carlo simulation. In all other cases, the test relies on a *dependent multiplier bootstrap*. Details can be found in the second reference.

Usage

```
simCpDist(x.learn = NULL, m = NULL, n, gamma = 0.25, delta = 1e-4,
          B = 1000, method = c("sim", "mult"), b = NULL,
          weights = c("parzen", "bartlett"), g = 5,
          L.method = c("max","median","mean","min"))
```

```
threshCpDist(sims, p = 1, alpha = 0.05, type = 7)

detCpDist(x.learn, x, gamma = 0.25, delta = 1e-4)

monCpDist(det, thresh, statistic = c("mac", "mmc", "mmk", "mk", "mc"),
          plot = TRUE)
```

Arguments

`x.learn`	a data matrix whose rows are continuous observations, representing the learning sample.
`m`	a strictly positive integer specifying the size of the learning sample if `x.learn` is not specified; the latter implies that the observations are univariate and assumed to be independent; if `m` is not specified, it is taken equal to `nrow(x.learn)`.
`n`	a strictly positive integer specifying the monitoring horizon; the monitoring period is `m+1`, ..., `n`.
`gamma`	a real parameter between 0 and 0.5 appearing in the definition of the weight function used in the detector function.
`delta`	a real parameter between 0 and 1 appearing in the definition of the weight function used in the detector function.
`B`	the number of trajectories of the detector function to simulate under the null.
`method`	a string specifying the trajectory simulation method; can be either `"sim"` (Monte Carlo simulation – only in the univariate case under the assumption of serial independence) or `"mult"` (the dependent multiplier bootstrap).
`b`	strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the first reference. The value 1 will create i.i.d. multiplier sequences suitable for serially independent observations. If set to `NULL`, `b` will be estimated from `x.learn` using the function `bOptEmpProc()`; see the procedure described in Section 5 of the first reference.
`weights`	a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the first reference.
`g`	a strictly positive integer specifying the number of points of the uniform grid on $(0,1)^d$ (where d is `ncol(x)`) involved in the estimation of the bandwidth parameter; see Section 5 of the first reference. The number of points of the grid is given by `g^ncol(x)` so that `g` needs to be decreased as d increases.
`L.method`	a string specifying how the parameter L involved in the estimation of the bandwidth parameter is computed; see Section 5 of the first reference.
`sims`	an object of class `sims.cpDist` containing simulated trajectories of the detector function under the null.
`p`	a strictly positive integer specifying the number of steps of the piece constant threshold function; `p` should not be taken too large (say, smaller than 4) if `method = "mult"`.
`alpha`	the value of the desired significance level for the sequential test.

type	an integer between 1 and 9 selecting one of the nine quantile algorithms detailed in the help of the function quantile().
x	a data matrix whose rows are continuous observations corresponding to the new observations to be monitored for a change in contemporary distribution.
det	an object of class det.cpDist representing a detector function computed using detCpDist().
thresh	an object of class thresh.cpDist representing a threshold function estimated using threshCpDist().
statistic	a string specifying the statistic/detector to be used for the monitoring; can be either "mac", "mmc", "mmk", "mc" or "mk"; the last letter specifies whether it is a Cramér-von Mises-like statistic (letter "c") or a Kolmogorov-Smirnov-like statistic (letter "k"); the letters before specify the type of aggregation steps used to compute the detectors ("m" for maximum, "a" for average); "mac" corresponds to the detector $T_{m,q}$ in the second reference, "mmc" to the detector $S_{m,q}$, "mmk" to the detector $R_{m,q}$, "mc" to the detector Q_m and "mk" to the detector P_m.
plot	logical indicating whether the monitoring should be plotted.

Details

The testing procedure is described in detail in the second reference.

Value

All functions return lists whose components have explicit names. The function monCpDist() in particular returns a list whose components are

alarm	a logical indicating whether the detector function has exceeded the threshold function.
time.alarm	an integer corresponding to the time at which the detector function has exceeded the threshold function or NA.
times.max	a vector of times at which the successive detectors "mmc" (if statistic = "mac" or statistic = "mmc") or "mmk" (if statistic = "mmk") have reached their maximum; a vector of NA's if statistic = "mc" or statistic = "mk"; this sequence of times can be used to estimate the time of change from the time of alarm.
time.change	an integer giving the estimated time of change if alarm is TRUE; the latter is simply the value in times.max which corresponds to time.alarm.

Note

This is a test for continuous (multivariate) time series.

References

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

I. Kojadinovic and G. Verdier (2020), Nonparametric sequential change-point detection for multivariate time series based on empirical distribution functions, 52 pages, http://arxiv.org/abs/2004.12322.

See Also

see cpDist() for the corresponding a posteriori (offline) test.

