

# Package ‘hdbinseg’

September 9, 2025

**Type** Package

**Title** Change-Point Analysis of High-Dimensional Time Series via Binary Segmentation

**Version** 1.0.3

**Date** 2025-09-09

**Description** Binary segmentation methods for detecting and estimating multiple change-points in the mean or second-order structure of high-dimensional time series as described in Cho and Fryzlewicz (2014) <doi:10.1111/rssb.12079> and Cho (2016) <doi:10.1214/16-EJS1155>.

**Depends** R (>= 4.5.0)

**License** GPL (>= 3)

**Suggests** RcppArmadillo

**Imports** Rcpp (>= 0.12.10), foreach, iterators, doParallel

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.3.2

**NeedsCompilation** yes

**Author** Haeran Cho [aut, cre],  
Piotr Fryzlewicz [aut]

**Maintainer** Haeran Cho <haeran.cho@bristol.ac.uk>

**Repository** CRAN

**Encoding** UTF-8

**Date/Publication** 2025-09-09 12:50:06 UTC

## Contents

|                    |   |
|--------------------|---|
| dcbs.alg . . . . . | 2 |
| dcbs.thr . . . . . | 3 |
| sbs.alg . . . . .  | 4 |
| sbs.thr . . . . .  | 6 |

|              |          |
|--------------|----------|
| <b>Index</b> | <b>8</b> |
|--------------|----------|

dcbs.alg

*Double CUSUM Binary Segmentation***Description**

Perform the Double CUSUM Binary Segmentation algorithm detecting change points in the mean or second-order structure of the data.

**Usage**

```
dcbs.alg(
  x,
  cp.type = c(1, 2)[1],
  phi = 0.5,
  thr = NULL,
  trim = NULL,
  height = NULL,
  tau = NULL,
  temporal = TRUE,
  scales = NULL,
  diag = FALSE,
  B = 1000,
  q = 0.01,
  do.parallel = 4
)
```

**Arguments**

|                      |  |
|----------------------|--|
| <code>x</code>       | input data matrix, with each row representing the component time series  |
| <code>cp.type</code> | <code>cp.type = 1</code> specifies change points in the mean, <code>cp.type = 2</code> specifies change points in the second-order structure   |
| <code>phi</code>     | choice of parameter for weights in Double CUSUM statistic; $0 \leq \phi \leq 1$ or $\phi = -1$ allowed with the latter leading to the DC statistic combining $\phi = 0$ and $\phi = 1/2$ , see Section 4.1 of Cho (2016) for further details   |
| <code>thr</code>     | pre-defined threshold values; when <code>thr = NULL</code> , bootstrap procedure is employed for the threshold selection; when <code>thr != NULL</code> and <code>cp.type = 1</code> , <code>length(thr)</code> should be one, if <code>cp.type = 2</code> , <code>length(thr)</code> should match <code>length(scales)</code> |
| <code>trim</code>    | length of the intervals trimmed off around the change point candidates; <code>trim = NULL</code> activates the default choice ( <code>trim = round(log(dim(x)[2]))</code> )  |
| <code>height</code>  | maximum height of the binary tree; <code>height = NULL</code> activates the default choice ( <code>height = floor(log(dim(x)[2], 2)/2)</code> )  |
| <code>tau</code>     | a vector containing the scaling constant for each row of <code>x</code> ; if <code>tau = NULL</code> , a data-driven choice is made which takes into account the presence of possibly multiple mean shifts and temporal dependence when <code>temporal = TRUE</code>   |

|             |  |
|-------------|--|
| temporal    | used when <code>cp.type = 1</code> ; if <code>temporal = FALSE</code> , rows of <code>x</code> are scaled by <code>mad</code> estimates, if <code>temporal = TRUE</code> , their long-run variance estimates are used                                  |
| scales      | used when <code>cp.type = 2</code> ; negative integers representing Haar wavelet scales to be used for computing $nrow(x) * (nrow(x) + 1) / 2$ dimensional wavelet transformation of <code>x</code> ; a small negative integer represents a fine scale |
| diag        | used when <code>cp.type = 2</code> ; if <code>diag = TRUE</code> , only changes in the diagonal elements of the autocovariance matrices are searched for   |
| B           | used when <code>is.null(thr)</code> ; number of bootstrap samples for threshold selection  |
| q           | used when <code>is.null(thr)</code> ; indicates the quantile of bootstrap test statistics to be used for threshold selection   |
| do.parallel | used when <code>is.null(thr)</code> ; number of copies of R running in parallel, if <code>do.parallel = 0</code> , <code>%do%</code> operator is used, see also <code>foreach</code>   |

