Package 'HDCD'

June 3, 2024

Type Package

Title High-Dimensional Changepoint Detection

Version 1.1

Date 2024-06-02

Maintainer Per August Jarval Moen <pamoen@math.uio.no>

Description Efficient implementations of the following multiple changepoint detection algorithms: Efficient Sparsity Adaptive Change-point estimator by Moen, Glad and Tveten (2023) <doi:10.48550/arXiv.2306.04702> , Informative Sparse Projection for Estimating Changepoints by Wang and Samworth (2017) <doi:10.1111/rssb.12243>, and the method of Pilliat et al (2023) <doi:10.1214/23-EJS2126>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Imports mclust, Rdpack

RdMacros Rdpack

NeedsCompilation yes

Author Per August Jarval Moen [cre, aut]
(<https://orcid.org/0009-0003-9990-8341>)

Repository CRAN

Date/Publication 2024-06-02 23:20:26 UTC

R topics documented:

ARI *ARI*

Description

Computes the Adjusted Rand Index (ARI) of a vector of estimated change-points.

Usage

```
ARI(etas, eta_hats, n)
```

Arguments

etas	Vector of true change-points
eta_hats	Vector of estimated change-points
n	Sample size

Value

The ARI

Examples

```
library(HDCD)
n = 400
true_changepoints = c(50,100)
est_changepoints = c(51,110)
ARI(true_changepoints, est_changepoints,n)
```

CUSUM *CUSUM transformation of a matrix*

Description

R wrapper for C function computing the CUSUM transformation of a matrix over an interval $(s, e]$. For compatibility with C indexing, the user should subtract 1 from both s and e when supplying the arguments to the function. If start and stop are not supplied, the CUSUM is computed over the full data, so $(s, e] = (0, n]$. In this case, `CUSUM` returns the same result as `cusum.transform` in the package `InspectChangepoint` (Wang and Samworth 2020).

Usage

```
CUSUM(X, start = NULL, stop = NULL)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`start`	Starting point of interval over which the CUSUM should be computed, subtracted by one
`stop`	Ending point of interval over which the CUSUM should be computed, subtracted by one

Value

A matrix of CUSUM values. The (i, j)-th element corresponds to the CUSUM transformation of the i-th row of X, computed over the interval $(\texttt{start} + 1, \texttt{end} + 1]$ and evaluated at position $\texttt{start}+1+j$, i.e. $\sqrt{\frac{e-v}{(e-s)(v-s)}} \sum_{t=s+1}^{v} X_{i,t} - \sqrt{\frac{v-s}{(e-s)(e-v)}} \sum_{t=v+1}^{e} X_{i,t}$, where $s = (\texttt{start}+1)$, $e = (\texttt{stop} + 1)$ and $v = \texttt{start} + 1 + j$.

References

Wang T, Samworth R (2020). *InspectChangepoint: High-Dimensional Changepoint Estimation via Sparse Projection*. R package version 1.1, https://CRAN.R-project.org/package=InspectChangepoint.

Examples

```
n = 10
p = 10
set.seed(101)
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# CUSUM over the full data (s,e] = (0,n]
X_cusum = CUSUM(X)

# CUSUM over (s,e] = (3,9]:
s = 3
e = 9
X_cusum = CUSUM(X, start = s-1, stop = e-1)
```

ESAC *Efficient Sparsity Adaptive Change-point estimator*

Description

R wrapper for C function implementing the full ESAC algorithm (see Moen et al. 2023).

Usage

```
ESAC(
  X,
  threshold_d = 1.5,
  threshold_s = 1,
  alpha = 1.5,
  K = 5,
  debug = FALSE,
  empirical = FALSE,
  tol = 0.001,
  N = 1000,
  thresholds = NULL,
  thresholds_test = NULL,
  threshold_d_test = threshold_d,
  threshold_s_test = threshold_s,
  fast = FALSE,
  rescale_variance = TRUE,
  trim = FALSE,
  NOT = TRUE,
  midpoint = FALSE
)
```

Arguments

X	Matrix of observations, where each row contains a time series
threshold_d	Leading constant for $\lambda(t) \propto r(t)$ for $t = p$. Only relevant when thresholds=NULL
threshold_s	Leading constant for $\lambda(t) \propto r(t)$ for $t \leq \sqrt{p \log n}$. Only relevant when thresholds=NULL
alpha	Parameter for generating seeded intervals
K	Parameter for generating seeded intervals
debug	If TRUE, diagnostic prints are provided during execution
empirical	If TRUE, detection thresholds are based on Monte Carlo simulation using ESAC_calibrate
tol	If empirical=TRUE, tol is the false error probability tolerance
N	If empirical=TRUE, N is the number of Monte Carlo samples used
thresholds	Vector of manually chosen values of $\lambda(t)$ for $t \in \mathcal{T}$, decreasing in t

`thresholds_test`
: Vector of manually chosen values of $\gamma(t)$ for $t \in \mathcal{T}$, decreasing in t

`threshold_d_test`
: Leading constant for $\gamma(t) \propto r(t)$ for $t = p$. Only relevant when `empirical=FALSE` and `thresholds_test=NULL`

`threshold_s_test`
: Leading constant for $\gamma(t) \propto r(t)$ for $t \leq \sqrt{p \log n}$. Only relevant when `empirical=FALSE` and `thresholds_test=NULL`

`fast`
: If `TRUE`, ESAC only tests for a change-point at the midpoint of each seeded interval

`rescale_variance`
: If `TRUE`, each row of the data is re-scaled by a MAD estimate using `rescale_variance`

`trim`
: If `TRUE`, interval trimming is performed

`NOT`
: If `TRUE`, ESAC uses Narrowest-Over-Threshold selection of change-points

`midpoint`
: If `TRUE`, change-point positions are estimated by the mid-point of the seeded interval in which the penalized score is the largest

Value

A list containing

`changepoints`
: vector of estimated change-points

`changepointnumber`
: number of changepoints

`CUSUMval`
: the penalized score at the corresponding change-point in `changepoints`

`coordinates`
: a matrix of zeros and ones indicating which time series are affected by a change in mean, with each row corresponding to the change-point in `changepoints`

`scales`
: vector of estimated noise level for each series

`startpoints`
: start point of the seeded interval detecting the corresponding change-point in `changepoints`

`endpoints`
: end point of the seeded interval detecting the corresponding change-point in `changepoints`

`thresholds`
: vector of values of $\lambda(t)$ for $t \in \mathcal{T}$ in decreasing order

`thresholds_test`
: vector of values of $\gamma(t)$ for $t \in \mathcal{T}$ in decreasing order

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 50
p = 50
set.seed(100)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +1

# Vanilla ESAC:
res = ESAC(X)
res$changepoints

# Manually setting leading constants for \lambda(t) and \gamma(t)
res = ESAC(X,
           threshold_d = 2, threshold_s = 2, #leading constants for \lambda(t)
           threshold_d_test = 2, threshold_s_test = 2 #leading constants for \gamma(t)
)
res$changepoints #estimated change-point locations

# Empirical choice of thresholds:
res = ESAC(X, empirical = TRUE, N = 100, tol = 1/100)
res$changepoints

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_emp = ESAC_calibrate(n,p, N=100, tol=1/100)
res = ESAC(X, thresholds_test = thresholds_emp[[1]])
res$changepoints
```

