Package: changepoints (via r-universe)

October 8, 2024

Type Package

Title A Collection of Change-Point Detection Methods

Version 1.1.0

Date 2022-08-25

Maintainer Haotian Xu <haotian.xu@uclouvain.be>

Description Performs a series of offline and/or online change-point detection algorithms for 1) univariate mean: <doi:10.1214/20-EJS1710>, <arXiv:2006.03283>; 2) univariate polynomials: <doi:10.1214/21-EJS1963>; 3) univariate and multivariate nonparametric settings: <doi:10.1214/21-EJS1809>, <doi:10.1109/TIT.2021.3130330>; 4) high-dimensional covariances: <doi:10.3150/20-BEJ1249>; 5) high-dimensional networks with and without missing values: <doi:10.1214/20-AOS1953>, <arXiv:2101.05477>, <arXiv:2110.06450>; 6) high-dimensional linear regression models: <arXiv:2010.10410>, <arXiv:2207.12453>; 7) high-dimensional vector autoregressive models: <arXiv:1909.06359>; 8) high-dimensional self exciting point processes: <arXiv:2006.03572>; 9) dependent dynamic nonparametric random dot product graphs: <arXiv:1911.07494>; 10) univariate mean against adversarial attacks: <arXiv:2105.10417>.

Depends R (>= 3.5.0)

Imports stats, gglasso, glmnet, ks, MASS, data.tree, Rcpp

Suggests knitr, abind, DiagrammeR, rmarkdown

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 3)

RoxygenNote 7.2.1

Encoding UTF-8

URL https://github.com/HaotianXu/changepoints

VignetteBuilder knitr

NeedsCompilation yes

Author Haotian Xu [aut, cre], Oscar Padilla [aut], Daren Wang [aut], Mengchu Li [aut], Qin Wen [ctb]

Repository CRAN

Date/Publication 2022-09-04 14:30:08 UTC

Contents

---

aARC *Automatic adversarially robust univariate mean change point detection.*

Description

Perform the adversarially robust change point detection method with automatic selection of the contamination proportion epsilon when treating the inliner distributions as Gaussian.

Usage

```
aARC(y, t_dat, guess_true = 0.05, h, block_num = 1)
```

Arguments

y	A `numeric` vector of observations.
t_dat	A `numeric` vector of observations that is used to select epsilon in the Huber contamination model.
guess_true	A `numeric` scalar representing a guess of epsilon value.
h	An `integer` scalar representing block size.
block_num	An `integer` scalar representing number of blocks used when searching for local maximiser.

Value

An numeric vector of estimated change point locations.

Author(s)

Mengchu Li

References

Li and Yu (2021) <arXiv:2105.10417>.

Examples

```
#' ### simulate data with contamination
obs_num = 1000
D = 2
noise = 0.1 # proportion of contamination
mu0 = 0
mu1 = 1
sd =1
idmixture = rbinom(obs_num/D, 1, 1-noise)
dat = NULL
for (j in 1:D){
  for (i in 1:(obs_num/(2*D))){
    if (idmixture[i] == 1){
      dat = c(dat,rnorm(1,mu0,sd))
    }
    else{
      dat = c(dat,rnorm(1,mu1/(2*noise),0))
    }
  }
  for (i in (obs_num/(2*D)+1):(obs_num/D)){
    if (idmixture[i] == 1){
      dat = c(dat,rnorm(1,mu1,sd))
    }
    else{
      dat = c(dat,rnorm(1,mu1/(2*noise)-(1-noise)*mu1/noise,0))
    }
  }
}
plot(dat)
### perform aARC
aARC(dat, dat[1:200], h = 120)
```

ARC *Adversarially robust univariate mean change point detection.*

Description

Perform the adversarially robust change point detection method.

Usage

```
ARC(y, h, block_num = 1, epsilon, gaussian = TRUE)
```

Arguments

y	A numeric vector of observations.
h	An integer scalar representing block size.
block_num	An integer scalar representing number of blocks used when searching for local maximiser.
epsilon	A numeric scalar in (0,1) representing contamination proportion.
gaussian	A logical scalar representing whether to treat the inlier distribution (F) as Gaussian distribution.

Value

An numeric vector of estimated change point locations

Author(s)

Mengchu Li

References

Li and Yu (2021) <arXiv:2105.10417>.

Examples

```
### simulate data with contamination
obs_num = 1000
D = 2
noise = 0.1 # proportion of contamination
mu0 = 0
mu1 = 1
sd =1
idmixture = rbinom(obs_num/D, 1, 1-noise)
dat = NULL
for (j in 1:D){
  for (i in 1:(obs_num/(2*D))){
    if (idmixture[i] == 1){
      dat = c(dat,rnorm(1,mu0,sd))
    }
    else{
      dat = c(dat,rnorm(1,mu1/(2*noise),0))
    }
  }
  for (i in (obs_num/(2*D)+1):(obs_num/D)){
    if (idmixture[i] == 1){
      dat = c(dat,rnorm(1,mu1,sd))
    }
    else{
```

```
      dat = c(dat,rnorm(1,mu1/(2*noise)-(1-noise)*mu1/noise,0))
    }
  }
}
plot(dat)
### perform ARC
ARC(dat,h = 120, epsilon = 0.1)
```

BD_U *Backward detection with a robust bootstrap change point test using U-statistics for univariate mean change.*

Description

Perform the backward detection method with a robust bootstrap change point test using U-statistics on univariate data.

Usage

```
BD_U(y, M, B = 100, inter = NULL, inter_ini = TRUE)
```

Arguments

y	A numeric vector of observations.
M	An integer scalar representing initial block size of the backward detection algorithm.
B	An integer scalar representing the number of bootstrapped samples.
inter	A nuisance parameter.
inter_ini	A nuisance parameter.

Value

An numeric vector of estimated change point locations.

Author(s)

Mengchu Li

References

Yu and Chen (2019) <arXiv:1904.03372>.

`BS.cov` *Binary Segmentation for covariance change points detection through Operator Norm.*

Description

Perform binary segmentation for covariance change points detection through Operator Norm.

Usage

```
BS.cov(X, s, e, level = 0)
```

Arguments

`X`	A `numeric` matrix of observations with with horizontal axis being time, and vertical axis being dimensions.
`s`	A `integer` scalar of starting index.
`e`	A `integer` scalar of ending index.
`level`	A parameter for tracking the level at which a change point is detected. Should be fixed as 0.

Value

An object of `class` "BS", which is a `list` with the structure:

- S: A vector of estimated changepoints (sorted in strictly increasing order).
- Dval: A vector of values of CUSUM statistic based on KS distance.
- Level: A vector representing the levels at which each change point is detected.
- Parent: A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2021) <doi:10.3150/20-BEJ1249>.

See Also

`thresholdBS` for obtain change points estimation.

Examples

```
p = 10
A1 = gen.cov.mat(p, 1, "equal")
A2 = gen.cov.mat(p, 2, "diagonal")
A3 = gen.cov.mat(p, 3, "power")
X = cbind(t(MASS::mvrnorm(100, mu = rep(0, p), A1)),
          t(MASS::mvrnorm(150, mu = rep(0, p), A2)),
          t(MASS::mvrnorm(200, mu = rep(0, p), A3)))
temp = BS.cov(X, 1, 450)
thresholdBS(temp, 10)
```

`BS.uni.nonpar` *Standard binary segmentation for univariate nonparametric change points detection.*

Description

Perform standard binary segmentation for univariate nonparametric change points detection.

Usage

```
BS.uni.nonpar(Y, s, e, N, delta, level = 0)
```

Arguments

`Y`	A `numeric` matrix of observations with horizontal axis being time, and vertical axis being multiple observations on each time point.
`s`	A `integer` scalar of starting index.
`e`	A `integer` scalar of ending index.
`N`	A `integer` vector representing number of multiple observations on each time point.
`delta`	A positive `integer` scalar of minimum spacing.
`level`	Should be fixed as 0.

Value

An object of `class` "BS", which is a `list` with the following structure:

`S`	A vector of estimated change points (sorted in strictly increasing order).
`Dval`	A vector of values of CUSUM statistic based on KS distance.
`Level`	A vector representing the levels at which each change point is detected.
`Parent`	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Padilla, Yu, Wang and Rinaldo (2021) <doi:10.1214/21-EJS1809>.

See Also

`thresholdBS` for obtaining change points estimation, `tuneBSuninonpar` for a tuning version.

Examples

```
Y = t(as.matrix(c(rnorm(100, 0, 1), rnorm(100, 0, 10), rnorm(100, 0, 40))))
N = rep(1, 300)
temp = BS.uni.nonpar(Y, 1, 300, N, 5)
plot.ts(t(Y))
points(x = tail(temp$S[order(temp$Dval)],4), y = Y[,tail(temp$S[order(temp$Dval)],4)], col = "red")
```

BS.univar *Standard binary segmentation for univariate mean change points detection.*

Description

Perform standard binary segmentation for univariate mean change points detection.

Usage

```
BS.univar(y, s, e, delta = 2, level = 0)
```

Arguments

`y`	A `numeric` vector of observations.
`s`	A `integer` scalar of starting index.
`e`	A `integer` scalar of ending index.
`delta`	A positive `numeric` scalar of minimum spacing.
`level`	Should be fixed as 0.

Value

An object of `class` "BS", which is a `list` with the following structure:

`S`	A vector of estimated change point locations (sorted in strictly increasing order).
`Dval`	A vector of values of CUSUM statistic.
`Level`	A vector representing the levels at which each change point is detected.
`Parent`	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>.

See Also

`thresholdBS` for obtaining change points estimation, `tuneBSunivar` for a tuning version.

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
temp = BS.univar(y, 1, length(y), delta = 5)
plot.ts(y)
points(x = tail(temp$S[order(temp$Dval)],4),
       y = y[tail(temp$S[order(temp$Dval)],4)], col = "red")
BS_result = thresholdBS(temp, tau = 4)
BS_result
print(BS_result$BS_tree, "value")
print(BS_result$BS_tree_trimmed, "value")
cpt_hat = sort(BS_result$cpt_hat[,1]) # the threshold tau is specified to be 4
Hausdorff.dist(cpt_hat, cpt_true)
cpt_LR = local.refine.univar(cpt_hat, y)
Hausdorff.dist(cpt_LR, cpt_true)
```

---

calibrate.online.network.missing

Calibrate step for online change point detection for network data with missing values.

Description

Calibrate step for online change point detection for network data by controlling the false alarm rate at level alpha.

Usage

```
calibrate.online.network.missing(
  train_miss_list,
  train_eta_list,
  threshold_len,
  alpha_grid,
  permu_num,
  pi_lb_hat,
```

```
  pi_ub_hat,
  rho_hat,
  rank_hat,
  C_lambda = 2/3,
  delta = 5
)
```

Arguments

train_miss_list	A list of adjacency matrices (with entries being 0 or 1) with missing values being coercing to 0.
train_eta_list	A list of matrices associated with data_incomplete_list, each matrix indicates the missing entries in corresponding adjacency matrix.
threshold_len	An integer scalar of the length of tuned thresholds.
alpha_grid	A numeric vector in (0,1) representing the desired false alarm rate.
permu_num	An integer scalar of number of random permutation for calibration.
pi_lb_hat	A numeric scalar of the lower bound of the missing probability.
pi_ub_hat	A numeric scalar of the upper bound of the missing probability.
rho_hat	A numeric scalar of the sparsity parameter.
rank_hat	An integer scalar of the rank of the underlying graphon matrix.
C_lambda	A numeric scalar of an absolute constant, which is set to be 2/3 by default.
delta	An integer scalar of minimum spacing.

Value

A list with the following structure:

C_lambda	The absolute constant
rho_hat	the (estimated) sparsity parameter
rank_hat	the (estimated) rank of underlying graphon matrix
pi_lb_hat	the (estimated) lower bound of the missing probability
pi_ub_hat	the (estimated) upper bound of the missing probability
thresholds_array	A numeric array of calibrated threshold

Author(s)

Haotian Xu

References

Dubey, Xu and Yu (2021) <arxiv:2110.06450>

See Also

online.network.missing for detecting online change point.

Examples

