Package ‘SBCK’

January 20, 2025

Type Package

Title Statistical Bias Correction Kit

Version 1.0.0

Date 2023-09-11

Description Implementation of several recent multivariate bias correction methods with a unified interface to facilitate their use. A description and comparison between methods can be found in <doi:10.5194/esd-11-537-2020>.

URL https://github.com/yrobink/SBCK

Depends R (>= 3.3)

License GPL-3

Encoding UTF-8

NeedsCompilation yes

Imports Rcpp, methods, R6, ROOPSD (>= 0.3.5), transport

LinkingTo Rcpp, RcppEigen

RcppModules SBCK_cpp

RoxygenNote 7.2.3

Author Yoann Robin [aut, cre],
Mathieu Vrac [cph]

Maintainer Yoann Robin <yoann.robin.k@gmail.com>

Repository CRAN

Date/Publication 2023-09-11 11:50:10 UTC

Contents

AR2D2 *AR2D2 (Analogues Rank Resampling for Distributions and Dependences) method*

Description

Perform a multivariate (non stationary) bias correction.

Details

Use Quantiles shuffle in calibration and projection period with CDFt

Public fields

`mvq` [MVQuantilesShuffle] Class to transform dependance structure

`bc_method` [SBCK::] Bias correction method

`bckwargs` [list] List of arguments of bias correction

`bcm_` [SBCK::] Instancied bias correction method

`reverse` [bool] If we apply bc_method first and then shuffle, or reverse

Methods

Public methods:

- `AR2D2$new()`
- `AR2D2$fit()`
- `AR2D2$predict()`
- `AR2D2$clone()`

Method `new()`**:** Create a new AR2D2 object.

Usage:

```
AR2D2$new(
  col_cond = base::c(1),
  lag_search = 1,
  lag_keep = 1,
  bc_method = SBCK::CDFt,
  shuffle = "quantile",
  reverse = FALSE,
  ...
)
```

Arguments:

`col_cond` Conditionning colum

`lag_search` Number of lags to transform the dependence structure

`lag_keep` Number of lags to keep

`bc_method` Bias correction method

`shuffle` Shuffle method used, can be quantile or rank

`reverse` If we apply bc_method first and then shuffle, or reverse

`...` Others named arguments passed to bc_method$new

Returns: A new ‘AR2D2‘ object.

Method `fit()`**:** Fit the bias correction method. If X1 is NULL, the method is considered as stationary

Usage:

```
AR2D2$fit(Y0, X0, X1 = NULL)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
AR2D2$predict(X1 = NULL, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features or NULL] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL (and vice-versa), else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AR2D2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Vrac, M. et S. Thao (2020). “R2 D2 v2.0 : accounting for temporal dependences in multivariate bias correction via analogue rank resampling”. In : Geosci. Model Dev. 13.11, p. 5367-5387. doi :10.5194/gmd-13-5367-2020.

Examples

```
## Three 4-variate random variables
Y0 = matrix( stats::rnorm( n = 1000 ) , ncol = 4 ) ## Biased in calibration period
X0 = matrix( stats::rnorm( n = 1000 ) , ncol = 4 ) / 2 + 3 ## Reference in calibration period
X1 = matrix( stats::rnorm( n = 1000 ) , ncol = 4 ) * 2 + 6 ## Biased in projection period

## Bias correction
cond_col = base::c(2,4)
lag_search = 6
lag_keep = 3
## Step 1 : construction of the class AR2D2
ar2d2 = SBCK::AR2D2$new( cond_col , lag_search , lag_keep )
## Step 2 : Fit the bias correction model
ar2d2$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction
Z = ar2d2$predict(X1,X0)
```

`bin_width_estimator` *bin_width_estimator method*

Description

Lenght of cell to compute an histogram

Usage

```
bin_width_estimator(X, method = "auto")
```

Arguments

`X`	[matrix] A matrix containing data, nrow = n_samples, ncol = n_features
`method`	[string] Method to estimate bin_width, values are "auto", "FD" (Friedman Draconis, robust over outliners) or "Sturges". If "auto" is used and if nrow(X) < 1000, "Sturges" is used, else "FD" is used.

Value

[vector] Lenght of bins

Examples