`ESAC_calibrate` *Generates empirical penalty function $\gamma(t)$ for the ESAC algorithm using Monte Carlo simulation*

Description

R wrapper for C function choosing the penalty function $\gamma(t)$ by Monte Carlo simulation, as described in Appendix B in Moen et al. (2023).

Usage

```
ESAC_calibrate(
  n,
  p,
  alpha = 1.5,
  K = 5,
  N = 1000,
  tol = 0.001,
  bonferroni = TRUE,
```

```
  fast = FALSE,
  rescale_variance = TRUE,
  tdf = NULL,
  debug = FALSE
)
```

Arguments

n	Number of observations
p	Number time series
alpha	Parameter for generating seeded intervals
K	Parameter for generating seeded intervals
N	Number of Monte Carlo samples used
tol	False error probability tolerance
bonferroni	If TRUE, a Bonferroni correction applied and the empirical penalty function $\gamma(t)$ is chosen by simulating leading constants of $r(t)$ through Monte Carlo simulation.
fast	If TRUE, ESAC only tests for a change-point at the midpoint of each seeded interval
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
tdf	If NULL, samples are drawn from a Gaussian distribution. Otherwise, they are drawn from a t distribution with tdf degrees of freedom.
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

without_partial	a vector of values of $\gamma(t)$ for $t \in \mathcal{T}$ decreasing in t
with_partial	same as without_partial
as	vector of threshold values $a(t)$ for $t \in \mathcal{T}$ decreasing in t
nu_as	vector of conditional expectations $\nu_{a(t)}$ of a thresholded Gaussian, for $t \in \mathcal{T}$ decreasing in t

#'

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 50
p = 50

set.seed(100)
thresholds_emp = ESAC_calibrate(n,p, N=100, tol=1/100)
set.seed(100)
thresholds_emp_without_bonferroni = ESAC_calibrate(n,p, N=100, tol=1/100,bonferroni=FALSE)
thresholds_emp[[1]] # vector of \gamma(t) for t = p,...,1
thresholds_emp_without_bonferroni[[1]] # vector of \gamma(t) for t = p,...,1

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +2

res = ESAC(X, thresholds_test = thresholds_emp[[1]])
res$changepoints
```

ESAC_test *ESAC single change-point test*

Description

R wrapper for C function testing for a single change-point using ESAC (see Moen et al. 2023).

Usage

```
ESAC_test(
  X,
  threshold_d = 1.5,
  threshold_s = 1,
  debug = FALSE,
  empirical = FALSE,
  thresholds = NULL,
  fast = FALSE,
  tol = 0.001,
  N = 1000,
  rescale_variance = TRUE
)
```

Arguments

X	Matrix of observations, where each row contains a time series
threshold_d	Leading constant for $\gamma(t) \propto r(t)$ for $t = p$. Only relevant when empirical=FALSE and thresholds=NULL

threshold_s	Leading constant for $\gamma(t) \propto r(t)$ for $t \leq \sqrt{p \log n}$. Only relevant when empirical=FALSE and thresholds=NULL
debug	If TRUE, diagnostic prints are provided during execution
empirical	If TRUE, detection thresholds are based on Monte Carlo simulation using ESAC_test_calibrate
thresholds	Vector of manually chosen values of $\gamma(t)$ for $t \in \mathcal{T}$, decreasing in t
fast	If TRUE, ESAC only tests for a change-point at the midpoint of each seeded interval
tol	If empirical=TRUE, tol is the false error probability tolerance
N	If empirical=TRUE, N is the number of Monte Carlo samples used
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance

Value

1 if a change-point is detected, 0 otherwise

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 50
p = 50

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 26:n] = X[1:5, 26:n] +1

# Vanilla ESAC:
resX = ESAC_test(X)
resX
resY = ESAC_test(Y)
resY

# Manually setting leading constants for \lambda(t) and \gamma(t)
resX = ESAC_test(X,
                 threshold_d = 2, threshold_s = 2, #leading constants for \gamma(t)
)
resX
resY = ESAC_test(Y,
                 threshold_d = 2, threshold_s = 2, #leading constants for \gamma(t)
)
resY
```

```
# Empirical choice of thresholds:
resX = ESAC_test(X, empirical = TRUE, N = 100, tol = 1/100)
resX
resY = ESAC_test(Y, empirical = TRUE, N = 100, tol = 1/100)
resY

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_test_emp = ESAC_test_calibrate(n,p, N=100, tol=1/100,bonferroni=TRUE)
resX = ESAC_test(X, thresholds = thresholds_test_emp[[1]])
resX
resY = ESAC_test(Y, thresholds = thresholds_test_emp[[1]])
resY
```

`ESAC_test_calibrate` *Generates empirical penalty function $\gamma(t)$ for single change-point testing using Monte Carlo simulation*

Description

R wrapper for C function choosing the penalty function $\gamma(t)$ by Monte Carlo simulation, as described in Appendix B in Moen et al. (2023), for testing for a single change-point.