```
p = 6 # number of nodes
rho = 0.5 # sparsity parameter
block_num = 3 # number of groups for SBM
train_obs_num = 150 # sample size for each segment
conn1_mat = rho * matrix(c(0.6,1,0.6,1,0.6,0.5,0.6,0.5,0.6), nrow = 3) # connectivity matrix
set.seed(1)
can_vec = sample(1:p, replace = FALSE) # randomly assign nodes into groups
sbm = simu.SBM(conn1_mat, can_vec, train_obs_num, symm = TRUE, self = TRUE)
train_mat = sbm$obs_mat
train_list = lapply(1:ncol(train_mat), function(t) lowertri2mat(train_mat[,t], p, diag = TRUE))
pi_mat = matrix(0.9, p, p)
train_eta_list = lapply(1:length(train_list), function(t) gen.missing(pi_mat, symm = TRUE))
train_miss_list = lapply(1:length(train_list), function(t) train_eta_list[[t]] * train_list[[t]])
pi_lb_hat = quantile(Reduce("+", train_eta_list)/train_obs_num, 0.05) # estimator of pi_lb
pi_ub_hat = quantile(Reduce("+", train_eta_list)/train_obs_num, 0.95) # estimator of pi_ub
C_lambda = 2/3
lambda = lambda.network.missing(1, length(train_miss_list), length(train_miss_list), 0.05,
                                rho = 0.509, pi_ub = pi_ub_hat, p, C_lambda)
graphon_miss_impute = softImpute.network.missing(train_miss_list, train_eta_list, lambda, 1)
graphon_miss_hat = graphon_miss_impute$u %*% diag(as.numeric(graphon_miss_impute$d)) %*%
                   t(graphon_miss_impute$v)
rho_hat = quantile(graphon_miss_hat, 0.95)
rank_hat = sum(graphon_miss_impute$d != 0)
alpha_grid = c(0.05)
permu_num = 10
threshold_len = 30
temp = calibrate.online.network.missing(train_miss_list, train_eta_list, threshold_len, alpha_grid,
                permu_num, pi_lb_hat, pi_ub_hat, rho_hat, rank_hat, C_lambda, delta = 5)
```

---

changepoints *changepoints-package: A Collections of Change-Point Detection Methods*

Description

Performs a series of offline and/or online change-point detection algorithms for 1) univariate mean; 2) univariate polynomials; 3) univariate and multivariate nonparametric settings; 4) high-dimensional covariances; 5) high-dimensional networks with and without missing values; 6) high-dimensional linear regression models; 7) high-dimensional vector autoregressive models; 8) high-dimensional self exciting point processes; 9) dependent dynamic nonparametric random dot product graphs; 10) univariate mean against adversarial attacks. For more informations, see Wang et al. (2020) <arXiv:1810.09498>; Yu et al. (2020) <arXiv:2006.03283>; Yu and Chatterjee (2020) <arXiv:2007.09910>; Padilla et al. (2021) <arXiv:1905.10019>; Padilla et al. (2019) <arXiv:1910.13289>; Wang et al. (2021) <arXiv:1712.09912>; Wang et al. (2018) <arXiv:1809.09602>; Padilla et al. (2019)

<arXiv:1911.07494>; Yu et al. (2021) <arXiv:2101.05477>; Rinaldo et al. (2020) <arXiv:2010.10410>; Wang et al. (2019) <arXiv:1909.06359>; Wang et al. (2020) <arXiv:2006.03572>; Dubey et al. (2021) <arXiv:2110.06450>; Li and Yu (2021) <arXiv:2105.10417>.

`CI.regression` *Confidence interval construction of change points for regression settings with change points.*

Description

Construct element-wise confidence interval for change points.

Usage

```
CI.regression(
  cpt_init,
  cpt_LR,
  beta_hat,
  y,
  X,
  w = 0.9,
  B = 1000,
  M,
  alpha_vec,
  rounding = TRUE
)
```

Arguments

`cpt_init`	An `integer` vector of initial changepoints estimation (sorted in strictly increasing order).
`cpt_LR`	An `integer` vector of refined changepoints estimation (sorted in strictly increasing order).
`beta_hat`	A `numeric` (px(K_hat+1))matrix of estimated regression coefficients.
`y`	A `numeric` vector of response variable.
`X`	A `numeric` matrix of covariates with vertical axis being time.
`w`	A `numeric` scalar in (0,1) representing the weight for interval truncation.
`B`	An `integer` scalar corresponding to the number of simulated two-sided Brownian motion with drift.
`M`	An `integer` scalar corresponding to the length for each side of the limiting distribution, i.e. the two-sided Brownian motion with drift.
`alpha_vec`	An `numeric` vector in (0,1) representing the vector of significance levels.
`rounding`	A `boolean` scalar representing if the confidence intervals need to be rounded into integer intervals.

Value

An length(cpt_init)-2-length(alpha_vec) array of confidence intervals.

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 5
p = 10
n = 200
cpt_true = c(70, 140)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
lambda_set = c(0.1, 0.5, 1, 2)
zeta_set = c(10, 15, 20)
temp = CV.search.DPDU.regression(y = data$y, X = data$X, lambda_set, zeta_set)
temp$test_error # test error result
# find the indices of lambda_set and zeta_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
lambda_set[min_idx[2]]
zeta_set[min_idx[1]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
beta_hat = matrix(unlist(temp$beta_hat[min_idx[1], min_idx[2]]), ncol = length(cpt_init)+1)
cpt_LR = local.refine.DPDU.regression(cpt_init, beta_hat, data$y, data$X, w = 0.9)
alpha_vec = c(0.01, 0.05, 0.1)
CI.regression(cpt_init, cpt_LR, beta_hat, data$y, data$X, w = 0.9, B = 1000, M = n, alpha_vec)
```

CV.search.DP.LR.regression

Grid search based on Cross-Validation of all tuning parameters (gamma, lambda and zeta) for regression.

Description

Perform grid search based on Cross-Validation of all tuning parameters (gamma, lambda and zeta)

Usage

```
CV.search.DP.LR.regression(
  y,
  X,
  gamma_set,
```

```
  lambda_set,
  zeta_set,
  delta,
  eps = 0.001
)
```

Arguments

y	A numeric vector of response variable.
X	A numeric matrix of covariates with vertical axis being time.
gamma_set	A numeric vector of candidate tuning parameter associated with the l0 penalty.
lambda_set	A numeric vector of candidate tuning parameter for the lasso penalty.
zeta_set	A numeric vector of candidate tuning parameter for the group lasso.
delta	A strictly positive integer scalar of minimum spacing.
eps	A numeric scalar of precision level for convergence of lasso.

Value

A list with the following structure:

cpt_hat	A list of vector of estimated changepoints (sorted in strictly increasing order)
K_hat	A list of scalar of number of estimated changepoints
test_error	A list of vector of testing errors (each row corresponding to each gamma, and each column corresponding to each lambda)
train_error	A list of vector of training errors

Author(s)

Daren Wang & Haotian Xu

References

Rinaldo, Wang, Wen, Willett and Yu (2020) <arxiv:2010.10410>

Examples

```
set.seed(123)
d0 = 8
p = 15
n = 100
cpt_true = c(30, 70)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
gamma_set = c(0.01, 0.1)
lambda_set = c(0.01, 0.1)
temp = CV.search.DP.regression(y = data$y, X = data$X, gamma_set, lambda_set, delta = 2)
temp$test_error # test error result
# find the indices of gamma_set and lambda_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
```

```
gamma_set[min_idx[1]]
lambda_set[min_idx[2]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
zeta_set = c(0.1, 1)
temp_zeta = CV.search.DP.LR.regression(data$y, data$X, gamma_set[min_idx[1]],
                  lambda_set[min_idx[2]], zeta_set, delta = 2, eps = 0.001)
min_zeta_idx = which.min(unlist(temp_zeta$test_error))
cpt_LR = local.refine.regression(cpt_init, data$y, X = data$X, zeta = zeta_set[min_zeta_idx])
Hausdorff.dist(cpt_init, cpt_true)
Hausdorff.dist(cpt_LR, cpt_true)
```

`CV.search.DP.poly` *Grid search for dynamic programming to select the tuning parameter through Cross-Validation.*

Description

Perform grid search for dynamic programming to select the tuning parameter through Cross-Validation.

Usage

```
CV.search.DP.poly(y, r, gamma_set, delta)
```

Arguments

`y`	A `numeric` vector of observations.
`r`	An `integer` scalar order of polynomials.
`gamma_set`	A `numeric` vector of candidate tuning parameter associated with the l0 penalty.
`delta`	A positive `integer` scalar of minimum spacing.

Value

A `list` with the following structure:

`cpt_hat`	A list of vector of estimated change points locations (sorted in strictly increasing order)
`K_hat`	A list of scalar of number of estimated change points
`test_error`	A list of vector of testing errors
`train_error`	A list of vector of training errors

Author(s)

Haotian Xu

References

Yu and Chatterjee (2020) <arXiv:2007.09910>

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
plot.ts(y)
gamma_set = 3:9
DP_result = CV.search.DP.poly(y, r = 2, gamma_set, delta = 5)
min_idx = which.min(DP_result$test_error)
cpt_init = unlist(DP_result$cpt_hat[min_idx])
local.refine.poly(cpt_init, y, r = 2, delta_lr = 5)
```

`CV.search.DP.regression`

Grid search based on cross-validation of dynamic programming for regression change points localisation with l_0 penalisation.

Description

Perform grid search to select tuning parameters gamma (for l_0 penalty of DP) and lambda (for lasso penalty) based on cross-validation.

Usage

```
CV.search.DP.regression(y, X, gamma_set, lambda_set, delta, eps = 0.001)
```

Arguments

`y`	A `numeric` vector of response variable.
`X`	A `numeric` matrix of covariates with vertical axis being time.
`gamma_set`	A `numeric` vector of candidate tuning parameters associated with l_0 penalty of DP.
`lambda_set`	A `numeric` vector of candidate tuning parameters for lasso penalty.
`delta`	A strictly positive `integer` scalar of minimum spacing.
`eps`	A `numeric` scalar of precision level for convergence of lasso.

Value

A `list` with the following structure:

`cpt_hat`	A list of vector of estimated change points
`K_hat`	A list of scalar of number of estimated change points
`test_error`	A list of vector of testing errors (each row corresponding to each gamma, and each column corresponding to each lambda)
`train_error`	A list of vector of training errors

Author(s)

Daren Wang

References

Rinaldo, Wang, Wen, Willett and Yu (2020) <arxiv:2010.10410>

Examples

```
d0 = 10
p = 20
n = 100
cpt_true = c(30, 70)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
gamma_set = c(0.01, 0.1, 1)
lambda_set = c(0.01, 0.1, 1, 3)
temp = CV.search.DP.regression(y = data$y, X = data$X, gamma_set, lambda_set, delta = 2)
temp$test_error # test error result
# find the indices of gamma_set and lambda_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
gamma_set[min_idx[1]]
lambda_set[min_idx[2]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
```

CV.search.DP.univar *Grid search for dynamic programming to select the tuning parameter through Cross-Validation.*

Description

Perform grid search for dynamic programming to select the tuning parameter through Cross-Validation.

Usage

```
CV.search.DP.univar(y, gamma_set, delta)
```

Arguments

y	A numeric vector of observations.
gamma_set	A numeric vector of candidate tuning parameter associated with the l_0 penalty.
delta	A positive integer scalar of minimum spacing.

Value

A list with the following structure:

cpt_hat	A list of vector of estimated change points (sorted in strictly increasing order).
K_hat	A list of scalar of number of estimated change points.
test_error	A list of vector of testing errors.
train_error	A list of vector of training errors.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
gamma_set = 1:5
DP_result = CV.search.DP.univar(y, gamma_set, delta = 5)
min_idx = which.min(DP_result$test_error)
cpt_hat = unlist(DP_result$cpt_hat[min_idx])
Hausdorff.dist(cpt_hat, cpt_true)
```

CV.search.DP.VAR1 *Grid search based on cross-validation of dynamic programming for VAR change points detection via l_0 penalty.*

Description

Perform grid search based on cross-validation of dynamic programming for VAR change points detection.