```
X = base::cbind( stats::rnorm( n = 2000 ) , stats::rexp(2000) )
## Friedman Draconis is used
binw_width = SBCK::bin_width_estimator( X , method = "auto" )
X = stats::rnorm( n = 500 )
## Sturges is used
binw_width = SBCK::bin_width_estimator( X , method = "auto" )
```

CDFt *CDFt method (Cumulative Distribution Function transfer)*

Description

Perform an univariate bias correction of X with respect to Y.

Details

Correction is applied margins by margins.

Public fields

`n_features` [integer] Number of features

`tol` [double] Floatting point tolerance

`distY0` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

`distY1` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

`distX0` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

`distX1` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

Methods

Public methods:

- `CDFt$new()`
- `CDFt$fit()`
- `CDFt$predict()`
- `CDFt$clone()`

Method `new()`**:** Create a new CDFt object.

Usage:

```
CDFt$new(...)
```

Arguments:

`...` Optional arguments are: - distX0, distX1, models in calibration and projection period, see ROOPSD - distY0, distY1, observations in calibration and projection period, see ROOPSD - kwargsX0, kwargsX1, list of arguments for each respective distribution - kwargsY0, kwargsY1, list of arguments for each respective distribution - scale_left_tail [float] Scale applied on the left support (min to median) between calibration and projection period. If NULL (default), it is determined during the fit. If == 1, equivalent to the original algorithm of CDFt. - scale_right_tail [float] Scale applied on the right support (median to max) between calibration and projection period. If NULL (default), it is determined during the fit.

If == 1, equivalent to the original algorithm of CDFt. - normalize_cdf [bool or vector of bool] If a normalization is applied to the data to maximize the overlap of the support. Can be a bool (True or False, applied for all colums), or a list of bool of size 'n_features' to distinguished each columns.

Returns: A new 'CDFt' object.

Method `fit()`**:** Fit the bias correction method

Usage:

`CDFt$fit(Y0, X0, X1)`

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

`CDFt$predict(X1, X0 = NULL)`

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`CDFt$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Michelangeli, P.-A., Vrac, M., and Loukos, H.: Probabilistic downscaling approaches: Application to wind cumulative distribution functions, Geophys. Res. Lett., 36, L11708, https://doi.org/10.1029/2009GL038401, 2009.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bias correction
```

```
## Step 1 : construction of the class CDFt
cdft = SBCK::CDFt$new()
## Step 2 : Fit the bias correction model
cdft$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction, Z is a list containing
## corrections
Z = cdft$predict(X1,X0)
Z$Z0 ## Correction in calibration period
Z$Z1 ## Correction in projection period
```

---

chebyshev *Chebyshev distance*

Description

Compute Chebyshev distance between two dataset or SparseHist X and Y

Usage

```
chebyshev(X, Y)
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)

Value

[float] value of distance

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=2)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
d = SBCK::chebyshev(  X ,   Y )
d = SBCK::chebyshev(muX ,   Y )
d = SBCK::chebyshev(  X , muY )
d = SBCK::chebyshev(muX , muY )
```

`cpp_pairwise_distances_XCall`
cpp_pairwise_distances_XCall

Description

Pairwise distances between X and themselves with a R function (metric). DO NOT USE, use SBCK::pairwise_distances

Usage

```
cpp_pairwise_distances_XCall(X,metric)
```

Arguments

`X`	[Rcpp::NumericMatrix] Matrix
`metric`	[Rcpp::Function] R function

`cpp_pairwise_distances_Xstr`
cpp_pairwise_distances_Xstr

Description

Pairwise distances between X and themselves with a compiled str_metric. DO NOT USE, use SBCK::pairwise_distances

Usage

```
cpp_pairwise_distances_Xstr(X,str_metric)
```

Arguments

`X`	[Rcpp::NumericMatrix] Matrix
`str_metric`	[std::string] c++ string

cpp_pairwise_distances_XYCall

cpp_pairwise_distances_XYCall

Description

Pairwise distances between X and Y with a R function (metric). DO NOT USE, use SBCK::pairwise_distances

Usage

```
cpp_pairwise_distances_XYCall(X,Y,metric)
```

Arguments

X	[Rcpp::NumericMatrix] Matrix
Y	[Rcpp::NumericMatrix] Matrix
metric	[Rcpp::Function] R function

cpp_pairwise_distances_XYstr

cpp_pairwise_distances_XYstr

Description

Pairwise distances between two differents matrix X and Y with a compiled str_metric. DO NOT USE, use SBCK::pairwise_distances