Usage

```
ESAC_test_calibrate(
  n,
  p,
  bonferroni = TRUE,
  N = 1000,
  tol = 1/1000,
  fast = FALSE,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

`n`	Number of observations
`p`	Number time series
`bonferroni`	If `TRUE`, a Bonferroni correction applied and the empirical penalty function $\gamma(t)$ is chosen by simulating leading constants of $r(t)$ through Monte Carlo simulation.
`N`	Number of Monte Carlo samples used
`tol`	False positive probability tolerance
`fast`	If `TRUE`, ESAC only tests for a change-point at the midpoint of the interval $(0, \ldots, n]$

`rescale_variance`
: If `TRUE`, each row of the data is re-scaled by a MAD estimate using `rescale_variance`

`debug`
: If `TRUE`, diagnostic prints are provided during execution

Value

A list containing a vector of values of $\gamma(t)$ for $t \in \mathcal{T}$ decreasing (element #1), a vector of corresponding values of the threshold $a(t)$ (element # 3), a vector of corresponding values of $\nu_{a(t)}$

A list containing

`without_partial`
: a vector of values of $\gamma(t)$ for $t \in \mathcal{T}$ decreasing in t

`with_partial`
: same as `without_partial`

`as`
: vector of threshold values $a(t)$ for $t \in \mathcal{T}$ decreasing in t

`nu_as`
: vector of conditional expectations $\nu_{a(t)}$ of a thresholded Gaussian, for $t \in \mathcal{T}$ decreasing in t

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 50
p = 50

set.seed(100)
thresholds_emp = ESAC_test_calibrate(n,p, bonferroni=TRUE,N=100, tol=1/100)
set.seed(100)
thresholds_emp_without_bonferroni = ESAC_test_calibrate(n,p, bonferroni=FALSE,N=100, tol=1/100)
thresholds_emp[[1]] # vector of \gamma(t) for t = p,...,1
thresholds_emp_without_bonferroni[[1]] # vector of \gamma(t) for t = p,...,1

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 26:n] = X[1:5, 26:n] +2
resX = ESAC_test(X, thresholds = thresholds_emp[[1]])
resX
resY = ESAC_test(Y,  thresholds = thresholds_emp[[1]])
resY
```

hausdorff *Hausdorff distance between two sets*

Description

Computes the Hausdorff distance between two sets represented as vectors v1 and v2. If v1 == NULL and v2 != NULL, then the largest distance between an element of v1 and the set $\{1, n\}$ is returned, and vice versa. If both vectors are NULL, 0 is returned.

Usage

```
hausdorff(v1, v2, n)
```

Arguments

v1	Vector representing set 1
v2	Vector representing set 2
n	Sample size (only relevant when either v1 or v2 is NULL)

Value

The Hausdorff distance between v1 and v2

Examples

```
library(HDCD)
n = 400
true_changepoints = c(50,100)
est_changepoints = c(51,110)
hausdorff(true_changepoints, est_changepoints,n)
hausdorff(true_changepoints, NULL,n)
hausdorff(NULL, est_changepoints,n)
hausdorff(NULL,NULL)
```

Inspect *Informative sparse projection for estimating change-points (Inspect)*

Description

R wrapper for C function implementing a Narrowest-Over-Threshold variant of Inspect Wang and Samworth (2018) as specified in Appendix C in Moen et al. (2023). Note that the algorithm is only implemented for $\mathcal{S} = \mathcal{S}_2$, in the notation of Moen et al. (2023).

Usage

```
Inspect(
  X,
  lambda = NULL,
  xi = NULL,
  alpha = 1.5,
  K = 5,
  eps = 1e-10,
  empirical = FALSE,
  maxiter = 10000,
  N = 100,
  tol = 1/100,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

X	Matrix of observations, where each row contains a time series
lambda	Manually specified value of λ (can be NULL, in which case $\lambda \leftarrow \sqrt{\log(p \log n)/2)}$
xi	Manually specified value of ξ (can be NULL, in which case $\xi \leftarrow 4\sqrt{\log(np)})$
alpha	Parameter for generating seeded intervals
K	Parameter for generating seeded intervals
eps	Threshold for declaring numerical convergence of the power method
empirical	If TRUE, the detection threshold xi is based on Monte Carlo simulation using Inspect_calibrate
maxiter	Maximum number of iterations for the power method
N	If empirical=TRUE, N is the number of Monte Carlo samples used
tol	If empirical=TRUE, tol is the false error probability tolerance
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

changepoints	vector of estimated change-points
changepointnumber	number of changepoints
CUSUMval	vector with the sparse projected CUSUMs corresponding to changepoints
coordinates	a matrix of zeros and ones indicating which time series are affected by a change in mean, with each row corresponding to the change-point in changepoints
scales	vector of estimated noise level for each series

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Wang T, Samworth RJ (2018). "High dimensional change point estimation via sparse projection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **80**(1), 57–83. ISSN 1467-9868, doi:10.1111/rssb.12243, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12243.

Examples

```
library(HDCD)
n = 50
p = 50
set.seed(100)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +1