Usage

```
CV.search.DP.VAR1(DATA, gamma_set, lambda_set, delta, eps = 0.001)
```

Arguments

DATA	A numeric matrix of observations with horizontal axis being time, and vertical axis being dimensions.
gamma_set	A numeric vector of candidate tuning parameters associated with the l_0 penalty.
lambda_set	A numeric vector of candidate tuning parameters for lasso penalty.
delta	A strictly integer scalar of minimum spacing.
eps	A numeric scalar of precision level for convergence of lasso.

Value

A list with the following structure:

cpt_hat	A list of vector of estimated change points
K_hat	A list of scalar of number of estimated change points
test_error	A list of vector of testing errors (each row corresponding to each gamma, and each column corresponding to each lambda)
train_error	A list of vector of training errors

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu, Rinaldo and Willett (2019) <arxiv:1909.06359>

Examples

```
set.seed(123)
p = 10
sigma = 1
n = 20
v1 = 2*(seq(1,p,1)%%2) - 1
v2 = -v1
AA = matrix(0, nrow = p, ncol = p-2)
A1 = cbind(v1,v2,AA)*0.1
A2 = cbind(v2,v1,AA)*0.1
A3 = A1
cpt_true = c(40, 80)
data = simu.VAR1(sigma, p, 2*n+1, A1)
data = cbind(data, simu.VAR1(sigma, p, 2*n, A2, vzero=c(data[,ncol(data)])))
data = cbind(data, simu.VAR1(sigma, p, 2*n, A3, vzero=c(data[,ncol(data)])))
gamma_set = c(0.1, 0.5, 1)
lambda_set = c(0.1, 1, 3.2)
temp = CV.search.DP.VAR1(data, gamma_set, lambda_set, delta = 5)
temp$test_error # test error result
# find the indices of gamma.set and lambda.set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
Hausdorff.dist(cpt_init, cpt_true)
```

CV.search.DPDU.regression

Grid search based on cross-validation of dynamic programming for regression change points localisation with l_0 penalisation.

Description

Perform grid search to select tuning parameters gamma (for l_0 penalty of DP) and lambda (for lasso penalty) based on cross-validation.

Usage

```
CV.search.DPDU.regression(y, X, lambda_set, zeta_set, eps = 0.001)
```

Arguments

y	A numeric vector of response variable.
X	A numeric matrix of covariates with vertical axis being time.
lambda_set	A numeric vector of candidate tuning parameters for lasso penalty.
zeta_set	An integer vector of tuning parameter associated with l_0 penalty (minimum interval size).
eps	A numeric scalar of precision level for convergence of lasso.

Value

A list with the following structure:

cpt_hat	A list of vectors of estimated change points
K_hat	A list of scalars of number of estimated change points
test_error	A matrix of testing errors (each row corresponding to each gamma, and each column corresponding to each lambda)
train_error	A matrix of training errors
beta_hat	A list of matrices of estimated regression coefficients

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 5
p = 30
n = 200
cpt_true = 100
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
lambda_set = c(0.01, 0.1, 1, 2)
zeta_set = c(10, 15, 20)
temp = CV.search.DPDU.regression(y = data$y, X = data$X, lambda_set, zeta_set)
temp$test_error # test error result
# find the indices of lambda_set and zeta_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
lambda_set[min_idx[2]]
zeta_set[min_idx[1]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
beta_hat = matrix(unlist(temp$beta_hat[min_idx[1], min_idx[2]]), ncol = length(cpt_init)+1)
```

DP.poly *Dynamic programming algorithm for univariate polynomials change points detection.*

Description

Perform dynamic programming algorithm for univariate polynomials change points detection.

Usage

```
DP.poly(y, r, gamma, delta)
```

Arguments

y	A numeric vector of observations.
r	An integer scalar order of polynomials.
gamma	A numeric scalar of the tuning parameter associated with the l_0 penalty.
delta	A strictly integer scalar of minimum spacing.

Value

A list with the following structure:

An object of class "DP", which is a list with the following structure:

partition	A vector of the best partition.
yhat	A vector of mean estimation for corresponding to the best partition.
cpt	A vector of change points estimation.

Author(s)

Haotian Xu

References

Yu and Chatterjee (2020) <arXiv:2007.09910>

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
plot.ts(y)
temp = DP.poly(y, r = 2, gamma = 15, delta = 5)
temp$cpt
```

`DP.regression` *Dynamic programming algorithm for regression change points localisation with l_0 penalisation.*

Description

Perform dynamic programming algorithm for regression change points localisation.

Usage

```
DP.regression(y, X, gamma, lambda, delta, eps = 0.001)
```

Arguments

`y`	A `numeric` vector of response variable.
`X`	A `numeric` matrix of covariates with vertical axis being time.
`gamma`	A positive `numeric` scalar stands for tuning parameter associated with l_0 penalty.
`lambda`	A positive `numeric` scalar stands for tuning parameter associated with the lasso penalty.
`delta`	A positive `integer` scalar stands for minimum spacing.
`eps`	A `numeric` scalar of precision level for convergence of lasso.

Value

An object of `class` "DP", which is a `list` with the following structure:

`partition`	A vector of the best partition.
`cpt`	A vector of change points estimation.

Author(s)

Daren Wang & Haotian Xu

References

Rinaldo, Wang, Wen, Willett and Yu (2020) <arxiv:2010.10410>

Examples

```
d0 = 10
p = 20
n = 100
cpt_true = c(30, 70)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
temp = DP.regression(y = data$y, X = data$X, gamma = 2, lambda = 1, delta = 5)
cpt_hat = temp$cpt
```

DP.SEPP *Dynamic programming for SEPP change points detection through l_0 penalty.*

Description

Perform dynamic programming for SEPP change points detection.

Usage

```
DP.SEPP(DATA, gamma, lambda, delta, delta2, intercept, threshold)
```

Arguments

DATA	A numeric matrix of observations with horizontal axis being time.
gamma	A numeric scalar of tuning parameter associated with the l_0 penalty.
lambda	A numeric scalar of tuning parameter for lasso penalty.
delta	An integer scalar of minimum spacing for dynamic programming.
delta2	An integer scalar representing the maximal of the change point spacing (for reducing computation cost).
intercept	A numeric scalar representing the intercept of the model, which is assumed to be known.
threshold	A numeric scalar representing the upper bound for each coordinate of X_t (for stability).

Value

An object of class "DP", which is a list with the following structure:

partition	A vector of the best partition.
cpt	A vector of change points estimation.

Author(s)

Daren Wang & Haotian Xu

References

Wang, D., Yu, Y., & Willett, R. (2020). Detecting Abrupt Changes in High-Dimensional Self-Exciting Poisson Processes. arXiv preprint arXiv:2006.03572.

Examples

```
p = 8 # dimension
n = 15
s = 3 # s is sparsity
factor = 0.2 # large factor gives exact recovery
threshold = 4 # thresholding makes the process stable
intercept = 1/2 # intercept of the model. Assume to be known as in the existing literature
A1 = A2 = A3 = matrix(0, p, p)
diag(A1[,-1]) = 1
diag(A1) = 1
diag(A1[-1,]) = -1
A1 = A1*factor
A1[(s+1):p, (s+1):p] = 0
diag(A2[,-1]) = 1
diag(A2) = -1
diag(A2[-1,]) = 1
A2 = A2*factor
A2[(s+1):p, (s+1):p] = 0
data1 = simu.SEPP(intercept, n, A1, threshold, vzero = NULL)
data2 = simu.SEPP(intercept, n, A2, threshold, vzero = data1[,n])
data = cbind(data1, data2)
gamma = 0.1
delta = 0.5*n
delta2 = 1.5*n
intercept = 1/2
threshold = 6
DP_result = DP.SEPP(data, gamma = gamma, lambda = 0.03, delta, delta2, intercept, threshold)
cpt_hat = DP_result$cpt
```

`DP.univar` *Dynamic programming for univariate mean change points detection through l_0 penalty.*

Description

Perform dynamic programming for univariate mean change points detection.

Usage

```
DP.univar(y, gamma, delta)
```

Arguments

`y`	A `numeric` vector of observations.
`gamma`	A `numeric` scalar of the tuning parameter associated with l_0 penalty.
`delta`	A positive `integer` scalar of minimum spacing.

Value

An object of class "DP", which is a list with the following structure:

partition	A vector of the best partition.
yhat	A vector of mean estimation for corresponding to the best partition.
cpt	A vector of change points estimation.

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>

Examples

```
set.seed(123)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(1,30),rep(0,120),rep(1,130))
DP_result = DP.univar(y, gamma = 5, delta = 5)
cpt_hat = DP_result$cpt
Hausdorff.dist(cpt_hat, cpt_true)
```

DP.VAR1 *Dynamic programming for VAR1 change points detection through l_0 penalty.*

Description

Perform dynamic programming for VAR1 change points detection through l_0 penalty.

Usage

```
DP.VAR1(X_futu, X_curr, gamma, lambda, delta, eps = 0.001)
```

Arguments

X_futu	A numeric matrix of time series at one step ahead, with horizontal axis being time.
X_curr	A numeric matrix of time series at current step, with horizontal axis being time.
gamma	A numeric scalar of the tuning parameter associated with l_0 penalty.
lambda	A numeric scalar of the tuning parameter for lasso penalty.
delta	A strictly integer scalar of minimum spacing.
eps	A numeric scalar of precision level for convergence of lasso.

Value

An object of class "DP", which is a list with the following structure:

partition	A vector of the best partition.
cpt	A vector of change points estimation.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu, Rinaldo and Willett (2019) <arxiv:1909.06359>

Examples

```
p = 10
sigma = 1
n = 20
v1 = 2*(seq(1,p,1)%%2) - 1
v2 = -v1
AA = matrix(0, nrow = p, ncol = p-2)
A1 = cbind(v1,v2,AA)*0.1
A2 = cbind(v2,v1,AA)*0.1
A3 = A1
data = simu.VAR1(sigma, p, 2*n+1, A1)
data = cbind(data, simu.VAR1(sigma, p, 2*n, A2, vzero=c(data[,ncol(data)])))
data = cbind(data, simu.VAR1(sigma, p, 2*n, A3, vzero=c(data[,ncol(data)])))
N = ncol(data)
X_curr = data[,1:(N-1)]
X_futu = data[,2:N]
DP_result = DP.VAR1(X_futu, X_curr, gamma = 1, lambda = 1, delta = 5)
DP_result$cpt
```

DPDU.regression *Dynamic programming with dynamic update algorithm for regression change points localisation with l_0 penalisation.*

Description

Perform DPDU algorithm for regression change points localisation.

Usage

```
DPDU.regression(y, X, lambda, zeta, eps = 0.001)
```

Arguments

y	A numeric vector of response variable.
X	A numeric matrix of covariates with vertical axis being time.
lambda	A positive numeric scalar of tuning parameter for lasso penalty.
zeta	A positive integer scalar of tuning parameter associated with l_0 penalty (minimum interval size).
eps	A numeric scalar of precision level for convergence of lasso.

Value

An object of class "DP", which is a list with the following structure:

partition	A vector of the best partition.
cpt	A vector of change points estimation.