Examples

```
## Not run:
## Example of montoring for the period m+1, ..., n
m <- 100 # size of the learning sample
n <- 150 # monitoring horizon

## The learning sample
set.seed(123)
x.learn <- matrix(rnorm(m))

## New observations with a large change in mean
## to simulate monitoring for the period m+1, ..., n
k <- 125 ## the true change-point
x <- matrix(c(rnorm(k-m), rnorm(n-k, mean = 2)))

## Step 1: Simulation of B trajectories of the detector functions under the null
B <- 1e4

## Under the assumption of serial independence
## (no need to specify the learning sample)
traj.sim <- simCpDist(m = m, n = n, B = B, method = "sim")

## Without the assumption of serial independence
## (the learning sample is compulsory; the larger it is, the better;
## the monitoring horizon n should not be too large)
traj.mult <- simCpDist(x.learn = x.learn, n = n, B = B, method = "mult")

## Step 2: Compute threshold functions with p steps
p <- 2
tf.sim <- threshCpDist(traj.sim, p = p) # p can be taken large
                                        # if B is very large
tf.mult <- threshCpDist(traj.mult, p = p) # p should not be taken too
                                          # large unless both m and B
                                          # are very large

## Step 3: Compute the detectors for the monitoring period m+1, ... , n
det <- detCpDist(x.learn = x.learn, x = x)

## Step 4: Monitoring

## Simulate the monitoring with the first threshold function
monCpDist(det, tf.sim)
```

```
## Simulate the monitoring with the second threshold function
monCpDist(det, tf.mult)

## Simulate the monitoring with the first threshold function
## and another detector function
monCpDist(det, tf.sim, statistic = "mmk")

## Alternative steps 3 and 4:

## Compute the detectors for the monitoring period m+1, ... , m+20 only
det <- detCpDist(x.learn = x.learn, x = x[1:20,,drop = FALSE])

## Simulate the monitoring with the first threshold function
monCpDist(det, tf.sim)

## Simulate the monitoring with the second threshold function
monCpDist(det, tf.mult)

## End(Not run)
```

---

seqCpMean — *Open-end Nonparametric Sequential Change-Point Detection Test for Univariate Time Series Sensitive to Changes in the Mean*

Description

Open-end nonparametric sequential test for change-point detection based on the retrospective CUSUM statistic. The observations need to be univariate but can be serially dependent. To carry out the test, two steps are required. The first step consists of computing a detector function. The second step consists of comparing the detector function to a suitable constant threshold function. Each of these steps corresponds to one of the functions in the usage section below. The current implementation is preliminary and not optimized for real-time monitoring (but could still be used for that). Details can be found in the third reference.

Usage

```
detCpMean(x.learn, x, sigma = NULL, b = NULL,
          weights = c("parzen", "bartlett"))

monCpMean(det, statistic = c("t", "s", "r", "e", "cs"), eta = 0.001,
          gamma = 0.45, alpha = 0.05, sigma = NULL, plot = TRUE)
```

Arguments

`x.learn`	a numeric vector representing the learning sample.
`x`	a numeric vector representing the observations collected after the beginning of the monitoring for a change in mean.

sigma — an estimate of the long-run variance of the time series of which x.learn is a stretch. If set to NULL, sigma will be estimated using an approach similar to those described in the fourth reference.

b — strictly positive integer specifying the value of the bandwidth for the estimation of the long-run variance if sigma is not provided. If set to NULL, b will be estimated from x.learn using the function bOpt().

weights — a string specifying the kernel for creating the weights used for the estimation of the long-run variance if sigma is not provided; see Section 5 of the first reference.

det — an object of class det.cpMean representing a detector function computed using detCpMean().

statistic — a string specifying the statistic/detector to be used for the monitoring; can be either "t", "s", "r", "e" or "cs"; "t" corresponds to the detector T_m in the third reference, "s" to the detector S_m, "r" to the detector R_m, "e" to the detector E_m and "cs" to so-called ordinary CUSUM detector denoted by Q_m in the third reference. Note that the detector E_m was proposed in the second reference.

eta — a real parameter whose role is described in detail in the third reference.

gamma — a real parameter that can improve the power of the sequential test at the beginning of the monitoring; possible values are 0, 0.1, 0.25, 0.45, 0.65 and 0.85, but not for all statistics; see the third reference.

alpha — the value of the desired significance level for the sequential test.

plot — logical indicating whether the monitoring should be plotted.

Details

The testing procedure is described in detail in the third reference. An alternative way of estimating the long-run variance is to use the function lrvar() of the package **sandwich** and to pass it through the argument sigma.

Value

Both functions return lists whose components have explicit names. The function monCpMean() in particular returns a list whose components are

alarm — a logical indicating whether the detector function has exceeded the threshold function.

time.alarm — an integer corresponding to the time at which the detector function has exceeded the threshold function or NA.

times.max — a vector of times at which the successive detectors "r" (if statistic = "r", statistic = "s" or statistic = "t") or "e" (if statistic = "e") have reached their maximum; a vector of NA's if statistic = "cs"; this sequence of times can be used to estimate the time of change from the time of alarm.

time.change — an integer giving the estimated time of change if alarm is TRUE; the latter is simply the value in times.max which corresponds to time.alarm.

statistic — the value of statistic in the call of the function.