# Vanilla Inspect:
res = Inspect(X)
res$changepoints

# Manually setting leading constants for \lambda(t) and \gamma(t)
res = Inspect(X,
              lambda = sqrt(log(p*log(n))/2),
              xi = 4*sqrt(log(n*p))
)
res$changepoints #estimated change-point locations

# Empirical choice of thresholds:
res = Inspect(X, empirical=TRUE, N = 100, tol = 1/100)
res$changepoints

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_emp = Inspect_calibrate(n,p, N=100, tol=1/100)
res = Inspect(X, xi = thresholds_emp$max_value)
res$changepoints
```

`Inspect_calibrate` *Generates empirical detection threshold ξ using Monte Carlo simulation*

Description

R wrapper for C function choosing empirical detection threshold ξ for the Narrowest-Over-Threshold variant of Inspect (as specified in section 4.2 in Moen et al. 2023) using Monte Carlo simulation.

Usage

```
Inspect_calibrate(
  n,
  p,
  N = 100,
  tol = 1/100,
  lambda = NULL,
  alpha = 1.5,
  K = 5,
  eps = 1e-10,
  maxiter = 10000,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

n	Number of observations
p	Number time series
N	Number of Monte Carlo samples used
tol	False positive probability tolerance
lambda	Manually specified value of λ (can be NULL, in which case $\lambda \leftarrow \sqrt{\log(p \log n)/2)}$
alpha	Parameter for generating seeded intervals
K	Parameter for generating seeded intervals
eps	Threshold for declaring numerical convergence of the power method
maxiter	Maximum number of iterations for the power method
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

max_value	the empirical threshold

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 50
p = 50
```

```
set.seed(100)
thresholds_emp = Inspect_calibrate(n,p, N=100, tol=1/100)
thresholds_emp$max_value # xi

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +2

res = Inspect(X, xi = thresholds_emp$max_value)
res$changepoints
```

`Inspect_test` *Inspect single change-point test*

Description

R wrapper for C function testing for a single change-point using Inspect Wang and Samworth (2018).

Usage

```
Inspect_test(
  X,
  lambda = NULL,
  xi = NULL,
  eps = 1e-10,
  empirical = FALSE,
  N = 100,
  tol = 1/100,
  maxiter = 10000,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`lambda`	Manually specified value of λ (can be `NULL`, in which case $\lambda \leftarrow \sqrt{\log(p \log n)/2)}$
`xi`	Manually specified value of ξ (can be NULL, in which case $\xi \leftarrow 4\sqrt{\log(np)})$
`eps`	Threshold for declaring numerical convergence of the power method
`empirical`	If `TRUE`, the detection threshold xi is based on Monte Carlo simulation using `Inspect_test_calibrate`
`N`	If `empirical=TRUE`, `N` is the number of Monte Carlo samples used
`tol`	If `empirical=TRUE`, `tol` is the false error probability tolerance

maxiter	Maximum number of iterations for the power method
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
debug	If TRUE, diagnostic prints are provided during execution

Value

1 if a change-point is detected, 0 otherwise

References

Wang T, Samworth RJ (2018). "High dimensional change point estimation via sparse projection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **80**(1), 57–83. ISSN 1467-9868, doi:10.1111/rssb.12243, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12243.

Examples

```
library(HDCD)
n = 50
p = 50

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 26:n] = X[1:5, 26:n] +1

# Vanilla Inspect:
resX = Inspect_test(X)
resX
resY = Inspect_test(Y)
resY

# Manually setting \lambda and \xi:
resX = Inspect_test(X,
                    lambda = sqrt(log(p*log(n))/2),
                    xi = 4*sqrt(log(n*p))
)
resX
resY = Inspect_test(Y,
                    lambda = sqrt(log(p*log(n))/2),
                    xi = 4*sqrt(log(n*p))
)
resY

# Empirical choice of thresholds:
resX = Inspect_test(X, empirical = TRUE, N = 100, tol = 1/100)
resX
resY = Inspect_test(Y, empirical = TRUE, N = 100, tol = 1/100)
```

```
resY

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_test_emp = Inspect_test_calibrate(n,p, N=100, tol=1/100)
resX = Inspect_test(X, xi = thresholds_test_emp$max_value)
resX
resY = Inspect_test(Y, xi = thresholds_test_emp$max_value)
resY
```

`Inspect_test_calibrate` *Generates empirical detection threshold ξ for single change-point testing using Monte Carlo simulation*

Description

R wrapper for C function choosing the empirical detection threshold ξ for Inspect Wang and Samworth (2018) for single change-point testing using Monte Carlo simulation.

Usage

```
Inspect_test_calibrate(
  n,
  p,
  N = 100,
  tol = 1/100,
  lambda = NULL,
  eps = 1e-10,
  maxiter = 10000,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

`n`	Number of observations
`p`	Number time series
`N`	Number of Monte Carlo samples used
`tol`	False positive probability tolerance
`lambda`	Manually specified value of λ (can be `NULL`, in which case $\lambda \leftarrow \sqrt{\log(p \log n)/2)}$
`eps`	Threshold for declaring numerical convergence of the power method
`maxiter`	Maximum number of iterations for the power method
`rescale_variance`	If `TRUE`, each row of the data is re-scaled by a MAD estimate using `rescale_variance`
`debug`	If `TRUE`, diagnostic prints are provided during execution

Value

A list containing

`max_value` the empirical threshold

References

Wang T, Samworth RJ (2018). "High dimensional change point estimation via sparse projection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **80**(1), 57–83. ISSN 1467-9868, doi:10.1111/rssb.12243, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12243.

Examples

```
library(HDCD)
n = 50
p = 50

set.seed(100)
thresholds_emp = Inspect_test_calibrate(n,p,N=100, tol=1/100)
thresholds_emp


# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 26:n] = X[1:5, 26:n] +2
resX = Inspect_test(X, xi = thresholds_emp$max_value)
resX
resY = Inspect_test(Y,  xi = thresholds_emp$max_value)
resY
```

`Pilliat` *Pilliat multiple change-point detection algorithm*

Description

R wrapper function for C implementation of the multiple change-point detection algorithm by Pilliat et al. (2023), using seeded intervals generated by Algorithm 4 in Moen et al. (2023). For the sake of simplicity, detection thresholds are chosen independently of the width of the interval in which a change-point is tested for (so $r = 1$ is set for all intervals).

Usage

```
Pilliat(
  X,
  threshold_d_const = 4,
  threshold_bj_const = 6,
  threshold_partial_const = 4,
  K = 2,
  alpha = 1.5,
  empirical = FALSE,
  threshold_dense = NULL,
  thresholds_partial = NULL,
  thresholds_bj = NULL,
  N = 100,
  tol = 0.01,
  rescale_variance = TRUE,
  test_all = FALSE,
  debug = FALSE
)
```

Arguments

X	Matrix of observations, where each row contains a time series
threshold_d_const	Leading constant for the analytical detection threshold for the dense statistic
threshold_bj_const	Leading constant for p_0 when computing the detection threshold for the Berk-Jones statistic
threshold_partial_const	Leading constant for the analytical detection threshold for the partial sum statistic
K	Parameter for generating seeded intervals
alpha	Parameter for generating seeded intervals
empirical	If TRUE, detection thresholds are based on Monte Carlo simulation using Pilliat_calibrate
threshold_dense	Manually specified value of detection threshold for the dense statistic
thresholds_partial	Vector of manually specified detection thresholds for the partial sum statistic, for sparsities/partial sums $t = 1, 2, 4, \ldots, 2^{\lfloor \log_2(p) \rfloor}$
thresholds_bj	Vector of manually specified detection thresholds for the Berk-Jones statistic, order corresponding to $x = 1, 2, \ldots, x_0$
N	If empirical=TRUE, N is the number of Monte Carlo samples used
tol	If empirical=TRUE, tol is the false error probability tolerance
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate (see rescale_variance)
test_all	If TRUE, the algorithm tests for a change-point in all candidate positions of each considered interval
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

`changepoints`	vector of estimated change-points
`number_of_changepoints`	number of changepoints
`scales`	vector of estimated noise level for each series
`startpoints`	start point of the seeded interval detecting the corresponding change-point in `changepoints`
`endpoints`	end point of the seeded interval detecting the corresponding change-point in `changepoints`

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Pilliat E, Carpentier A, Verzelen N (2023). "Optimal multiple change-point detection for high-dimensional data." *Electronic Journal of Statistics*, **17**(1), 1240 – 1315.

Examples

```
library(HDCD)
n = 50
p = 50
set.seed(100)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +2

# Vanilla Pilliat:
res = Pilliat(X)
res$changepoints

# Manually setting leading constants for detection thresholds
res = Pilliat(X, threshold_d_const = 4, threshold_bj_const = 6, threshold_partial_const=4)
res$changepoints #estimated change-point locations

# Empirical choice of thresholds:
res = Pilliat(X, empirical = TRUE, N = 100, tol = 1/100)
res$changepoints

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_emp = Pilliat_calibrate(n,p, N=100, tol=1/100)
thresholds_emp$thresholds_partial # thresholds for partial sum statistic
thresholds_emp$thresholds_bj # thresholds for Berk-Jones statistic
thresholds_emp$threshold_dense # thresholds for Berk-Jones statistic
res = Pilliat(X, threshold_dense =thresholds_emp$threshold_dense,
              thresholds_bj = thresholds_emp$thresholds_bj,
```

```
              thresholds_partial =thresholds_emp$thresholds_partial )
res$changepoints
```

`Pilliat_calibrate` *Generates detection thresholds for the Pilliat algorithm using Monte Carlo simulation*

Description

R wrapper for function choosing detection thresholds for the Dense, Partial sum and Berk-Jones statistics in the multiple change-point detection algorithm of Pilliat et al. (2023) using Monte Carlo simulation. When `Bonferroni==TRUE`, the detection thresholds are chosen by simulating the leading constant in the theoretical detection thresholds given in Pilliat et al. (2023), similarly as described in Appendix B in Moen et al. (2023) for ESAC. When `Bonferroni==TRUE`, the thresholds for the Berk-Jones statistic are theoretical and not chosen by Monte Carlo simulation.

Usage

```
Pilliat_calibrate(
  n,
  p,
  N = 100,
  tol = 0.01,
  bonferroni = TRUE,
  threshold_bj_const = 6,
  K = 2,
  alpha = 1.5,
  rescale_variance = TRUE,
  test_all = FALSE,
  debug = FALSE
)
```

Arguments

`n`	Number of observations
`p`	Number time series
`N`	Number of Monte Carlo samples used
`tol`	False error probability tolerance
`bonferroni`	If `TRUE`, a Bonferroni correction applied and the detection thresholds for each statistic is chosen by simulating the leading constant in the theoretical detection thresholds
`threshold_bj_const`	Leading constant for p_0 for the Berk-Jones statistic
`K`	Parameter for generating seeded intervals
`alpha`	Parameter for generating seeded intervals

`rescale_variance`
: If `TRUE`, each row of the data is re-scaled by a MAD estimate (see `rescale_variance`)

`test_all`
: If `TRUE`, a change-point test is applied to each candidate change-point position in each interval. If `FALSE`, only the mid-point of each interval is considered

`debug`
: If `TRUE`, diagnostic prints are provided during execution

Value

A list containing

`thresholds_partial`
: vector of thresholds for the Partial Sum statistic (respectively for $t = 1, 2, 4, \ldots, 2^{\lfloor \log_2(p) \rfloor}$ number of terms in the partial sum)

`threshold_dense`
: threshold for the dense statistic

`thresholds_bj`
: vector of thresholds for the Berk-Jones static (respectively for $x = 1, 2, \ldots, x_0$)

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Pilliat E, Carpentier A, Verzelen N (2023). "Optimal multiple change-point detection for high-dimensional data." *Electronic Journal of Statistics*, **17**(1), 1240 – 1315.

Examples