### Value

S3 bin. tree object, which contains the following fields:

|      |   |
|------|---|
| tree | a <code>list</code> object containing information about the nodes at which change points are detected |
| mat  | matrix concatenation of the nodes of <code>tree</code>  |
| ecp  | estimated change points   |
| thr  | threshold used to construct the tree  |

### References

H. Cho (2016) change point detection in panel data via double CUSUM statistic. *Electronic Journal of Statistics*, vol. 10, pp. 2000–2038.

### Examples

```
x <- matrix(rnorm(10*100), nrow = 10)
dcbs.alg(x, cp.type = 1, phi=.5, temporal = FALSE, do.parallel = 0)$ecp
```

```
x <- matrix(rnorm(100*300), nrow = 100)
x[1:10, 151:300] <- x[1:10, 151:300] + 1
dcbs.alg(x, cp.type = 1, phi=-1, temporal = FALSE, do.parallel = 0)$ecp
```

---

dcbs.thr

*Bootstrapping for threshold selection in DCBS algorithm*


---

### Description

Generate thresholds for DCBS algorithm via bootstrapping

**Usage**

```
dcbs.thr(
  z,
  interval = c(1, dim(z)[2]),
  phi = 0.5,
  cp.type = 1,
  do.clean.cp = FALSE,
  temporal = TRUE,
  scales = NULL,
  diag = FALSE,
  sgn = NULL,
  B = 1000,
  q = 0.01,
  do.parallel = 4
)
```

**Arguments**

`z` input data matrix, with each row representing the component time series

`interval` a vector of two containing the start and the end points of the interval from which the bootstrap test statistics are to be calculated

`phi`, `cp.type`, `temporal`, `scales`, `diag`, `B`, `q`, `do.parallel`  
see [dcbs.alg](#)

`do.clean.cp` if `do.clean.cp = TRUE` pre-change point cleaning is performed

`sgn` if `diag = FALSE`, wavelet transformations of the cross-covariances are computed with the matching signs

**Value**

a numeric value for the threshold

---

sbs.alg

*Sparsified Binary Segmentation*

---

**Description**

Perform the Sparsified Binary Segmentation algorithm detecting change-points in the mean or second-order structure of the data.

**Usage**

```
sbs.alg(
  x,
  cp.type = c(1, 2)[1],
  thr = NULL,
  trim = NULL,
```

```

height = NULL,
tau = NULL,
temporal = TRUE,
scales = NULL,
diag = FALSE,
B = 1000,
q = 0.01,
do.parallel = 4
)