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 10
p = 20
n = 100
cpt_true = c(30, 70)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
temp = DPDU.regression(y = data$y, X = data$X, lambda = 1, zeta = 20)
cpt_hat = temp$cpt
```

---

gen.cov.mat *Generate population covariance matrix with dimension p.*

Description

Generate population covariance matrix with dimension p.

Usage

```
gen.cov.mat(p, sigma2, type)
```

Arguments

p	A integer scalar of dimensionality.
sigma2	A positive numeric scalar representing the variance of each entry.
type	Specify the type of a covariance matrix: Diagonal structure ("diagonal"); Equal correlation structure ("equal"); Power decay structure ("power").

Value

A numeric p-by-p matrix.

Author(s)

Haotian Xu

Examples

```
gen.cov.mat(p = 5, sigma2 = 1, type = "diagonal")
```

---

gen.missing *Function to generate a matrix with values 0 or 1, where 0 indicating the entry is missing*

Description

Function to generate a matrix with values 0 or 1, where 0 indicating the entry is missing

Usage

```
gen.missing(pi_mat, symm = TRUE)
```

Arguments

pi_mat	A numeric pxp matrix, for each entry, the value representing the probability of missing.
symm	A logic scalar indicating if the output matrix needs to be symmetric.

Value

A numeric p x p matrix.

Author(s)

Haotian Xu

Examples

```
p = 5
pi_mat = matrix(0.9, p, p)
eta_mat = gen.missing(pi_mat, symm = TRUE)
```

`gen.piece.poly` *Generate univariate data from piecewise polynomials of degree at most r.*

Description

Generate univariate data from piecewise polynomials (currently, only the linear, quadratic functions and cubic functions are considered).

Usage

```
gen.piece.poly(init_coef_vec, cpt_vec, kappa_mat, n, sigma)
```

Arguments

`init_coef_vec`	A (r+1)-dim `numeric` vector of coefficients for the first segment.
`cpt_vec`	A K-dim `integer` vector of change points.
`kappa_mat`	A (r+1)xK `numeric` matrix where the i-th column represents the jump sizes for coefficients associated with the i-th change point.
`n`	An `integer` scalar of sample size.
`sigma`	A `numeric` scalar of standard deviation of error terms.

Value

A vector of data generated from piecewise polynomials.

Author(s)

Haotian Xu

References

Yu and Chatterjee (2020) <arXiv:2007.09910>.

Examples

```
r = 2
init_coef_vec = c(-2, 2, 9)
cpt_true = c(100, 200)
n = 300
sigma = 1
kappa_mat = cbind(c(3, 9, -27), c(-3, 9, -27))
plot.ts(gen.piece.poly(init_coef_vec, cpt_true, kappa_mat, n, sigma), ylab = "y")
```

gen.piece.poly.noiseless *Mean function of piecewise polynomials.*

Description

Compute mean function of piecewise polynomials (currently, only the linear, quadratic functions and cubic functions are considered).

Usage

```
gen.piece.poly.noiseless(init_coef_vec, cpt_vec, kappa_mat, n)
```

Arguments

init_coef_vec	A numeric vector of coefficients for the first segment.
cpt_vec	An integer vector of change points.
kappa_mat	A numeric matrix where the i-th column represents the jump sizes for coefficients associated with the i-th change point.
n	An integer scalar of sample size.

Value

A vector of mean function of piecewise polynomials.

Author(s)

Haotian Xu

References

Yu and Chatterjee (2020) <arXiv:2007.09910>

Examples

```
r = 2
init_coef_vec = c(-2, 2, 9)
cpt_true = c(100, 200)
n = 300
kappa_mat = cbind(c(3, 9, -27), c(-3, 9, -27))
plot.ts(gen.piece.poly.noiseless(init_coef_vec, cpt_true, kappa_mat, n),
        ylab = "Values of piecewise polynomials")
```

Hausdorff.dist *Bidirectional Hausdorff distance.*

Description

Compute the bidirectional Hausdorff distance between two sets.

Usage

```
Hausdorff.dist(vec1, vec2)
```

Arguments

vec1	A integer vector forms a subset of 1, 2, ..., n.
vec2	A integer vector forms a subset of 1, 2, ..., n.

Value

An integer scalar of bidirectional Hausdorff distance.

Author(s)

Daren Wang

Examples

```
vec1 = sample.int(1000, size = 50)
vec2 = sample.int(2000, size = 100)
Hausdorff.dist(vec1, vec2)
```

huber_mean *Element-wise adaptive Huber mean estimator.*

Description

Computes the element-wise adaptive Huber mean estimator.

Usage

```
huber_mean(x, tau)
```

Arguments

x	A numeric vector of observations.
tau	A numeric scalar corresponding to the robustification parameter (larger than 0).

Value

A numeric scalar corresponding to the adaptive Huber mean estimator.

Author(s)

Haotian Xu

Examples

```
set.seed(123)
y = rnorm(100)
mean(y)
huber_mean(y, 1.345)
```

---

lambda.network.missing

Function to compute the default thresholding parameter for leading singular value in the soft-impute algorithm.

Description

Function to compute the default thresholding parameter for leading singular value in the soft-impute algorithm.

Usage

```
lambda.network.missing(s, e, t, alpha, rho, pi_ub, p, C_lambda)
```

Arguments

s	An integer scalar of the starting index.
e	An integer scalar of the ending index.
t	An integer scalar of the splitting index.
alpha	A numeric scalar in (0,1) representing the desired false alarm rate.
rho	A numeric scalar of the sparsity parameter.
pi_ub	A numeric scalar of the upper bound of the missing probability.
p	An integer scalar of the dimensionality of the graphon matrix.
C_lambda	A numeric scalar of an absolute constant, which is set to be 2/3 by default.

Details

The default thresholding parameter is given in Theorem 2 of the reference.

Value

The default thresholding parameter for leading singular value in the soft-impute algorithm

References

Dubey, Xu and Yu (2021) <arxiv:2110.06450>

`local.refine.CV.VAR1` *Local refinement for VAR1 change points detection.*

Description

Perform local refinement for VAR1 change points detection.

Usage

```
local.refine.CV.VAR1(cpt_init, DATA, zeta_set, delta_local)
```

Arguments

`cpt_init`	A `integer` vector of initial change points estimation (sorted in strictly increasing order).
`DATA`	A `numeric` matrix of observations with with horizontal axis being time, and vertical axis being dimensions.
`zeta_set`	A `numeric` vector of candidate tuning parameters for group lasso penalty.
`delta_local`	A strictly `integer` scalar of minimum spacing for group lasso.

Value

A `list` with the following structure:

`cpt_hat`	A vector of estimated change point locations (sorted in strictly increasing order)
`zeta`	A scalar of selected zeta by cross-validation

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu, Rinaldo and Willett (2019) <arxiv:1909.06359>

`local.refine.DPDU.regression` *Local refinement for DPDU regression change points localisation.*

Description

Perform local refinement for regression change points localisation.

Usage

```
local.refine.DPDU.regression(cpt_init, beta_hat, y, X, w = 0.9)
```

Arguments

`cpt_init`	An `integer` vector of initial changepoints estimation (sorted in strictly increasing order).
`beta_hat`	A `numeric` (px(K_hat+1))matrix of estimated regression coefficients.
`y`	A `numeric` vector of response variable.
`X`	A `numeric` matrix of covariates with vertical axis being time..
`w`	A `numeric` scalar in (0,1) representing the weight for interval truncation.

Value

A vector of locally refined change points estimation.

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 5
p = 30
n = 200
cpt_true = 100
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
lambda_set = c(0.01, 0.1, 1, 2)
zeta_set = c(10, 15, 20)
temp = CV.search.DPDU.regression(y = data$y, X = data$X, lambda_set, zeta_set)
temp$test_error # test error result
# find the indices of lambda_set and zeta_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
```

```
lambda_set[min_idx[2]]
zeta_set[min_idx[1]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
beta_hat = matrix(unlist(temp$beta_hat[min_idx[1], min_idx[2]]), ncol = length(cpt_init)+1)
cpt_refined = local.refine.DPDU.regression(cpt_init, beta_hat, data$y, data$X, w = 0.9)
```

`local.refine.network` *Local refinement for network change points detection.*

Description

Perform local refinement for network change points detection.

Usage

```
local.refine.network(
  cpt_init,
  data_mat1,
  data_mat2,
  self = FALSE,
  w = 0.5,
  tau2,
  tau3 = Inf
)
```

Arguments

`cpt_init`	A `integer` vector of initial change points estimation (sorted in strictly increasing order).
`data_mat1`	A `numeric` matrix of observations with with horizontal axis being time, and with each column be the vectorized adjacency matrix.
`data_mat2`	A independent copy of data_mat1.
`self`	A `logic` scalar indicating if adjacency matrices are required to have self-loop.
`w`	A `numeric` scalar in (0,1) indicating the level of shrinkage (large shrinkage if w is small) on the interval between the (k-1)th and (k+1)th preliminary change-point estimator.
`tau2`	A positive `numeric` scalar for USVT corresponding to the threshold for singular values of input matrix.
`tau3`	A positive `numeric` scalar for USVT corresponding to the threshold for entries of output matrix.

Value

A `numeric` vector of locally refined change point locations.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu and Rinaldo (2018) <arxiv:1809.09602>.

Examples

```
p = 15 # number of nodes
rho = 0.5 # sparsity parameter
block_num = 3 # number of groups for SBM
n = 100 # sample size for each segment
# connectivity matrix for the first and the third segments
conn1_mat = rho * matrix(c(0.6,1,0.6,1,0.6,0.5,0.6,0.5,0.6), nrow = 3)
# connectivity matrix for the second segment
conn2_mat = rho * matrix(c(0.6,0.5,0.6,0.5,0.6,1,0.6,1,0.6), nrow = 3)
set.seed(1)
can_vec = sample(1:p, replace = FALSE) # randomly assign nodes into groups
sbm1 = simu.SBM(conn1_mat, can_vec, n, symm = TRUE, self = TRUE)
sbm2 = simu.SBM(conn2_mat, can_vec, n, symm = TRUE, self = TRUE)
data_mat = cbind(sbm1$obs_mat, sbm2$obs_mat)
data_mat1 = data_mat[,seq(1,ncol(data_mat),2)]
data_mat2 = data_mat[,seq(2,ncol(data_mat),2)]
M = 10
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(data_mat1))
temp = WBS.network(data_mat1, data_mat2, 1, ncol(data_mat1), intervals$Alpha,
                   intervals$Beta, delta = 5)
rho_hat = quantile(rowMeans(data_mat), 0.95)
tau = p*rho_hat*(log(n))^2/20 # default threshold given in the paper
cpt_init = unlist(thresholdBS(temp, tau)$cpt_hat[,1])
cpt_refined = local.refine.network(cpt_init, data_mat1, data_mat2, self = TRUE,
                                   tau2 = p*rho_hat/3, tau3 = Inf)
cpt_WBS = 2*cpt_init
cpt_refined = 2*cpt_refined
```

`local.refine.poly` *Local refinement for univariate polynomials change point detection.*

Description

Perform local refinement for univariate polynomials change point detection.

Usage

```
local.refine.poly(cpt_init, y, r, delta_lr)
```

Arguments

cpt_init	An integer vector of initial change points estimation (sorted in strictly increasing order).
y	A numeric vector of univariate time series.
r	An integer scalar order of polynomials.
delta_lr	A positive integer scalar of minimum spacing for local refinement.

Value

An integer vector of locally refined change point estimation.