```
library(HDCD)
n = 50
p = 50

set.seed(100)
thresholds_emp = Pilliat_calibrate(n,p, N=100, tol=1/100)
thresholds_emp$thresholds_partial # thresholds for partial sum statistic
thresholds_emp$thresholds_bj # thresholds for Berk-Jones statistic
thresholds_emp$threshold_dense # thresholds for Berk-Jones statistic
set.seed(100)
thresholds_emp_without_bonferroni = Pilliat_calibrate(n,p, N=100, tol=1/100,bonferroni = FALSE)
thresholds_emp_without_bonferroni$thresholds_partial # thresholds for partial sum statistic
thresholds_emp_without_bonferroni$thresholds_bj # thresholds for Berk-Jones statistic
thresholds_emp_without_bonferroni$threshold_dense # thresholds for Berk-Jones statistic

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +2

res = Pilliat(X, threshold_dense =thresholds_emp$threshold_dense,
              thresholds_bj = thresholds_emp$thresholds_bj,
              thresholds_partial =thresholds_emp$thresholds_partial )
res$changepoints
```

`Pilliat_test` *Pilliat single change-point test*

Description

R wrapper function testing for a single change-point using the three test statistics in the multiple change point detection algorithm of Pilliat et al. (2023). See also Appendix E in Moen et al. (2023).

Usage

```
Pilliat_test(
  X,
  empirical = FALSE,
  N = 100,
  tol = 0.05,
  thresholds_partial = NULL,
  threshold_dense = NULL,
  thresholds_bj = NULL,
  threshold_d_const = 4,
  threshold_bj_const = 6,
  threshold_partial_const = 4,
  rescale_variance = TRUE,
  fast = FALSE,
  debug = FALSE
)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`empirical`	If `TRUE`, detection thresholds are based on Monte Carlo simulation
`N`	If `empirical=TRUE`, `N` is the number of Monte Carlo samples used
`tol`	If `empirical=TRUE`, `tol` is the false error probability tolerance
`thresholds_partial`	Vector of manually specified detection thresholds for the partial sum statistic, for sparsities/partial sums $t = 1, 2, 4, \ldots, 2^{\lfloor \log_2(p) \rfloor}$
`threshold_dense`	Manually specified value of detection threshold for the dense statistic
`thresholds_bj`	Vector of manually specified detection thresholds for the Berk-Jones statistic, order corresponding to $x = 1, 2, \ldots, x_0$
`threshold_d_const`	Leading constant for the analytical detection threshold for the dense statistic
`threshold_bj_const`	Leading constant for p_0 when computing the detection threshold for the Berk-Jones statistic

`threshold_partial_const`
: Leading constant for the analytical detection threshold for the partial sum statistic

`rescale_variance`
: If `TRUE`, each row of the data is re-scaled by a MAD estimate (see `rescale_variance`)

`fast`
: If `TRUE`, only the mid-point of $(0, \ldots, n]$ is tested for a change-point. Otherwise a test is performed at each candidate change-point poisition

`debug`
: If `TRUE`, diagnostic prints are provided during execution

Value

1 if a change-point is detected, 0 otherwise

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Pilliat E, Carpentier A, Verzelen N (2023). "Optimal multiple change-point detection for high-dimensional data." *Electronic Journal of Statistics*, **17**(1), 1240 – 1315.

Examples

```
library(HDCD)
n = 200
p = 200

# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 100:200] = X[1:5, 100:200] +1

# Vanilla Pilliat test:
resX = Pilliat_test(X)
resX
resY = Pilliat_test(Y)
resY

# Manually setting leading constants for the theoretical thresholds for the three
# test statistics used
resX = Pilliat_test(X,
                    threshold_d_const=4,
                    threshold_bj_const=6,
                    threshold_partial_const=4
)
resX
resY = Pilliat_test(Y,
                    threshold_d_const=4,
                    threshold_bj_const=6,
```

```
                    threshold_partial_const=4
)
resY

# Empirical choice of thresholds:
resX = Pilliat_test(X, empirical = TRUE, N = 100, tol = 1/100)
resX
resY = Pilliat_test(Y, empirical = TRUE, N = 100, tol = 1/100)
resY

# Manual empirical choice of thresholds (equivalent to the above)
thresholds_test_emp = Pilliat_test_calibrate(n,p, N=100, tol=1/100,bonferroni=TRUE)
resX = Pilliat_test(X,
                    threshold_dense=thresholds_test_emp$threshold_dense,
                    thresholds_bj = thresholds_test_emp$thresholds_bj,
                    thresholds_partial = thresholds_test_emp$thresholds_partial
)
resX
resY = Pilliat_test(Y,
                    threshold_dense=thresholds_test_emp$threshold_dense,
                    thresholds_bj = thresholds_test_emp$thresholds_bj,
                    thresholds_partial = thresholds_test_emp$thresholds_partial
)
resY
```

`Pilliat_test_calibrate`

Generates detection thresholds for the Pilliat algorithm for testing for a single change-point using Monte Carlo simulation

---

Description

R wrapper for function choosing detection thresholds for the Dense, Partial sum and Berk-Jones statistics in the multiple change-point detection algorithm of Pilliat et al. (2023) for single change-point testing using Monte Carlo simulation. When `Bonferroni==TRUE`, the detection thresholds are chosen by simulating the leading constant in the theoretical detection thresholds given in Pilliat et al. (2023), similarly as described in Appendix B in Moen et al. (2023) for ESAC. When `Bonferroni==TRUE`, the thresholds for the Berk-Jones statistic are theoretical and not chosen by Monte Carlo simulation.