```

### Arguments

|                          |   |
|--------------------------|---|
| <code>x</code>           | input data matrix, with each row representing the component time series   |
| <code>cp.type</code>     | <code>cp.type = 1</code> specifies change-points in the mean, <code>cp.type = 2</code> specifies change-points in the second-order structure  |
| <code>thr</code>         | pre-defined threshold values; when <code>thr = NULL</code> , bootstrap procedure is employed for the threshold selection; when <code>thr != NULL</code> and <code>cp.type = 1</code> , <code>length(thr)</code> should match <code>nrow(x)</code> , if <code>cp.type = 2</code> , <code>length(thr)</code> should match <code>nrow(x)*(nrow(x)+1)/2*length(s</code> |
| <code>trim</code>        | length of the intervals trimmed off around the change-point candidates; <code>trim = NULL</code> activates the default choice ( <code>trim = round(log(dim(x)[2]))</code> )   |
| <code>height</code>      | maximum height of the binary tree; <code>height = NULL</code> activates the default choice ( <code>height = floor(log(dim(x)[2], 2)/2)</code> )   |
| <code>tau</code>         | a vector containing the scaling constant for each row of <code>x</code> ; if <code>tau = NULL</code> , a data-driven choice is made which takes into account the presence of possibly multiple mean shifts and temporal dependence when <code>temporal = TRUE</code>  |
| <code>temporal</code>    | used when <code>cp.type = 1</code> ; if <code>temporal = FALSE</code> , rows of <code>x</code> are scaled by <code>mad</code> estimates, if <code>temporal = TRUE</code> , their long-run variance estimates are used   |
| <code>scales</code>      | used when <code>cp.type = 2</code> ; negative integers representing Haar wavelet scales to be used for computing <code>nrow(x)*(nrow(x)+1)/2</code> dimensional wavelet transformation of <code>x</code> ; a small negative integer represents a fine scale   |
| <code>diag</code>        | used when <code>cp.type = 2</code> ; if <code>diag = TRUE</code> , only changes in the diagonal elements of the autocovariance matrices are searched for  |
| <code>B</code>           | used when <code>is.null(thr)</code> ; number of bootstrap samples for threshold selection   |
| <code>q</code>           | used when <code>is.null(thr)</code> ; quantile of bootstrap test statistics to be used for threshold selection  |
| <code>do.parallel</code> | used when <code>is.null(thr)</code> ; number of copies of R running in parallel, if <code>do.parallel = 0</code> , <code>%do%</code> operator is used, see also <code>foreach</code>  |

### Value

S3 `bin.tree` object, which contains the following fields:

|                   |   |
|-------------------|---|
| <code>tree</code> | a <code>list</code> object containing information about the nodes at which change-points are detected |
| <code>mat</code>  | matrix concatenation of the nodes of <code>tree</code>  |
| <code>ecp</code>  | estimated change-points   |
| <code>thr</code>  | threshold used to construct the tree  |

## References

H. Cho and P. Fryzlewicz (2014) Multiple-change-point detection for high dimensional time series via sparsified binary segmentation. *JRSSB*, vol. 77, pp. 475–507.

## Examples

```
x <- matrix(rnorm(20*300), nrow = 20)
sbs.alg(x, cp.type = 2, scales = -1, diag = TRUE, do.parallel = 0)$ecp
```

```
x <- matrix(rnorm(100*300), nrow = 100)
x[1:10, 151:300] <- x[1:10, 151:300]*sqrt(2)
sbs.alg(x, cp.type = 2, scales = -1, diag = TRUE, do.parallel = 0)$ecp
```

---

sbs.thr

*Bootstrapping for threshold selection in SBS algorithm*


---

## Description

Generate thresholds for SBS algorithm via bootstrapping

## Usage

```
sbs.thr(
  z,
  interval = c(1, dim(z)[2]),
  cp.type = 1,
  do.clean.cp = TRUE,
  scales = NULL,
  diag = FALSE,
  sgn = NULL,
  B = 1000,
  q = 0.01,
  do.parallel = 4
)
```

## Arguments

|   |   |
|---|---|
| <code>z</code>  | input data matrix, with each row representing the component time series   |
| <code>interval</code>   | a vector of two containing the start and the end points of the interval from which the bootstrap test statistics are to be calculated |
| <code>cp.type</code> , <code>scales</code> , <code>diag</code> , <code>B</code> , <code>q</code> , <code>do.parallel</code> | see <a href="#">sbs.alg</a>   |
| <code>do.clean.cp</code>  | if <code>do.clean.cp = TRUE</code> pre-change-point cleaning is performed   |
| <code>sgn</code>  | if <code>diag = FALSE</code> , wavelet transformations of the cross-covariances are computed with the matching signs                  |

**Value**

if `cp.type = 1`, a vector of length `nrow(z)`, each containing the threshold applied to the CUSUM statistics from the corresponding coordinate of `z` if `cp.type = 2`, a vector of length `length(scales)*nrow(z)` (when `diag = TRUE`) or `length(scales)*nrow(z)*(nrow(z)+1)/2` (when `diag = FALSE`), each containing the threshold applied to the CUSUM statistics of the corresponding wavelet transformation of `z`

# Index

dcbs.alg, 2, 4

dcbs.thr, 3

foreach, 3, 5

list, 3, 5

mad, 3, 5

sbs.alg, 4, 6

sbs.thr, 6