Author(s)

Haotian Xu

References

Yu and Chatterjee (2020) <arXiv:2007.09910>

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
plot.ts(y)
gamma_set = 3:9
DP_result = CV.search.DP.poly(y, r = 2, gamma_set, delta = 5)
min_idx = which.min(DP_result$test_error)
cpt_init = unlist(DP_result$cpt_hat[min_idx])
local.refine.poly(cpt_init, y, r = 2, delta_lr = 5)
```

---

local.refine.regression

Local refinement for regression change points localisation.

Description

Perform local refinement for regression change points localisation.

Usage

```
local.refine.regression(cpt_init, y, X, zeta)
```

Arguments

cpt_init	An integer vector of initial changepoints estimation (sorted in strictly increasing order).
y	A numeric vector of response variable.
X	A numeric matrix of covariates with vertical axis being time..
zeta	A numeric scalar of tuning parameter for the group lasso.

Value

A vector of locally refined change points estimation.

Author(s)

Daren Wang & Haotian Xu

References

Rinaldo, A., Wang, D., Wen, Q., Willett, R., & Yu, Y. (2021, March). Localizing changes in high-dimensional regression models. In International Conference on Artificial Intelligence and Statistics (pp. 2089-2097). PMLR.

Rinaldo, Wang, Wen, Willett and Yu (2020) <arxiv:2010.10410>

Examples

```
d0 = 10
p = 20
n = 100
cpt_true = c(30, 70)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
gamma_set = c(0.01, 0.1, 1)
lambda_set = c(0.01, 0.1, 1, 3)
temp = CV.search.DP.regression(y = data$y, X = data$X, gamma_set, lambda_set, delta = 2)
temp$test_error # test error result
# find the indices of gamma_set and lambda_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
gamma_set[min_idx[1]]
lambda_set[min_idx[2]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
local.refine.regression(cpt_init, data$y, X = data$X, zeta = 0.5)
```

---

local.refine.univar *Local refinement of an initial estimator for univariate mean change points detection.*

Description

Perform local refinement for univariate mean change points detection.

Usage

```
local.refine.univar(cpt_init, y)
```

Arguments

cpt_init	An integer vector of initial change points estimation (sorted in strictly increasing order).
y	A numeric vector of univariate time series.

Value

An integer vector of locally refined change point estimation.

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>.

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
gamma_set = 1:5
DP_result = CV.search.DP.univar(y, gamma_set, delta = 5)
min_idx = which.min(DP_result$test_error)
cpt_hat = unlist(DP_result$cpt_hat[min_idx])
Hausdorff.dist(cpt_hat, cpt_true)
cpt_LR = local.refine.univar(cpt_hat, y)
Hausdorff.dist(cpt_LR, cpt_true)
```

---

local.refine.VAR1 *Local refinement for VAR1 change points detection.*

Description

Perform local refinement for VAR1 change points detection.

Usage

```
local.refine.VAR1(cpt_init, DATA, zeta)
```

Arguments

cpt_init	A integer vector of initial change points estimation (sorted in strictly increasing order).
DATA	A numeric matrix of observations with with horizontal axis being time, and vertical axis being dimensions.
zeta	A numeric scalar of lasso penalty.

Value

An integer vector of locally refined change points estimation.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu, Rinaldo and Willett (2019) <arxiv:1909.06359>.

See Also

local.refine.CV.VAR1.

---

lowertri2mat *Transform a vector containing lower diagonal entries into a symmetric matrix of dimension p.*

Description

Transform a vector containing lower diagonal entries into a symmetric matrix of dimension p.

Usage

```
lowertri2mat(lowertri_vec, p, diag = FALSE)
```

Arguments

lowertri_vec	A numeric vector containing lower diagonal entries.
p	A integer scalar of dimensionality.
diag	A logic scalar indicating if the diagonal entries are contained in lowertri_vec.

Value

A numeric p x p symmetric matrix.

Author(s)

Haotian Xu

Examples

```
A = matrix(1:16, 4, 4)
B = lowertri2mat(A[lower.tri(A)], 4, diag = FALSE)
C = lowertri2mat(A[lower.tri(A, diag = TRUE)], 4, diag = TRUE)
```

LRV.regression — *Long-run variance estimation for regression settings with change points.*

Description

Estimating long-run variance for regression settings with change points.

Usage

```
LRV.regression(cpt_init, beta_hat, y, X, w = 0.9, block_size)
```

Arguments

cpt_init	An integer vector of initial changepoints estimation (sorted in strictly increasing order).
beta_hat	A numeric (px(K_hat+1))matrix of estimated regression coefficients.
y	A numeric vector of response variable.
X	A numeric matrix of covariates with vertical axis being time.
w	A numeric scalar in (0,1) representing the weight for interval truncation.
block_size	An integer scalar corresponding to the block size S in the paper.

Value

A vector of long-run variance estimators associated with all local refined intervals.

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 5
p = 10
n = 200
cpt_true = c(70, 140)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
lambda_set = c(0.1, 0.5, 1, 2)
zeta_set = c(10, 15, 20)
temp = CV.search.DPDU.regression(y = data$y, X = data$X, lambda_set, zeta_set)
temp$test_error # test error result
# find the indices of lambda_set and zeta_set which minimizes the test error
min_idx = as.vector(arrayInd(which.min(temp$test_error), dim(temp$test_error)))
lambda_set[min_idx[2]]
zeta_set[min_idx[1]]
cpt_init = unlist(temp$cpt_hat[min_idx[1], min_idx[2]])
beta_hat = matrix(unlist(temp$beta_hat[min_idx[1], min_idx[2]]), ncol = length(cpt_init)+1)
interval_refine = trim_interval(n, cpt_init)
# choose S
block_size = ceiling(sqrt(min(floor(interval_refine[,2]) - ceiling(interval_refine[,1])))/2)
LRV_est = LRV.regression(cpt_init, beta_hat, data$y, data$X, w = 0.9, block_size)
```

`online.network` *Online change point detection for network data.*

Description

Perform online change point detection for network data by controlling the false alarm rate at level alpha or controlling the average run length gamma. The default choice of the tuning parameters tau1, tau2 and tau3 are used (see Section 4.1 of the reference).

Usage

```
online.network(
  data_mat1,
  data_mat2,
  self = TRUE,
  b_vec = NULL,
  train_mat = NULL,
  alpha = NULL,
  gamma = NULL,
  permu_num = NULL
)
```

Arguments

`data_mat1`	A `numeric` matrix of observations with with horizontal axis being time, and with each column be the vectorized adjacency matrix.

data_mat2	A numeric matrix of observations with with horizontal axis being time, and with each column be the vectorized adjacency matrix (data_mat1 and data_mat2 are independent and have the same dimensions).
self	A logic scalar indicating if adjacency matrices are required to have self-loop.
b_vec	A numeric vector of thresholds b_t with t >= 2.
train_mat	A numeric matrix of training data from a pre-change distribution(no change point), which is only needed to when b_vec is NULL in order to calibrate b_t.
alpha	A numeric scalar in (0,1) representing the level.
gamma	An integer scalar of desired average run length.
permu_num	An integer scalar of number of random permutation for calibration.

Value

A list with the following structure:

cpt	Estimated change point
score	A numeric vector of computed cumsum statistics
b_vec	A numeric vector of thresholds b_t with t >= 2

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Yu, Padilla, Wang and Rinaldo (2021) <arxiv:2101.05477>

Examples

```
set.seed(123)
p = 15 # number of nodes
rho = 0.5 # sparsity parameter
block_num = 3 # number of groups for SBM
n = 100 # sample size for each segment
# connectivity matrix for the first and the third segments
conn1_mat = rho * matrix(c(0.6,1,0.6,1,0.6,0.5,0.6,0.5,0.6), nrow = 3)
# connectivity matrix for the second segment
conn2_mat = rho * matrix(c(0.6,0.5,0.6,0.5,0.6,1,0.6,1,0.6), nrow = 3)
set.seed(1)
can_vec = sample(1:p, replace = FALSE) # randomly assign nodes into groups
sbm1 = simu.SBM(conn1_mat, can_vec, n, symm = TRUE, self = TRUE)
sbm2 = simu.SBM(conn2_mat, can_vec, n, symm = TRUE, self = TRUE)
data_mat = cbind(sbm1$obs_mat, sbm2$obs_mat)
data_mat1 = data_mat[,seq(1,ncol(data_mat),2)]
data_mat2 = data_mat[,seq(2,ncol(data_mat),2)]
train_mat = simu.SBM(conn1_mat, can_vec, n = 150, symm = TRUE, self = TRUE)$obs_mat
temp = online.network(data_mat1, data_mat2, self = TRUE, b_vec = NULL, train_mat, alpha = 0.05,
                      gamma = NULL, permu_num = 20)
cpt_hat = 2 * temp$cpt
```

`online.network.missing` *Online change point detection for network data with missing values.*

Description

Perform online change point detection for network with missing values by controlling the false alarm rate at level alpha.

Usage

```
online.network.missing(
  data_incomplete_list,
  eta_list,
  alpha_grid,
  thresholds_array,
  rho_hat,
  pi_ub_hat,
  C_lambda = 2/3,
  delta = 5
)
```

Arguments

`data_incomplete_list`	A `list` of adjacency matrices (with entries being 0 or 1) with missing values being coercing to 0.
`eta_list`	A `list` of matrices associated with data_incomplete_list, each matrix indicates the missing entries in corresponding adjacency matrix.
`alpha_grid`	A `numeric` vector in (0,1) representing the desired false alarm rate.
`thresholds_array`	A `numeric` array of calibrated thresholds.
`rho_hat`	A `numeric` scalar of the sparsity parameter.
`pi_ub_hat`	A `numeric` scalar of the upper bound of the missing probability.
`C_lambda`	A `numeric` scalar of an absolute constant, which is set to be 2/3 by default.
`delta`	An `integer` scalar of minimum spacing.

Value

Online change point estimator.

Author(s)

Haotian Xu

References

Dubey, Xu and Yu (2021) <arxiv:2110.06450>

See Also

`calibrate.online.network.missing` for calibrating thresholds.

`online.univar` *Online change point detection with controlled false alarm rate or average run length.*

Description

Perform online change point detection with controlled false alarm rate or average run length.

Usage

```
online.univar(
  y_vec,
  b_vec = NULL,
  train_vec = NULL,
  alpha = NULL,
  gamma = NULL,
  permu_num = NULL
)
```

Arguments

`y_vec`	A `numeric` vector of observations.
`b_vec`	A `numeric` vector of thresholds b_t with t >= 2.
`train_vec`	A `numeric` vector of training data from a pre-change distribution (no change point), which is only needed to when b_vec is NULL in order to calibrate b_t.
`alpha`	A `numeric` scalar of desired false alarm rate.
`gamma`	An `integer` scalar of desired average run length.
`permu_num`	An `integer` scalar of number of random permutation for calibration.

Value

A `list` with the following structure:

`cpt_hat`	An `integer` scalar of estimated change point location
`b_vec`	A `numeric` vector of thresholds b_t with t >= 2

Author(s)

Haotian Xu

References

Yu, Padilla, Wang and Rinaldo (2020) <arxiv:2006.03283>

Examples

```
y_vec = rnorm(150) + c(rep(0, 100), rep(1, 50))
train_vec = rnorm(100)
# control the false alarm rate
temp1 = online.univar(y_vec = y_vec, train_vec = train_vec, alpha = 0.05, permu_num = 20)
temp1$cpt_hat
temp1$b_vec # calibrated threshold
```

`online.univar.multi` *Online change point detection with potentially multiple change points.*

Description

Perform Online change point detection with potentially multiple change points.

Usage