Usage

```
Pilliat_test_calibrate(
  n,
  p,
  N = 100,
  tol = 1/100,
  threshold_bj_const = 6,
  bonferroni = TRUE,
```

```
  rescale_variance = TRUE,
  fast = FALSE,
  debug = FALSE
)
```

Arguments

n	Number of observations
p	Number time series
N	Number of Monte Carlo samples used
tol	False error probability tolerance
threshold_bj_const	Leading constant for p_0 for the Berk-Jones statistic
bonferroni	If TRUE, a Bonferroni correction applied and the detection thresholds for each statistic is chosen by simulating the leading constant in the theoretical detection thresholds
rescale_variance	If TRUE, each row of the data is rescaled by a MAD estimate
fast	If FALSE, a change-point test is applied to each candidate change-point position in each interval. If FALSE, only the mid-point of each interval is considered
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

thresholds_partial	vector of thresholds for the Partial Sum statistic (respectively for $t = 1, 2, 4, \ldots, 2^{\lfloor \log_2(p) \rfloor}$ number of terms in the partial sum)
threshold_dense	threshold for the dense statistic
thresholds_bj	vector of thresholds for the Berk-Jones static (respectively for $x = 1, 2, \ldots, x_0$)

Examples

```
library(HDCD)
n = 50
p = 50

set.seed(100)
thresholds_test_emp = Pilliat_test_calibrate(n,p, bonferroni=TRUE,N=100, tol=1/100)
set.seed(100)
thresholds_test_emp_without_bonferroni = Pilliat_test_calibrate(n,p,
                                         bonferroni=FALSE,N=100, tol=1/100)
thresholds_test_emp # thresholds with bonferroni correction
thresholds_test_emp_without_bonferroni # thresholds without bonferroni correction

# Generating data
```

```
X = matrix(rnorm(n*p), ncol = n, nrow=p)
Y = matrix(rnorm(n*p), ncol = n, nrow=p)

# Adding a single sparse change-point to X (and not Y):
X[1:5, 25:50] = X[1:5, 25:50] +2
resX = Pilliat_test(X,
                    threshold_dense=thresholds_test_emp$threshold_dense,
                    thresholds_bj = thresholds_test_emp$thresholds_bj,
                    thresholds_partial = thresholds_test_emp$thresholds_partial
)
resX
resY = Pilliat_test(Y,
                    threshold_dense=thresholds_test_emp$threshold_dense,
                    thresholds_bj = thresholds_test_emp$thresholds_bj,
                    thresholds_partial = thresholds_test_emp$thresholds_partial
)
resY
```

`rescale_variance` *Re-scales each row of matrix by its MAD estimate*

Description

R wrapper for C function computing the (rescaled) median absolute difference in differences for each row of the input matrix. The rescaling factor is set to 1.05 (corresponding to the Normal distribution). Each row of the input matrix then re-scaled by the corresponding noise estimate.

Usage

```
rescale_variance(X, debug = FALSE)
```

Arguments

`X`	A $p \times n$ matrix
`debug`	If `TRUE`, diagnostic prints are provided during execution

Value

A list containing

`X`	the input matrix, variance re-scaled and flattened
`scales`	vector of MAD estimates of the noise level of each row of the input matrix

Examples

```
library(HDCD)
n = 200
p = 500
set.seed(101)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)

ret = rescale_variance(X)
ret$X #rescaled matrix
ret$scales #estimated noise level for each time series (each row)

# Note that the rescaled matrix is in (column wise) vector form. To transform it back to a matrix,
# do the following:
rescaled_X = matrix(ret$X, nrow = p, ncol=n)
```

`single_CUSUM` *CUSUM transformation of matrix at a specific position*

Description

R wrapper for C function computing the CUSUM transformation of matrix over an interval $(s, e]$ evaluated at a specific position. For compatibility with C indexing, the user should subtract 1 from s, e and v when supplying the arguments to the function. If start and stop are not supplied, the CUSUM is computed over the full data, so $(s, e] = (0, n]$.

Usage

```
single_CUSUM(X, start = NULL, stop = NULL, pos)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`start`	Starting point of interval over which the CUSUM should be computed, subtracted by one
`stop`	Ending point of interval over which the CUSUM should be computed, subtracted by one
`pos`	Position at which the CUSUM should be evaluated, subtracted by one

Value

A vector of CUSUM values, each corresponding to a row of the input matrix. The i-th element corresponds to the CUSUM transformation of the i-th row of X, computed over the interval $(\texttt{start} + 1, \texttt{end}+1]$ and evaluated at position `pos`, i.e. $\sqrt{\frac{e-v}{(e-s)(v-s)}}\sum_{t=s+1}^{v} X_{i,t} - \sqrt{\frac{v-s}{(e-s)(e-v)}}\sum_{t=v+1}^{e} X_{i,t}$, where $s = (\texttt{start} + 1)$, $e = (\texttt{stop} + 1)$ and $v = \texttt{pos} + 1$.

Examples

```
n = 10
p = 10
set.seed(101)
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# CUSUM over the full data (s,e] = (0,n] evaluated at position v=4
position = 4
X_cusum_single = single_CUSUM(X,pos = position-1)
X_cusum_single

# verifying that this corresponds to the 4-th row of output of CUSUM():
X_cusum = CUSUM(X)
X_cusum[,4]
```

`single_ESAC` *Efficient Sparsity Adaptive Change-point estimator for a single change-point*

Description

R wrapper for C function implementing ESAC for single change-point estimation, as described in section 3.1 in Moen et al. (2023)

Usage

```
single_ESAC(
  X,
  threshold_d = 1.5,
  threshold_s = 1,
  rescale_variance = FALSE,
  debug = FALSE
)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`threshold_d`	Leading constant for $\lambda(t) \propto r(t)$ for $t = p$
`threshold_s`	Leading constant for $\lambda(t) \propto r(t)$ for $t \leq \sqrt{p \log n}$.
`rescale_variance`	If `TRUE`, each row of the data is re-scaled by a MAD estimate using `rescale_variance`
`debug`	If `TRUE`, diagnostic prints are provided during execution

Value

A list containing

`pos`	estimated change-point location
`s`	the value of $t \in \mathcal{T}$ at which the sparsity specific score is maximized

References

Moen PAJ, Glad IK, Tveten M (2023). "Efficient sparsity adaptive changepoint estimation." Arxiv preprint, 2306.04702, https://doi.org/10.48550/arXiv.2306.04702.