Usage

```
cpp_pairwise_distances_XYstr(X,Y,str_metric)
```

Arguments

X	[Rcpp::NumericMatrix] Matrix
Y	[Rcpp::NumericMatrix] Matrix
str_metric	[std::string] c++ string

dataset_bimodal_reverse_2d *dataset_bimodal_reverse_2d*

Description

Generate a testing dataset from bimodale random bivariate Gaussian distribution

Usage

```
dataset_bimodal_reverse_2d(n_samples)
```

Arguments

n_samples	[integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_bimodal_reverse_2d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

dataset_gaussian_2d *dataset_gaussian_2d*

Description

Generate a testing dataset from random bivariate Gaussian distribution

Usage

```
dataset_gaussian_2d(n_samples)
```

Arguments

n_samples	[integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_gaussian_2d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

---

dataset_gaussian_exp_2d

dataset_gaussian_exp_2d

Description

Generate a testing dataset such that the biased dataset is a distribution of the the form Normal x Exp and the reference of the the form Exp x Normal.

Usage

```
dataset_gaussian_exp_2d(n_samples)
```

Arguments

`n_samples`	[integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_gaussian_exp_2d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

---

dataset_gaussian_exp_mixture_1d

dataset_gaussian_exp_mixture_1d

Description

Generate a univariate testing dataset from a mixture of gaussian and exponential distribution

Usage

```
dataset_gaussian_exp_mixture_1d(n_samples)
```

Arguments

n_samples [integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_gaussian_exp_mixture_1d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

---

dataset_gaussian_L_2d *dataset_gaussian_L_2d*

Description

Generate a testing dataset such that the biased dataset is a normal distribution and reference a mixture a normal with a form in "L"

Usage

```
dataset_gaussian_L_2d(n_samples)
```

Arguments

n_samples [integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_gaussian_L_2d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

`dataset_gaussian_VS_exp_1d` *dataset_gaussian_VS_exp_1d*

Description

Generate a univariate testing dataset such that biased data follow an exponential law whereas reference follow a normal distribution

Usage

```
dataset_gaussian_VS_exp_1d(n_samples)
```

Arguments

`n_samples`	[integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_gaussian_VS_exp_1d(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

`dataset_like_tas_pr` *dataset_like_tas_pr*

Description

Generate a testing dataset similar to temperature and precipitation. The method is the following: - Data from a multivariate normal law (dim = 2) are drawn - The quantile mapping is used to map the last column into the exponential law - Values lower than a fixed quantile are replaced by 0

Usage

```
dataset_like_tas_pr(n_samples)
```

Arguments

`n_samples`	[integer] numbers of samples drawn

Value

[list] a list containing X0, X1 (biased in calibration/projection) and Y0 (reference in calibration)

Examples

```
XY = SBCK::dataset_like_tas_pr(2000)
XY$X0 ## Biased in calibration period
XY$Y0 ## Reference in calibration period
XY$X1 ## Biased in projection period
```

---

data_to_hist *data_to_hist*

Description

Just a function to transform two datasets into SparseHist, if X or Y (or the both) are already a SparseHist, update just the second

Usage

```
data_to_hist(X, Y)
```

Arguments

X	[matrix or SparseHist]
Y	[matrix or SparseHist]

Value

[list(muX,muY)] a list with the two SparseHist

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rexp(2000)  )
Y = base::cbind( stats::rexp(2000)  , stats::rnorm(2000) )

bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four give the same result
SBCK::data_to_hist( X   , Y )
SBCK::data_to_hist( muX , Y )
SBCK::data_to_hist( X   , muY )
SBCK::data_to_hist( muX , muY )
```

`DistHelper` *Dist Helper*

Description

Class used by CDFt and QM to facilitate fit, do not use

Details

Used to parallel work for margins

Public fields

`dist` [ROOPSD distribution] name of class

`law` [ROOPSD distribution] class set

`kwargs` [list] arguments of dist

Methods

Public methods:

- `DistHelper$new()`
- `DistHelper$set_features()`
- `DistHelper$fit()`
- `DistHelper$is_frozen()`
- `DistHelper$is_parametric()`
- `DistHelper$clone()`

Method `new()`**:** Create a new DistHelper object.

Usage:

```
DistHelper$new(dist, kwargs)
```

Arguments:

`dist` [ROOPSD distribution or list] statistical law

`kwargs` [list] arguments passed to dist

Returns: A new ‘DistHelper‘ object.

Method `set_features()`**:** set the number of features

Usage:

```
DistHelper$set_features(n_features)
```

Arguments:

`n_features` [integer] numbers of features

Returns: NULL

Method `fit()`**:** fit the laws

Usage:

`DistHelper$fit(X, i)`

Arguments:

`X` [matrix] dataset to fit

`i` [integer] margins to fit

Returns: NULL

Method `is_frozen()`**:** Test if margins i is frozen

Usage:

`DistHelper$is_frozen(i)`

Arguments:

`i` [integer] margins to fit

Returns: [bool]

Method `is_parametric()`**:** Test if margins i is parametric

Usage:

`DistHelper$is_parametric(i)`

Arguments:

`i` [integer] margins to fit

Returns: [bool]

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`DistHelper$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
##
```

`dOTC` *dOTC (dynamical Optimal Transport Correction) method*

Description

Perform a multivariate (non stationary) bias correction.