```
online.univar.multi(
  y_vec,
  b_vec = NULL,
  train_vec = NULL,
  alpha = NULL,
  gamma = NULL,
  permu_num = NULL
)
```

Arguments

`y_vec`	A `numeric` vector of observations.
`b_vec`	A `numeric` vector of thresholds b_t with t >= 2.
`train_vec`	A `numeric` vector of training data from a pre-change distribution (no change point), which is only needed to when b_vec is NULL in order to calibrate b_t.
`alpha`	A `numeric` scalar of desired false alarm rate.
`gamma`	An `integer` scalar of desired average run length.
`permu_num`	An `integer` scalar of number of random permutation for calibration.

Details

#' @title Online change point detection with controlled average run length. #' @description Perform online change point detection via CUSUM (single change point, type 2). #' @param y_vec A `numeric` vector of observations. #' @param gamma A `integer` scalar of interval length (>= 2). #' @param tau_gamma A `numeric` scalar of threshold. #' @param ... Additional arguments. #' @return An `integer` scalar of estimated change points location. #' @export #' @author Haotian Xu #' @examples #' TO DO online.univar.one2 = function(y_vec, gamma, tau_gamma, ...) t = 1 FLAG = 0 while(FLAG == 0 & t <= length(y_vec)) t = t + 1 e = max(t-gamma, 0) cusum_vec = sapply((e+1):(t-1), function(s) sqrt((t-s)*(s-e)/(t-e)) * abs(mean(y_vec[(e+1):s]) - mean(y_vec[(s+1):t]))) FLAG = 1 - prod(cusum_vec <= tau_gamma)

return(t)

#' @title Online change point detection via CUSUM (single change point, type 3). #' @description Perform online change point detection via CUSUM (single change point, type 3). #' @param y_vec A `numeric` vector of observations. #' @param tau_vec A `numeric` vector of thresholds at time t>= 1. #' @param ... Additional arguments. #' @return An `integer` scalar of estimated change point location. #' @export #' @author Haotian Xu #' @examples #' TO DO online.univar.one3 = function(y_vec, tau_vec, ...) if(length(y_vec) != length(tau_vec)) stop("y_vec and tau_vec should have the same length.")

t = 1 FLAG = 0 while(FLAG == 0 & t <= length(y_vec)) t = t + 1 J = floor(log2(t)) j = 0 while(j < J & FLAG == 0) j = j + 1 s_j = t - 2^(j-1) cusum = sqrt((t-s_j)*s_j/t) * abs(mean(y_vec[1:s_j]) - mean(y_vec[(s_j+1):t])) FLAG = (cusum > tau_vec[t])

return(t)

Value

An `integer` vector of estimated change points.

Author(s)

Haotian Xu

References

Yu, Padilla, Wang and Rinaldo (2020) <arxiv:2006.03283>

Examples

```
y_vec = rnorm(200) + c(rep(0, 50), rep(1, 100), rep(0, 50))
train_vec = rnorm(150)
# control the false alarm rate
temp1 = online.univar.multi(y_vec = y_vec, train_vec = train_vec, alpha = 0.05, permu_num = 20)
temp1
```

simu.change.regression *Simulate a sparse regression model with change points in coefficients.*

Description

Simulate a sparse regression model with change points in coefficients under temporal dependence.

Usage

```
simu.change.regression(
  d0,
  cpt_true,
  p,
  n,
  sigma,
  kappa,
  cov_type = "I",
  mod_X = "IID",
  mod_e = "IID"
)
```

Arguments

d0	A numeric scalar stands for the number of nonzero coefficients.
cpt_true	An integer vector contains true change points (sorted in strictly increasing order).
p	An integer scalar stands for the dimensionality.
n	An integer scalar stands for the sample size.
sigma	A numeric scalar stands for error standard deviation.
kappa	A numeric scalar stands for the minimum jump size of coefficient vector in l_2 norm.
cov_type	A character string stands for the type of covariance matrix of covariates. 'I': Identity; 'T': Toeplitz; 'E': Equal-correlation.
mod_X	A character string stands for the time series model followed by the covariates. 'IID': IID multivariate Gaussian; 'AR': Multivariate AR1 with rho = 0.3; Multivariate MA1 theta = 0.3.
mod_e	A character string stands for the time series model followed by the errors 'IID': IID univariate Gaussian; 'AR': Univariate AR1 with rho = 0.3; Univariate MA1 theta = 0.3.

Value

A `list` with the following structure:

`cpt_true`	A vector of true changepoints (sorted in strictly increasing order).
`X`	An n-by-p design matrix.
`y`	An n-dim vector of response variable.
`betafullmat`	A p-by-n matrix of coefficients.

Author(s)

Daren Wang, Zifeng Zhao & Haotian Xu

References

Rinaldo, Wang, Wen, Willett and Yu (2020) <arxiv:2010.10410>; Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

Examples

```
d0 = 10
p = 30
n = 100
cpt_true = c(10, 30, 40, 70, 90)
data = simu.change.regression(d0, cpt_true, p, n, sigma = 1, kappa = 9)
```

---

simu.RDPG *Simulate a dot product graph (without change point).*

Description

Simulate a dot product graph (without change point). The generated data is a matrix with each column corresponding to the vectorized adjacency (sub)matrix at a time point. For example, if the network matrix is required to be symmetric and without self-loop, only the strictly lower diagonal entries are considered.

Usage

```
simu.RDPG(x_mat, n, symm = TRUE, self = FALSE)
```

Arguments

`x_mat`	A `numeric` matrix representing the latent positions with horizontal axis being latent dimensions and vertical axis being nodes (each entry takes value in $[0, 1]$).
`n`	A `integer` scalar representing the number of observations.
`symm`	A `logic` scalar indicating if adjacency matrices are required to be symmetric.
`self`	A `logic` scalar indicating if adjacency matrices are required to have self-loop.

Value

A `list` with the following structure:

`obs_mat`	A matrix, with each column be the vectorized adjacency (sub)matrix. For example, if "symm = TRUE" and "self = FALSE", only the strictly lower triangular matrix is considered.
`graphon_mat`	Underlying graphon matrix.

Author(s)

Haotian Xu

Examples

```
p = 20 # number of nodes
n = 50 # sample size for each segment
lat_dim_num = 5 # number of latent dimensions
set.seed(1)
x_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
x_tilde_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
y_mat = rbind(x_tilde_mat[1:floor(p/4),], x_mat[(floor(p/4)+1):p,])
rdpg1 = simu.RDPG(x_mat, n, symm = TRUE, self = FALSE)
rdpg2 = simu.RDPG(y_mat, n, symm = TRUE, self = FALSE)
data1_mat = rdpg1$obs_mat
data2_mat = rdpg2$obs_mat
data_mat = cbind(data1_mat, data2_mat)
```

`simu.SBM` *Simulate a Stochastic Block Model (without change point).*

Description

Simulate a Stochastic Block Model (without change point). The generated data is a matrix with each column corresponding to the vectorized adjacency (sub)matrix at a time point. For example, if the network matrix is required to be symmetric and without self-loop, only the strictly lower diagonal entries are considered.

Usage

```
simu.SBM(connec_mat, can_vec, n, symm = FALSE, self = TRUE)
```

Arguments

`connec_mat`	A `numeric` symmetric matrix representing the connectivity matrix (each entry takes value in $[0, 1]$).
`can_vec`	A `integer` p-dim vector of node indices. can_vec is then divided into subvectors corresponding to blocks.

n	A integer scalar representing the number of observations.
symm	A logic scalar indicating if adjacency matrices are required to be symmetric.
self	A logic scalar indicating if adjacency matrices are required to have self-loop.

Value

A list with the following structure:

obs_mat	A matrix, with each column be the vectorized adjacency (sub)matrix. For example, if "symm = TRUE" and "self = FALSE", only the strictly lower triangular matrix is considered.
graphon_mat	Underlying graphon matrix.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu and Rinaldo (2018) <arxiv:1809.09602>.

Examples

```
p = 15 # number of nodes
rho = 0.5 # sparsity parameter
block_num = 3 # number of groups for SBM
n = 100 # sample size for each segment
# connectivity matrix for the first and the third segments
conn1_mat = rho * matrix(c(0.6,1,0.6,1,0.6,0.5,0.6,0.5,0.6), nrow = 3)
# connectivity matrix for the second segment
conn2_mat = rho * matrix(c(0.6,0.5,0.6,0.5,0.6,1,0.6,1,0.6), nrow = 3)
set.seed(1)
can_vec = sample(1:p, replace = FALSE) # randomly assign nodes into groups
sbm1 = simu.SBM(conn1_mat, can_vec, n, symm = TRUE, self = TRUE)
sbm2 = simu.SBM(conn2_mat, can_vec, n, symm = TRUE, self = TRUE)
data_mat = cbind(sbm1$obs_mat, sbm2$obs_mat)
```

---

simu.SEPP *Simulate a (stable) SEPP model (without change point).*

Description

Simulate a (stable) SEPP model (without change point).

Usage

```
simu.SEPP(intercept, n, A, threshold, vzero = NULL)
```

Arguments

intercept	A numeric scalar representing the intercept of the model.
n	An integer scalar representing sample size.
A	A numeric pXp matrix representing the coefficient matrix.
threshold	A numeric scalar representing the upper bound for each coordinate of X_t (for stability).
vzero	A numeric vector representing the observation at time 0. If vzero = NULL, each coordinate is generated from a Poisson distribution with mean lambda.

Value

A p-by-n matrix.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu, & Willett (2020). Detecting Abrupt Changes in High-Dimensional Self-Exciting Poisson Processes. <arXiv:2006.03572>.

Examples

```
p = 10 # dimension
n = 50
s = 5 # sparsity
factor = 0.12 # large factor gives exact recovery
threshold = 4 # thresholding makes the process stable
intercept = 1/2 # intercept of the model. Assume to be known as in the existing literature
A1 = A2 = A3 = matrix(0, p, p)
diag(A1[,-1]) = 1
diag(A1) = 1
diag(A1[-1,]) = -1
A1 = A1*factor
A1[(s+1):p, (s+1):p] = 0
diag(A2[,-1]) = 1
diag(A2) = -1
diag(A2[-1,]) = 1
A2 = A2*factor
A2[(s+1):p, (s+1):p] = 0
diag(A3[,-1]) = 1
diag(A3) = 1
diag(A3[-1,]) = -1
A3 = A3*factor
A3[(s+1):p, (s+1):p] = 0
data1 = simu.SEPP(intercept, n, A1, threshold, vzero = NULL)
data2 = simu.SEPP(intercept, n, A2, threshold, vzero = data1[,n])
data3 = simu.SEPP(intercept, n, A3, threshold, vzero = data2[,n])
data = cbind(data1, data2, data3)
```

```
dim(data)
```

`simu.VAR1` *Simulate from a VAR1 model (without change point).*

Description

Simulate data of size n and dimension p from a VAR1 model (without change point) with Gaussian i.i.d. error terms.

Usage

```
simu.VAR1(sigma, p, n, A, vzero = NULL)
```

Arguments

`sigma`	A `numeric` scalar representing the standard deviation of error terms.
`p`	An `integer` scalar representing dimension.
`n`	An `integer` scalar representing sample size.
`A`	A `numeric` p-by-p matrix representing the transition matrix of the VAR1 model.
`vzero`	A `numeric` vector representing the observation at time 0. If `vzero = NULL`,it is generated following the distribution of the error terms.

Value

A p-by-n matrix.

Author(s)

Daren Wang

References

Wang, Yu, Rinaldo and Willett (2019) <arxiv:1909.06359>

Examples

```
p = 10
sigma = 1
n = 100
A = matrix(rnorm(p*p), nrow = p)*0.1  # transition matrix
simu.VAR1(sigma, p, n, A)
```

`softImpute.network.missing` *Estimate graphon matrix by soft-impute for independent adjacency matrices with missing values.*

Description

Estimate graphon matrix by soft-impute for independent adjacency matrices with missing values.

Usage

```
softImpute.network.missing(
  data_incomplete_list,
  eta_list,
  lambda,
  a,
  it_max = 10000
)
```

Arguments

`data_incomplete_list`	A `list` of adjacency matrices (with entries being 0 or 1) with missing values being coercing to 0.
`eta_list`	A `list` of matrices associated with data_incomplete_list, each matrix indicates the missing entries in corresponding adjacency matrix.
`lambda`	A `numeric` scalar of thresholding parameter for leading singular value in the soft-impute algorithm.
`a`	A `numeric` scalar of truncation parameter in the soft-impute algorithm.
`it_max`	An `integer` scalar of maximum iteration for the soft-impute algorithm.

Value

Estimated graphon matrix

Author(s)

Haotian Xu

References

Dubey, Xu and Yu (2021) <arxiv:2110.06450>

thresholdBS *Thresholding a BS object with threshold value tau.*

Description

Given a BS object, perform thresholding to find the change point locations.

Usage