Examples

```
library(HDCD)
n = 500
p = 500
set.seed(101)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 201:500] = X[1:5, 201:500] +1

res = single_ESAC(X,rescale_variance=TRUE)
res$pos

# Manually setting the leading constants for \lambda(t):
# here \lambda(t) = 2 (\sqrt{p \log(n^4)}  + \log (n^4)) for t=p
# and             = 2 (t \log (ep\log n^4 / t^2) + \log(n^4))
res = single_ESAC(X, threshold_d = 2, threshold_s = 2)
res$pos
```

`single_Inspect` *Inspect for single change-point estimation*

Description

R wrapper for C function for single change-point estimation using Inspect (Wang and Samworth 2018). Note that the algorithm is only implemented for $\mathcal{S} = \mathcal{S}_2$, in the notation of Wang and Samworth (2018).

Usage

```
single_Inspect(
  X,
  lambda = sqrt(log(p * log(n))/2),
  eps = 1e-10,
  rescale_variance = FALSE,
  maxiter = 10000,
  debug = FALSE
)
```

Arguments

`X`	Matrix of observations, where each row contains a time series
`lambda`	Manually specified value of λ (can be `NULL`, in which case $\lambda \leftarrow \sqrt{\log(p \log n)/2})$

eps	Threshold for declaring numerical convergence of the power method
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
maxiter	Maximum number of iterations for the power method
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

pos	estimated change-point location
CUSUMval	projected CUSUM value at the estimated change-point position

References

Wang T, Samworth RJ (2018). "High dimensional change point estimation via sparse projection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **80**(1), 57–83. ISSN 1467-9868, doi:10.1111/rssb.12243, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12243.

Examples

```
library(HDCD)
n = 500
p = 500
set.seed(101)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 201:500] = X[1:5, 201:500] +1

res = single_Inspect(X,rescale_variance=TRUE)
res$pos

# Manually setting the value of \lambda:
res = single_Inspect(X, lambda = 2*sqrt(log(p*log(n))/2))
res$pos
```

---

single_SBS *Sparsified Binary Segmentation for single change-point estimation*

Description

R wrapper for C function for single change-point estimation using Sparsified Binary Segmentation Cho and Fryzlewicz (2015).

Usage

```
single_SBS(
  X,
  threshold = NULL,
  rescale_variance = TRUE,
  empirical = FALSE,
  N = 100,
  tol = 1/100,
  debug = FALSE
)
```

Arguments

X	Matrix of observations, where each row contains a time series
threshold	Manually specified value of the threshold π_T
rescale_variance	If TRUE, each row of the data is re-scaled by a MAD estimate using rescale_variance
empirical	If TRUE, the threshold is based on Monte Carlo simulation
N	If empirical=TRUE, N is the number of Monte Carlo samples used
tol	If empirical=TRUE, tol is the false error probability tolerance
debug	If TRUE, diagnostic prints are provided during execution

Value

A list containing

pos	estimated change-point location
maxval	maximum thresholded and aggregated CUSUM at the estimated change-point position

References

Cho H, Fryzlewicz P (2015). "Multiple-change-point detection for high dimensional time series via sparsified binary segmentation." *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, **77**(2), 475–507. ISSN 1369-7412, Publisher: [Royal Statistical Society, Wiley], https://www.jstor.org/stable/24774746.

Examples

```
# Single SBS
library(HDCD)
n = 50
p = 50
set.seed(101)
# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +1
```

```
res = single_SBS(X,threshold=7,rescale_variance=TRUE)
res$pos

# Choose threhsold by Monte Carlo:
res = single_SBS(X,empirical=TRUE,rescale_variance=TRUE)
res$pos
```

`single_SBS_calibrate` *Generates threshold π_T for Sparsified Binary Segmentation for single change-point detection*

Description

R wrapper for function choosing empirical threshold π_T using Monte Carlo simulation for single change-point Sparsified Binary Segmentation. More specifically, the function returns the empirical upper tol quantile of CUSUMs over p time series, each of length n, based on N number of runs.

Usage

```
single_SBS_calibrate(
  n,
  p,
  N = 100,
  tol = 1/100,
  rescale_variance = TRUE,
  debug = FALSE
)
```

Arguments

`n`	Number of observations
`p`	Number time series
`N`	Number of Monte Carlo samples used
`tol`	False positive probability tolerance
`rescale_variance`	If TRUE, each row of the data is rescaled by a MAD estimate
`debug`	If TRUE, diagnostic prints are provided during execution

Value

Threshold

Examples

```
library(HDCD)
n = 50
p = 50
set.seed(101)

# Simulate threshold
pi_T_squared = single_SBS_calibrate(n=n,p=p,N=100, tol=1/100, rescale_variance = TRUE)
pi_T_squared


# Generating data
X = matrix(rnorm(n*p), ncol = n, nrow=p)
# Adding a single sparse change-point:
X[1:5, 26:n] = X[1:5, 26:n] +1

# Run SBS
res = single_SBS(X,threshold=sqrt(pi_T_squared),rescale_variance=TRUE)
res$pos
```

Index