Details

Three random variables are needed, Y0, X0 and X1. The dynamic between X0 and X1 is estimated, and applied to Y0 to estimate Y1. Finally, OTC is used between X1 and the Y1 estimated.

Super class

SBCK::OTC -> dOTC

Methods

Public methods:

- dOTC$new()
- dOTC$fit()
- dOTC$predict()
- dOTC$clone()

Method `new()`**:** Create a new dOTC object.

Usage:

```
dOTC$new(
  bin_width = NULL,
  bin_origin = NULL,
  cov_factor = "std",
  ot = SBCK::OTNetworkSimplex$new()
)
```

Arguments:

`bin_width` [vector or NULL] A vector of lengths of the cells discretizing R^numbers of variables. If NULL, it is estimating during the fit

`bin_origin` [vector or NULL] Coordinate of lower corner of one cell. If NULL, c(0,...,0) is used

`cov_factor` [string or matrix] Covariance factor to correct the dynamic transferred between X0 and Y0. For string, available values are "std" and "cholesky"

`ot` [OTSolver] Optimal Transport solver, default is the network simplex

Returns: A new ‘dOTC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
dOTC$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Note: Only the center of the bins associated to the corrected points are returned, but all corrections of the form: » bw = dotc$bin_width / 2 » n = base::prod(base::dim(X1)) » Z1 = dotc$predict(X1) » Z1 = Z1 + t(matrix(stats::runif(n = n min = - bw , max = bw) , ncol = dim(X1)[1])) are equivalent for OTC.

Usage:

```
dOTC$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
dOTC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Robin, Y., Vrac, M., Naveau, P., Yiou, P.: Multivariate stochastic bias corrections with optimal transport, Hydrol. Earth Syst. Sci., 23, 773–786, 2019, https://doi.org/10.5194/hess-23-773-2019

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bin length
bin_width = c(0.2,0.2)

## Bias correction
## Step 1 : construction of the class dOTC
dotc = SBCK::dOTC$new( bin_width )
## Step 2 : Fit the bias correction model
dotc$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction, Z is a list containing
## corrections
Z = dotc$predict(X1,X0)
Z$Z0 ## Correction in calibration period
Z$Z1 ## Correction in projection period
```

`dTSMBC` *dTSMBC (dynamical Time Shifted Multivariate Bias Correction)*

Description

Perform a bias correction of auto-correlation

Details

Correct auto-correlation with a shift approach, taking into account of non stationarity.

Public fields

`shift` [Shift class] Shift class to shift data.

`bc_method` [SBCK::BC_method] Underlying bias correction method.

Active bindings

`method` [character] If inverse is by row or column, see class Shift

`ref` [integer] reference column/row to inverse shift, see class Shift. Default is 0.5 * (lag+1)

Methods

Public methods:

- `dTSMBC$new()`
- `dTSMBC$fit()`
- `dTSMBC$predict()`
- `dTSMBC$clone()`

Method `new()`**:** Create a new dTSMBC object.

Usage:

```
dTSMBC$new(lag, bc_method = dOTC, method = "row", ref = "middle", ...)
```

Arguments:

`lag` [integer] max lag of autocorrelation

`bc_method` [SBCK::BC_METHOD] bias correction method to use after shift of data, default is OTC

`method` [character] If inverse is by row or column, see class Shift

`ref` [integer] reference column/row to inverse shift, see class Shift. Default is 0.5 * (lag+1)

`...` [] All others arguments are passed to bc_method

Returns: A new ‘dTSMBC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
dTSMBC$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
dTSMBC$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
dTSMBC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Robin, Y. and Vrac, M.: Is time a variable like the others in multivariate statistical downscaling and bias correction?, Earth Syst. Dynam. Discuss. [preprint], https://doi.org/10.5194/esd-2021-12, in review, 2021.