`eta`	the value of `eta` in the call of the function.
`gamma`	the value of `gamma` in the call of the function.
`alpha`	the value of `alpha` in the call of the function.
`sigma`	the value of `sigma` in the call of the function.
`detector`	the successive values of the chosen detector.
`threshold`	the value of the constant threshold for the chosen detector.

References

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

J. Gösmann, T. Kley and H. Dette (2020), A new approach for open-end sequential change point monitoring, 49 pages, http://arxiv.org/abs/1906.03225.

M. Holmes and I. Kojadinovic (2020), Open-end nonparametric sequential change-point detection based on the retrospective CUSUM statistic, 41 pages, available on the arXiv.

D.N. Politis and H. White (2004), Automatic block-length selection for the dependent bootstrap, *Econometric Reviews* **23(1)**, pages 53-70.

See Also

see `cpMean()` for the corresponding a posteriori (offline) test.

Examples

```
## Not run:
## Example of open-end montoring
m <- 100 # size of the learning sample

## The learning sample
set.seed(123)
x.learn <- rnorm(m)

## New observations with a change in mean
## to simulate monitoring for the period m+1, ..., n
n <- 5000
k <- 2500 ## the true change-point
x <- c(rnorm(k-m), rnorm(n-k, mean = 0.2))

## Step 1: Compute the detector
det <- detCpMean(x.learn = x.learn, x = x)

## Step 2: Monitoring with the default detector
m1 <- monCpMean(det)
str(m1)

## Monitoring with another detector
m2 <- monCpMean(det, statistic = "s", gamma = 0.85)
str(m2)
```

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

---

stDistAutocop *Combined Test of Stationarity for Univariate Continuous Time Series Sensitive to Changes in the Distribution Function and the Autocopula*

Description

A nonparametric test of stationarity for univariate continuous time series resulting from a combination à la Fisher of the change-point test sensitive to changes in the distribution function implemented in `cpDist()` and the change-point test sensitive to changes in the autcopula implemented in `cpAutocop()`. Approximate p-values are obtained by combining two *multiplier* resampling schemes. Details can be found in the first reference.

Usage

```
stDistAutocop(x, lag = 1, b = NULL, pairwise = FALSE,
              weights = c("parzen", "bartlett"), m = 5, N = 1000)
```

Arguments

`x`	a one-column matrix containing continuous observations.
`lag`	an integer specifying at which lag to consider the autocopula; the autcopula is a (`lag`+1)-dimensional copula.
`b`	strictly positive integer specifying the value of the bandwidth parameter determining the serial dependence when generating dependent multiplier sequences using the 'moving average approach'; see Section 5 of the second reference. If set to `NULL`, `b` will be estimated using the function `bOptEmpProc()`; see the first reference.
`pairwise`	a logical specifying whether the test should focus only on the bivariate margins of the (`lag`+1)-dimensional autocopula.
`weights`	a string specifying the kernel for creating the weights used in the generation of dependent multiplier sequences within the 'moving average approach'; see Section 5 of the second reference.
`m`	a strictly positive integer specifying the number of points of the uniform grid on $(0, 1)$ involved in the estimation of the bandwidth parameter; see Section 5 of the second reference.
`N`	number of multiplier replications.

Details

The testing procedure is described in detail in the second section of the first reference.

Value

An object of class `htest` which is a list, some of the components of which are

`statistic` value of the test statistic.

`p.value` corresponding approximate p-value à Fisher.

`component.p.values`
p-values of the component tests arising in the combination.

`b` the value of parameter `b`.

Note

This is a test for continuous univariate time series.

References

A. Bücher, J.-D. Fermanian and I. Kojadinovic (2019), Combining cumulative sum change-point detection tests for assessing the stationarity of univariate time series, *Journal of Time Series Analysis* **40**, pages 124-150, http://arxiv.org/abs/1709.02673.

A. Bücher and I. Kojadinovic (2016), A dependent multiplier bootstrap for the sequential empirical copula process under strong mixing, *Bernoulli* **22:2**, pages 927-968, http://arxiv.org/abs/1306.3930.

See Also

see `cpDist()` and `cpAutocop()` for the component tests.

Examples

```
## AR1 example
n <- 200
k <- n/2 ## the true change-point
x <- matrix(c(arima.sim(list(ar = -0.1), n = k),
              arima.sim(list(ar = 0.5), n = n - k)))
stDistAutocop(x)

## AR2 example
n <- 200
k <- n/2 ## the true change-point
x <- matrix(c(arima.sim(list(ar = c(0,-0.1)), n = k),
              arima.sim(list(ar = c(0,0.5)), n = n - k)))
## Not run:
stDistAutocop(x)
stDistAutocop(x, lag = 2)
## End(Not run)
stDistAutocop(x, lag = 2, pairwise = TRUE)
```

Index