```
thresholdBS(BS_object, tau)
```

Arguments

BS_object	A BS object.
tau	A positive numeric scalar of thresholding value.

Value

A list with the following structure:

BS_tree_trimmed	BS_tree with change points which do not satisfy the thresholding criteria removed
cpt_hat	A matrix contains change point locations, values of corresponding statistic, and levels at which each change point is detected

Author(s)

Haotian Xu

See Also

BS.univar, BS.uni.nonpar, BS.cov, WBS.univar, WBS.uni.nonpar, WBS.multi.nonpar, WBS.network, WBSIP.cov

Examples

```
y = c(rnorm(100, 0, 1), rnorm(100, 10, 10), rnorm(100, 40, 10))
temp = BS.univar(y, 1, 300, 5)
plot.ts(y)
points(x = tail(temp$S[order(temp$Dval)],4), y = y[tail(temp$S[order(temp$Dval)],4)], col = "red")
thresholdBS(temp, 20)
```

`trim_interval` *Interval trimming based on initial change point localisation.*

Description

Performing the interval trimming for local refinement.

Usage

```
trim_interval(n, cpt_init, w = 0.9)
```

Arguments

`n`	An `integer` scalar corresponding to the sample size.
`cpt_init`	An `integer` vector of initial changepoints estimation (sorted in strictly increasing order).
`w`	A `numeric` scalar in (0,1) representing the weight for interval truncation.

Value

A matrix with each row be a trimmed interval.

Author(s)

Haotian Xu

References

Xu, Wang, Zhao and Yu (2022) <arXiv:2207.12453>.

`tuneBSmultinonpar` *A function to compute change points based on the multivariate nonparametic method with tuning parameter selected by FDR control.*

Description

A function to compute change points based on the multivariate nonparametic method with tuning parameter selected by FDR control.

Usage

```
tuneBSmultinonpar(BS_object, Y)
```

Arguments

BS_object	A "BS" object produced by WBS.multi.nonpar.
Y	A numeric matrix of observations with with horizontal axis being time, and vertical axis being dimension.

Value

A vector of estimated change points.

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Padilla, Yu, Wang and Rinaldo (2019) <arxiv:1910.13289>.

See Also

WBS.multi.nonpar.

Examples

```
n = 70
v = c(floor(n/3), 2*floor(n/3)) # location of change points
p = 4
Y = matrix(0, p, n) # matrix for data
mu0 = rep(0, p) # mean of the data
mu1 = rep(0, p)
mu1[1:floor(p/2)] = 2
Sigma0 = diag(p) #Covariance matrices of the data
Sigma1 = diag(p)*2
# Generate data
for(t in 1:n){
  if(t < v[1] || t > v[2]){
     Y[,t] = MASS::mvrnorm(n = 1, mu0, Sigma0)
  }
  if(t >= v[1] && t < v[2]){
     Y[,t] = MASS::mvrnorm(n = 1, mu1, Sigma1)
  }
}## close for generate data
M = 8
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(Y)) #Random intervals
K_max = 30
h = 5*(K_max*log(n)/n)^{1/p} # bandwith
temp = WBS.multi.nonpar(Y, Y, 1, ncol(Y), intervals$Alpha, intervals$Beta, h, delta = 10)
S = tuneBSmultinonpar(temp, Y)
```

tuneBSnonparRDPG *Change points detection for dependent dynamic random dot product graph models.*

Description

Perform Change points detection for dependent dynamic random dot product graph models. The tuning parameter tau for WBS is automatically selected based on the BIC-type scores defined in Equation (2.4) in Zou et al. (2014).

Usage

```
tuneBSnonparRDPG(BS_object, data_mat, lowerdiag = FALSE, d)
```

Arguments

BS_object	A "BS" object produced by WBS.nonpar.RDPG.
data_mat	A numeric matrix of observations with horizontal axis being time, and vertical axis being vectorized adjacency matrix.
lowerdiag	A logic scalar. TRUE, if each row of data_mat is the vectorization of the strictly lower diagonal elements in an adjacency matrix. FALSE, if each row of data_mat is the vectorization of all elements in an adjacency matrix.
d	A numeric scalar of the number of leading singular values of an adjacency matrix considered in the scaled PCA algorithm.

Value

A numeric vector of estimated change points.

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Padilla, Yu and Priebe (2019) <arxiv:1911.07494>.

See Also

WBS.nonpar.RDPG.

Examples

```
### generate data
p = 20 # number of nodes
n = 50 # sample size for each segment
lat_dim_num = 5 # number of latent dimensions
set.seed(1)
x_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
x_tilde_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
y_mat = rbind(x_tilde_mat[1:floor(p/4),], x_mat[(floor(p/4)+1):p,])
rdpg1 = simu.RDPG(x_mat, n, symm = TRUE, self = FALSE)
rdpg2 = simu.RDPG(y_mat, n, symm = TRUE, self = FALSE)
data1_mat = rdpg1$obs_mat
data2_mat = rdpg2$obs_mat
data_mat = cbind(data1_mat, data2_mat)
### detect change points
M = 20 # number of random intervals for WBS
d = 10 # parameter for scaled PCA algorithm
delta = 5
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(data_mat))
WBS_result = WBS.nonpar.RDPG(data_mat, lowerdiag = TRUE, d,
             Alpha = intervals$Alpha, Beta = intervals$Beta, delta)
cpt_hat = tuneBSnonparRDPG(WBS_result, data_mat, lowerdiag = TRUE, d)
```

`tuneBSuninonpar` *Wild binary segmentation for univariate nonparametric change points detection with tuning parameter selection.*

Description

Perform wild binary segmentation with tuning parameter selection based on sample splitting.

Usage

```
tuneBSuninonpar(BS_object, Y, N)
```

Arguments

`BS_object`	A "BS" object produced by `BS.uni.nonpar` or `WBS.uni.nonpar`.
`Y`	A `numeric` matrix of observations with horizontal axis being time, and vertical axis being multiple observations on each time point.
`N`	A `integer` vector representing number of multiple observations on each time point.

Value

A vector of estimated change points (sorted in strictly increasing order).

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Padilla, Yu, Wang and Rinaldo (2021) <doi:10.1214/21-EJS1809>.

See Also

`BS.uni.nonpar` and `WBS.uni.nonpar`.

Examples

```
Y = t(as.matrix(c(rnorm(100, 0, 1), rnorm(100, 0, 10), rnorm(50, 0, 40))))
W = Y # W is a copy of the matrix Y, it can be Y itself.
N = rep(1, 250)
M = 5
set.seed(123)
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(Y))
BS_object = WBS.uni.nonpar(W, 1, ncol(Y), intervals$Alpha, intervals$Beta, N, delta = 5)
cpt_hat = tuneBSuninonpar(BS_object, Y, N)
```

---

tuneBSunivar *Univariate mean change points detection based on standard or wild binary segmentation with tuning parameter selected by sSIC.*

Description

Perform univariate mean change points detection based on standard or wild binary segmentation. The threshold parameter tau for WBS is automatically selected based on the sSIC score defined in Equation (4) in Fryzlewicz (2014).

Usage

```
tuneBSunivar(BS_object, y)
```

Arguments

`BS_object`	A "BS" object produced by `BS.univar` or `WBS.univar`.
`y`	A `numeric` vector of observations.

Value

A `list` with the following structure:

`cpt`	A vector of estimated change point locations (sorted in strictly increasing order).
`tau`	A scalar of selected threshold tau based on sSIC.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>; Fryzlewicz (2014), Wild binary segmentation for multiple change-point detection, <DOI: 10.1214/14-AOS1245>.

See Also

`BS.univar` and `WBS.univar`.

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
## change points detection by WBS
intervals = WBS.intervals(M = 100, lower = 1, upper = length(y))
temp2 = WBS.univar(y, 1, length(y), intervals$Alpha, intervals$Beta, delta = 5)
WBS_result = tuneBSunivar(temp2, y)
cpt_WBS = WBS_result$cpt
Hausdorff.dist(cpt_WBS, cpt_true)
```

`WBS.intervals` *Generate random intervals for WBS.*

Description

Generate random intervals for WBS.

Usage

```
WBS.intervals(M, lower = 1, upper)
```

Arguments

`M`	A positive `integer` scalar of number of random intervals.
`lower`	A positive `integer` scalar of lower bound of random intervals.
`upper`	A positive `integer` scalar of upper bound of random intervals.

Value

A `list` with the following structure:

`Alpha`	A M-dim vector representing the starting indices of random intervals
`Beta`	A M-dim vector representing the ending indices of random intervals

Author(s)

Oscar Hernan Madrid Padilla

Examples

```
WBS.intervals(120, lower = 1, upper = 300)
```

`WBS.multi.nonpar` *Wild binary segmentation for multivariate nonparametric change points detection.*

Description

Perform wild binary segmentation for multivariate nonparametric change points detection.

Usage

```
WBS.multi.nonpar(Y, W, s, e, Alpha, Beta, h, delta, level = 0)
```

Arguments

`Y`	A `numeric` matrix of observations with with horizontal axis being time, and vertical axis being dimension.
`W`	A copy of the matrix Y, it can be Y itself.
`s`	A `integer` scalar of starting index.
`e`	A `integer` scalar of ending index.
`Alpha`	A `integer` vector of starting indices of random intervals.
`Beta`	A `integer` vector of ending indices of random intervals.
`h`	A `numeric` scalar of bandwidth parameter.
`delta`	A `integer` scalar of minimum spacing.
`level`	Should be fixed as 0.

Value

An object of `class` "BS", which is a `list` with the following structure:

`S`	A vector of estimated change points (sorted in strictly increasing order).
`Dval`	A vector of values of CUSUM statistic based on KS distance.
`Level`	A vector representing the levels at which each change point is detected.
`Parent`	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Oscar Hernan Madrid Padilla & Haotian Xu

References

Padilla, Yu, Wang and Rinaldo (2019) <arxiv:1910.13289>.

See Also

thresholdBS for obtain change points estimation, tuneBSmultinonpar for a tuning version.

Examples

```
n = 70
v = c(floor(n/3), 2*floor(n/3)) # location of change points
p = 4
Y = matrix(0, p, n) # matrix for data
mu0 = rep(0, p) # mean of the data
mu1 = rep(0, p)
mu1[1:floor(p/2)] = 2
Sigma0 = diag(p) #Covariance matrices of the data
Sigma1 = diag(p)*2
# Generate data
for(t in 1:n){
  if(t < v[1] || t > v[2]){
     Y[,t] = MASS::mvrnorm(n = 1, mu0, Sigma0)
  }
  if(t >= v[1] && t < v[2]){
     Y[,t] = MASS::mvrnorm(n = 1, mu1, Sigma1)
  }
}## close for generate data
M = 10
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(Y)) #Random intervals
K_max = 30
h = 5*(K_max*log(n)/n)^{1/p} # bandwith
temp = WBS.multi.nonpar(Y, Y, 1, ncol(Y), intervals$Alpha, intervals$Beta, h, delta = 10)
result = thresholdBS(temp, median(temp$Dval))
```

WBS.network *Wild binary segmentation for network change points detection.*

Description

Perform wild binary segmentation for network change points detection.

Usage

```
WBS.network(data_mat1, data_mat2, s, e, Alpha, Beta, delta, level = 0)
```

Arguments

data_mat1	A numeric matrix of observations with with horizontal axis being time, and with each column be the vectorized adjacency matrix.
data_mat2	An independent copy of data_mat1.
s	A integer scalar of starting index.
e	A integer scalar of ending index.
Alpha	A integer vector of starting indices of random intervals.
Beta	A integer vector of ending indices of random intervals.
delta	A positive integer scalar of minimum spacing.
level	Should be fixed as 0.

Value

An object of class "BS", which is a list with the following structure:

S	A vector of estimated change points (sorted in strictly increasing order).
Dval	A vector of values of CUSUM statistic based on KS distance.
Level	A vector representing the levels at which each change point is detected.
Parent	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Daren Wang & Haotian Xu

References

Wang, Yu and Rinaldo (2018) <arxiv:1809.09602>.