Examples

```
## arima model parameters
modelX0 = list( ar = base::c(  0.6 , 0.2 , -0.1 ) )
modelX1 = list( ar = base::c(  0.4 , 0.1 , -0.3 ) )
modelY0 = list( ar = base::c( -0.3 , 0.4 , -0.2 ) )

## arima random generator
rand.genX0 = function(n){ return(stats::rnorm( n , mean = 0.2 , sd = 1   )) }
rand.genX1 = function(n){ return(stats::rnorm( n , mean = 0.8 , sd = 1   )) }
rand.genY0 = function(n){ return(stats::rnorm( n , mean = 0   , sd = 0.7 )) }

## Generate two AR processes
X0 = stats::arima.sim( n = 1000 , model = modelX0 , rand.gen = rand.genX0 )
X1 = stats::arima.sim( n = 1000 , model = modelX1 , rand.gen = rand.genX1 )
Y0 = stats::arima.sim( n = 1000 , model = modelY0 , rand.gen = rand.genY0 )
X0 = as.vector( X0 )
X1 = as.vector( X1 )
Y0 = as.vector( Y0 + 5 )

## And correct it with 30 lags
dtsbc = SBCK::dTSMBC$new( 30 )
dtsbc$fit( Y0 , X0 , X1 )
Z = dtsbc$predict(X1,X0)
```

ECBC *ECBC (Empirical Copula Bias Correction) method*

Description

Perform a multivariate (non stationary) bias correction.

Details

use Schaake shuffle

Super class

`SBCK::CDFt` -> `ECBC`

Methods

Public methods:

- `ECBC$new()`
- `ECBC$fit()`
- `ECBC$predict()`
- `ECBC$clone()`

Method `new()`**:** Create a new ECBC object.

Usage:

```
ECBC$new(...)
```

Arguments:

`...` This class is based to CDFt, and takes the same arguments.

Returns: A new ‘ECBC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
ECBC$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
ECBC$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ECBC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Vrac, M. and P. Friederichs, 2015: Multivariate—Intervariable, Spatial, and Temporal—Bias Correction. J. Climate, 28, 218–237, https://doi.org/10.1175/JCLI-D-14-00059.1

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period


## Bias correction
## Step 1 : construction of the class ECBC
ecbc = SBCK::ECBC$new()
## Step 2 : Fit the bias correction model
ecbc$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction
Z = ecbc$predict(X1,X0)
```

`energy` *Energy distance*

Description

Compute Energy distance between two dataset or SparseHist X and Y

Usage

```
energy(X, Y, p = 2, metric = "euclidean")
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
p	[float] power of energy distance, default is 2.
metric	[str or function] metric for pairwise distance, default is "euclidean", see SBCK::pairwise_distances

Value

[float] value of distance

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=10)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
w2 = SBCK::energy(X,Y)
w2 = SBCK::energy(muX,Y)
w2 = SBCK::energy(X,muY)
w2 = SBCK::energy(muX,muY)
```

---

euclidean *Euclidean distance*

Description

Compute Euclidean distance between two dataset or SparseHist X and Y

Usage

```
euclidean(X, Y)
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)

Value

[float] value of distance

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=2)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
d = SBCK::euclidean(  X ,   Y )
d = SBCK::euclidean(muX ,   Y )
d = SBCK::euclidean(  X , muY )
d = SBCK::euclidean(muX , muY )
```

`IdBC` *IdBC (Identity Bias Correction) method*

Description

Always return X1 / X0 as correction.

Details

Only for comparison.

Methods

Public methods:

- `IdBC$new()`
- `IdBC$fit()`
- `IdBC$predict()`
- `IdBC$clone()`

Method `new()`**:** Create a new IdBC object.

Usage:

`IdBC$new()`

Returns: A new ‘IdBC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

`IdBC$fit(Y0, X0, X1 = NULL)`

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

X1 [matrix: n_samples * n_features] Model in projection, can be NULL for stationary BC method

Returns: NULL

Method `predict()`**:** Predict the correction. Use named keywords to use stationary or non-stationary method.

Usage:

```
IdBC$predict(X1 = NULL, X0 = NULL)
```

Arguments:

X1 [matrix: n_samples * n_features or NULL] Model in projection

X0 [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return X1 and / or X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
IdBC$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period


## Bias correction
## Step 1 : construction of the class IdBC
idbc = SBCK::IdBC$new()
## Step 2 : Fit the bias correction model
idbc$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction
Z = idbc$predict(X1,X0)
## Z$Z0 # == X0
## Z$Z1 # == X1
```

---

manhattan *Manhattan distance*

Description

Compute Manhattan distance between two dataset or SparseHist X and Y

Usage

```
manhattan(X, Y)
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)

Value

[float] value of distance

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=2)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
d = SBCK::manhattan(  X ,   Y )
d = SBCK::manhattan(muX ,   Y )
d = SBCK::manhattan(  X , muY )
d = SBCK::manhattan(muX , muY )
```

MBCn *MBCn (Multivariate Bias Correction)*

Description

Perform a multivariate bias correction.

Details

BC is performed with an alternance of rotation and univariate BC.

Public fields

`n_features` [integer] Numbers of features

`bc` [BC class] Univariate BC method

`metric` [function] distance between two datasets

`iter_slope` [Stopping class criteria] class used to test when stop

`bc_params` [list] Parameters of bc

`ortho_mat` [array] Array of orthogonal matrix

`tips` [array] Array which contains the product of ortho and inverse of next

`lbc` [list] list of BC method used.

Methods

Public methods:

- MBCn$new()
- MBCn$fit()
- MBCn$predict()
- MBCn$clone()

Method `new()`: Create a new MBCn object.

Usage:

```
MBCn$new(
  bc = QDM,
  metric = wasserstein,
  stopping_criteria = SlopeStoppingCriteria,
  stopping_criteria_params = list(minit = 20, maxit = 100, tol = 0.001),
  ...
)
```

Arguments:

`bc` [BC class] Univariate bias correction method

`metric` [function] distance between two datasets

`stopping_criteria` [Stopping class criteria] class use to test when to stop the iterations

`stopping_criteria_params` [list] parameters passed to stopping_criteria class

`...` [] Others arguments passed to bc.

Returns: A new ‘MBCn‘ object.

Method `fit()`: Fit the bias correction method

Usage:

```
MBCn$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`: Predict the correction

Usage:

```
MBCn$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
MBCn$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Cannon, A. J., Sobie, S. R., and Murdock, T. Q.: Bias correction of simulated precipitation by quantile mapping: how well do methods preserve relative changes in quantiles and extremes?, J. Climate, 28, 6938–6959, https://doi.org/10.1175/JCLI-D-14- 00754.1, 2015.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(200)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bias correction
## Step 1 : construction of the class MBCn
mbcn = SBCK::MBCn$new()
## Step 2 : Fit the bias correction model
mbcn$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction, Z is a list containing
## corrections
Z = mbcn$predict(X1,X0)
Z$Z0 ## Correction in calibration period
Z$Z1 ## Correction in projection period
```

---

minkowski *Minkowski distance*

Description

Compute Minkowski distance between two dataset or SparseHist X and Y. If p = 2, it is the Euclidean distance, for p = 1, it is the manhattan distance, if p = Inf, chebyshev distance is called.

Usage

```
minkowski(X, Y, p = 2)
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
p	[float] power of distance

Value

[float] value of distance

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=2)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
d = SBCK::minkowski(  X ,   Y , p = 3 )
d = SBCK::minkowski(muX ,   Y , p = 3 )
d = SBCK::minkowski(  X , muY , p = 3 )
d = SBCK::minkowski(muX , muY , p = 3 )
```

MRec — *MRec (Matrix Recorrelation) method*

Description

Perform a multivariate bias correction with Gaussian assumption.

Details

Only pearson correlations are corrected.

Public fields

`n_features` [integer] Numbers of features

Methods

Public methods:

- `MRec$new()`
- `MRec$fit()`
- `MRec$predict()`
- `MRec$clone()`

Method `new()`**:** Create a new MRec object.

Usage:

`MRec$new(distY = NULL, distX = NULL)`

Arguments:

`distY` [A list of ROOPSD distribution or NULL] Describe the law of each margins. A list permit to use different laws for each margins. Default is empirical.

`distX` [A list of ROOPSD distribution or NULL] Describe the law of each margins. A list permit to use different laws for each margins. Default is empirical.

Returns: A new 'MRec' object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
MRec$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
MRec$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MRec$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Bárdossy, A. and Pegram, G.: Multiscale spatial recorrelation of RCM precipitation to produce unbiased climate change scenarios over large areas and small, Water Resources Research, 48, 9502–, https://doi.org/10.1029/2011WR011524, 2012.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bias correction
## Step 1 : construction of the class MRec
mrec = SBCK::MRec$new()
```

```
## Step 2 : Fit the bias correction model
mrec$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction, Z is a list containing corrections.
Z = mrec$predict(X1,X0) ## X0 is optional, in this case Z0 is NULL
Z$Z0 ## Correction in calibration period
Z$Z1 ## Correction in projection period
```

MVQuantilesShuffle *MVQuantilesShuffle*

Description

Multivariate Schaake shuffle using the quantiles.