See Also

thresholdBS for obtaining change points estimation.

Examples

```
p = 15 # number of nodes
rho = 0.5 # sparsity parameter
block_num = 3 # number of groups for SBM
n = 100 # sample size for each segment
# connectivity matrix for the first and the third segments
conn1_mat = rho * matrix(c(0.6,1,0.6,1,0.6,0.5,0.6,0.5,0.6), nrow = 3)
# connectivity matrix for the second segment
conn2_mat = rho * matrix(c(0.6,0.5,0.6,0.5,0.6,1,0.6,1,0.6), nrow = 3)
set.seed(1)
can_vec = sample(1:p, replace = FALSE) # randomly assign nodes into groups
sbm1 = simu.SBM(conn1_mat, can_vec, n, symm = TRUE, self = TRUE)
sbm2 = simu.SBM(conn2_mat, can_vec, n, symm = TRUE, self = TRUE)
```

```
data_mat = cbind(sbm1$obs_mat, sbm2$obs_mat)
data_mat1 = data_mat[,seq(1,ncol(data_mat),2)]
data_mat2 = data_mat[,seq(2,ncol(data_mat),2)]
M = 10
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(data_mat1))
temp = WBS.network(data_mat1, data_mat2, 1, ncol(data_mat1),
                   intervals$Alpha, intervals$Beta, delta = 5)
rho_hat = quantile(rowMeans(data_mat), 0.95)
tau = p*rho_hat*(log(n))^2/20 # default threshold given in the paper
cpt_init = unlist(thresholdBS(temp, tau)$cpt_hat[,1])
cpt_refined = local.refine.network(cpt_init, data_mat1, data_mat2,
                      self = TRUE, tau2 = p*rho_hat/3, tau3 = Inf)
cpt_WBS = 2*cpt_init
cpt_refined = 2*cpt_refined
```

`WBS.nonpar.RDPG` *Wild binary segmentation for dependent dynamic random dot product graph models.*

Description

Perform wild binary segmentation for dependent dynamic random dot product graph models.

Usage

```
WBS.nonpar.RDPG(data_mat, lowerdiag = FALSE, d, Alpha, Beta, delta)
```

Arguments

`data_mat`	A `numeric` matrix of observations with horizontal axis being time, and vertical axis being vectorized adjacency matrix.
`lowerdiag`	A `logic` scalar. TRUE, if each row of data_mat is the vectorization of the strictly lower diagonal elements in an adjacency matrix. FALSE, if each row of data_mat is the vectorization of all elements in an adjacency matrix.
`d`	A `numeric` scalar of the number of leading singular values of an adjacency matrix considered in the scaled PCA algorithm.
`Alpha`	A `integer` vector of starting indices of random intervals.
`Beta`	A `integer` vector of ending indices of random intervals.
`delta`	A positive `integer` scalar of minimum spacing.

Value

An object of `class` "BS", which is a `list` with the following structure:

`S`	A vector of estimated change point locations (sorted in strictly increasing order).
`Dval`	A vector of values of CUSUM statistic.
`Level`	A vector representing the levels at which each change point is detected.
`Parent`	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Oscar Hernan Madrid Padilla, Haotian Xu

References

Padilla, Yu and Priebe (2019) <arxiv:1911.07494>.

See Also

`thresholdBS` for obtaining change points estimation, `tuneBSnonparRDPG` for a tuning version.

Examples

```
### generate data
p = 20 # number of nodes
n = 50 # sample size for each segment
lat_dim_num = 5 # number of latent dimensions
set.seed(1)
x_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
x_tilde_mat = matrix(runif(p*lat_dim_num), nrow = p, ncol = lat_dim_num)
y_mat = rbind(x_tilde_mat[1:floor(p/4),], x_mat[(floor(p/4)+1):p,])
rdpg1 = simu.RDPG(x_mat, n, symm = TRUE, self = FALSE)
rdpg2 = simu.RDPG(y_mat, n, symm = TRUE, self = FALSE)
data1_mat = rdpg1$obs_mat
data2_mat = rdpg2$obs_mat
data_mat = cbind(data1_mat, data2_mat)
### detect change points
M = 30 # number of random intervals for WBS
d = 10 # parameter for scaled PCA algorithm
delta = 5
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(data_mat))
WBS_result = WBS.nonpar.RDPG(data_mat, lowerdiag = TRUE, d,
             Alpha = intervals$Alpha, Beta = intervals$Beta, delta)
```

WBS.uni.nonpar *Wild binary segmentation for univariate nonparametric change points detection.*

Description

Perform wild binary segmentation for univariate nonparametric change points detection.

Usage

```
WBS.uni.nonpar(Y, s, e, Alpha, Beta, N, delta, level = 0)
```

Arguments

Y	A numeric matrix of observations with horizontal axis being time, and vertical axis being multiple observations on each time point.
s	A integer scalar of starting index.
e	A integer scalar of ending index.
Alpha	A integer vector of starting indices of random intervals.
Beta	A integer vector of ending indices of random intervals.
N	A integer vector representing number of multiple observations on each time point.
delta	A positive integer scalar of minimum spacing.
level	Should be fixed as 0.

Value

A list with the following structure:

S	A vector of estimated change points (sorted in strictly increasing order)
Dval	A vector of values of CUSUM statistic based on KS distance
Level	A vector representing the levels at which each change point is detected
Parent	A matrix with the starting indices on the first row and the ending indices on the second row

Author(s)

Oscar Hernan Madrid Padilla, Haotian Xu

References

Padilla, Yu, Wang and Rinaldo (2021) <doi:10.1214/21-EJS1809>.

See Also

thresholdBS for obtaining change points estimation, tuneBSuninonpar for a tuning version.

Examples

```
Y = t(as.matrix(c(rnorm(100, 0, 1), rnorm(100, 0, 10), rnorm(100, 0, 40))))
N = rep(1, 300)
M = 20
intervals = WBS.intervals(M = M, lower = 1, upper = ncol(Y))
temp = WBS.uni.nonpar(Y, 1, 300, intervals$Alpha, intervals$Beta, N, 5)
plot.ts(t(Y))
points(x = tail(temp$S[order(temp$Dval)], 4), y = Y[,tail(temp$S[order(temp$Dval)],4)], col = "red")
thresholdBS(temp, 2)
```

`WBS.uni.rob` *Robust wild binary segmentation for univariate mean change points detection.*

Description

Perform a robust version of the wild binary segmentation method using Huber loss.

Usage

```
WBS.uni.rob(y, s, e, Alpha, Beta, K = 1.345, delta, level = 0)
```

Arguments

`y`	A `numeric` vector of observations.
`s`	A `numeric` scalar representing the starting index of an interval.
`e`	A `numeric` scalar representing the ending index of an interval.
`Alpha`	An `integer` vector of starting indices of random intervals.
`Beta`	An `integer` vector of ending indices of random intervals.
`K`	A `numeric` scalar representing the robustification parameter in the Huber loss.
`delta`	A positive `integer` scalar of minimum spacing.
`level`	Should be fixed as 0.

Value

An object of `class` "BS", which is a `list` with the following structure:

`S`	A vector of estimated change points (sorted in strictly increasing order).
`Dval`	A vector of values of CUSUM statistic based on KS distance.
`Level`	A vector representing the levels at which each change point is detected.
`Parent`	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Mengchu Li & Haotian Xu

References

Fearnhead & Rigaill (2019) <doi:10.1080/01621459.2017.1385466>.

See Also

`thresholdBS` for obtaining change points estimation.

WBS.univar *Wild binary segmentation for univariate mean change points detection.*

Description

Perform wild binary segmentation for univariate mean change points detection.

Usage

```
WBS.univar(y, s, e, Alpha, Beta, delta = 2, level = 0)
```

Arguments

y	A numeric vector of observations.
s	A integer scalar of starting index.
e	A integer scalar of ending index.
Alpha	A integer vector of starting indices of random intervals.
Beta	A integer vector of ending indices of random intervals.
delta	A positive integer scalar of minimum spacing.
level	Should be fixed as 0.

Value

An object of class "BS", which is a list with the following structure:

S	A vector of estimated change point locations (sorted in strictly increasing order).
Dval	A vector of values of CUSUM statistic.
Level	A vector representing the levels at which each change point is detected.
Parent	A matrix with the starting indices on the first row and the ending indices on the second row.

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2020) <doi:10.1214/20-EJS1710>.

See Also

thresholdBS for obtaining change points estimation, tuneBSunivar for a tuning version.

Examples

```
set.seed(0)
cpt_true = c(20, 50, 170)
y = rnorm(300) + c(rep(0,20),rep(2,30),rep(0,120),rep(2,130))
intervals = WBS.intervals(M = 300, lower = 1, upper = length(y))
temp = WBS.univar(y, 1, length(y), intervals$Alpha, intervals$Beta, delta = 5)
plot.ts(y)
points(x = tail(temp$S[order(temp$Dval)],4),
       y = y[tail(temp$S[order(temp$Dval)],4)], col = "red")
WBS_result = thresholdBS(temp, tau = 4)
print(WBS_result$BS_tree, "value")
plot(WBS_result$BS_tree)
print(WBS_result$BS_tree_trimmed, "value")
plot(WBS_result$BS_tree_trimmed)
cpt_hat = sort(WBS_result$cpt_hat[,1]) # the threshold tau is specified to be 4
Hausdorff.dist(cpt_hat, cpt_true)
cpt_LR = local.refine.univar(cpt_hat, y)
Hausdorff.dist(cpt_LR, cpt_true)
```

`WBSIP.cov` *Wild binary segmentation for covariance change points detection through Independent Projection.*

Description

Perform wild binary segmentation for covariance change points detection through Independent Projection

Usage

```
WBSIP.cov(X, X_prime, s, e, Alpha, Beta, delta, level = 0)
```

Arguments

`X`	A `numeric` vector of observations.
`X_prime`	A `numeric` vector of observations which are independent copy of X.
`s`	A `integer` scalar of starting index.
`e`	A `integer` scalar of ending index.
`Alpha`	A `integer` vector of starting indices of random intervals.
`Beta`	A `integer` vector of ending indices of random intervals.
`delta`	A positive `integer` scalar of minimum spacing.
`level`	A parameter for tracking the level at which a change point is detected. Should be fixed as 0.

Value

An object of class "BS", which is a list with the following structure:

S	A vector of estimated change points (sorted in strictly increasing order)
Dval	A vector of values of CUSUM statistic based on KS distance
Level	A vector representing the levels at which each change point is detected
Parent	A matrix with the starting indices on the first row and the ending indices on the second row

Author(s)

Haotian Xu

References

Wang, Yu and Rinaldo (2021) <doi:10.3150/20-BEJ1249>.

See Also

thresholdBS for obtain change points estimation.

Examples

```
p = 10
A1 = gen.cov.mat(p, 1, "equal")
A2 = gen.cov.mat(p, 3, "power")
A3 = A1
set.seed(1234)
X = cbind(t(MASS::mvrnorm(50, mu = rep(0, p), A1)),
          t(MASS::mvrnorm(50, mu = rep(0, p), A2)),
          t(MASS::mvrnorm(50, mu = rep(0, p), A3)))
X_prime = cbind(t(MASS::mvrnorm(50, mu = rep(0, p), A1)),
                t(MASS::mvrnorm(50, mu = rep(0, p), A2)),
                t(MASS::mvrnorm(50, mu = rep(0, p), A3)))
intervals = WBS.intervals(M = 120, lower = 1, upper = dim(X)[2])
temp = WBSIP.cov(X, X_prime, 1, dim(X)[2], intervals$Alpha, intervals$Beta, delta = 5)
tau = sqrt(p*log(ncol(X)))*1.5
sort(thresholdBS(temp, tau)$cpt_hat[,1])
```

Index