Details

Used to reproduce the dependence structure of a dataset to another dataset

Public fields

`col_cond` [vector] Conditionning columns

`col_ucond` [vector] Un-conditionning columns

`lag_search` [integer] Number of lags to transform the dependence structure

`lag_keep` [integer] Number of lags to keep

`n_features` [integer] Number of features (dimensions), internal

`qY` [matrix] Quantile structure fitted, internal

`bsYc` [matrix] Block search fitted, internal

Methods

Public methods:

- `MVQuantilesShuffle$new()`
- `MVQuantilesShuffle$fit()`
- `MVQuantilesShuffle$transform()`
- `MVQuantilesShuffle$clone()`

Method `new()`**:** Create a new MVQuantilesShuffle object.

Usage:

```
MVQuantilesShuffle$new(col_cond = base::c(1), lag_search = 1, lag_keep = 1)
```

Arguments:

`col_cond` Conditionning colum

`lag_search` Number of lags to transform the dependence structure

`lag_keep` Number of lags to keep

Returns: A new ‘MVQuantilesShuffle‘ object.

Method `fit()`**:** Fit method

Usage:

```
MVQuantilesShuffle$fit(Y)
```

Arguments:

`Y` [vector] Dataset to infer the dependance structure

Returns: NULL

Method `transform()`**:** Transform method

Usage:

```
MVQuantilesShuffle$transform(X)
```

Arguments:

`X` [vector] Dataset to match the dependance structure with the Y fitted

Returns: Z The X with the quantiles structure of Y

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MVQuantilesShuffle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Vrac, M. et S. Thao (2020). “R2 D2 v2.0 : accounting for temporal dependences in multivariate bias correction via analogue rank resampling”. In : Geosci. Model Dev. 13.11, p. 5367-5387. doi :10.5194/gmd-13-5367-2020.

Examples

```
## Generate sample
X = matrix( stats::rnorm( n = 100 ) , ncol = 4 )
Y = matrix( stats::rnorm( n = 100 ) , ncol = 4 )

## Fit dependence structure
## Assume that the link beween column 2 and 4 is correct, and change also
## the auto-correlation structure until lag 3 = lag_keep - 1
mvq = MVQuantilesShuffle$new( base::c(2,4) , lag_search = 6 , lag_keep = 4 )
mvq$fit(Y)
Z = mvq$transform(X)
```

MVRanksShuffle *MVRanksShuffle*

Description

Multivariate Schaake shuffle using the ranks.

Details

Used to reproduce the dependence structure of a dataset to another dataset

Public fields

`col_cond` [vector] Conditionning columns

`col_ucond` [vector] Un-conditionning columns

`lag_search` [integer] Number of lags to transform the dependence structure

`lag_keep` [integer] Number of lags to keep

`n_features` [integer] Number of features (dimensions), internal

`qY` [matrix] Ranks structure fitted, internal

`bsYc` [matrix] Block search fitted, internal

Methods

Public methods:

- `MVRanksShuffle$new()`
- `MVRanksShuffle$fit()`
- `MVRanksShuffle$transform()`
- `MVRanksShuffle$clone()`

Method `new()`**:** Create a new MVRanksShuffle object.

Usage:

```
MVRanksShuffle$new(col_cond = base::c(1), lag_search = 1, lag_keep = 1)
```

Arguments:

`col_cond` Conditionning colum

`lag_search` Number of lags to transform the dependence structure

`lag_keep` Number of lags to keep

Returns: A new ‘MVRanksShuffle‘ object.

Method `fit()`**:** Fit method

Usage:

```
MVRanksShuffle$fit(Y)
```

Arguments:

Y [vector] Dataset to infer the dependance structure

Returns: NULL

Method `transform()`**:** Transform method

Usage:

`MVRanksShuffle$transform(X)`

Arguments:

X [vector] Dataset to match the dependance structure with the Y fitted

Returns: Z The X with the quantiles structure of Y

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`MVRanksShuffle$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

References

Vrac, M. et S. Thao (2020). “R2 D2 v2.0 : accounting for temporal dependences in multivariate bias correction via analogue rank resampling”. In : Geosci. Model Dev. 13.11, p. 5367-5387. doi :10.5194/gmd-13-5367-2020.

Examples

```
## Generate sample
X = matrix( stats::rnorm( n = 100 ) , ncol = 4 )
Y = matrix( stats::rnorm( n = 100 ) , ncol = 4 )

## Fit dependence structure
## Assume that the link beween column 2 and 4 is correct, and change also
## the auto-correlation structure until lag 3 = lag_keep - 1
mvr = MVRanksShuffle$new( base::c(2,4) , lag_search = 6 , lag_keep = 4 )
mvr$fit(Y)
Z = mvr$transform(X)
```

OTC *OTC (Optimal Transport Correction) method*

Description

Perform a multivariate bias correction of X0 with respect to Y0.

Details

Joint distribution, i.e. all dependence are corrected.

Public fields

`bin_width` [vector or NULL] A vector of lengths of the cells discretizing R^numbers of variables. If NULL, it is estimating during the fit

`bin_origin` [vector or NULL] Coordinate of lower corner of one cell. If NULL, c(0,...,0) is used

`muX` [SparseHist] Histogram of the data from the model

`muY` [SparseHist] Histogram of the data from the observations

`ot` [OTSolver] Optimal Transport solver, default is the network simplex

`plan` [matrix] The plan computed by the ot solver.

`n_features` [integer] Numbers of features

Methods

Public methods:

- `OTC$new()`
- `OTC$fit()`
- `OTC$predict()`
- `OTC$clone()`

Method `new()`**:** Create a new OTC object.

Usage:

```
OTC$new(bin_width = NULL, bin_origin = NULL, ot = SBCK::OTNetworkSimplex$new())
```

Arguments:

`bin_width` [vector or NULL] A vector of lengths of the cells discretizing R^numbers of variables. If NULL, it is estimating during the fit

`bin_origin` [vector or NULL] Coordinate of lower corner of one cell. If NULL, c(0,...,0) is used

`ot` [OTSolver] Optimal Transport solver, default is the network simplex

Returns: A new ‘OTC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
OTC$fit(Y0, X0)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

Returns: NULL

Method `predict()`**:** Predict the correction

Note: Only the center of the bins associated to the corrected points are returned, but all corrections of the form: » bw = otc$bin_width / 2 » n = base::prod(base::dim(X0)) » Z0 = otc$predict(X0) » Z0 = Z0 + t(matrix(stats::runif(n = n min = - bw , max = bw) , ncol = dim(X0)[1])) are equivalent for OTC.

Usage:

`OTC$predict(X0)`

Arguments:

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix] Return the corrections of X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`OTC$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Robin, Y., Vrac, M., Naveau, P., Yiou, P.: Multivariate stochastic bias corrections with optimal transport, Hydrol. Earth Syst. Sci., 23, 773–786, 2019, https://doi.org/10.5194/hess-23-773-2019

Examples

```
## Two bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period

## Bin length
bin_width = SBCK::bin_width_estimator( list(X0,Y0) )

## Bias correction
## Step 1 : construction of the class OTC
otc = SBCK::OTC$new( bin_width )
## Step 2 : Fit the bias correction model
otc$fit( Y0 , X0 )
## Step 3 : perform the bias correction, Z0 is the correction of
## X0 with respect to the estimation of Y0
Z0 = otc$predict(X0)
```

`OTHist` *Optimal Transport Histogram*

Description

Histogram

Details

Just a generic class which contains two arguments, p (probability) and c (center of bins)

Public fields

`p` [vector] Vector of probability

`c` [matrix] Vector of center of bins, with nrow = n_samples and ncol = n_features

`bin_width` [vector or NULL] A vector of lengths of the cells discretizing R^numbers of variables. If NULL, it is estimating during the fit

`bin_origin` [vector or NULL] Coordinate of lower corner of one cell. If NULL, c(0,...,0) is used

Methods

Public methods:

- `OTHist$new()`
- `OTHist$clone()`

Method `new()`**:** Create a new OTHist object.

Usage:

```
OTHist$new(p, c)
```

Arguments:

`p` [vector] Vector of probability

`c` [matrix] Vector of center of bins, with nrow = n_samples and ncol = n_features

Returns: A new ‘OTHist‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OTHist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Build a random discrete probability distribution
p = stats::rnorm(100)
p = p / base::sum(p)
c = base::seq( -1 , 1 , length = 100 )
mu = OTHist$new( p , c )
```

OTNetworkSimplex *Optimal Transport Network Simplex solver*

Description

Solve the optimal transport problem with the package 'transport'

Details

use the network simplex algorithm

Public fields

`p` [double] Power of the plan

`plan` [matrix] transport plan

`success` [bool] If the fit is a success or not

`C` [matrix] Cost matrix

Methods

Public methods:

- `OTNetworkSimplex$new()`
- `OTNetworkSimplex$fit()`
- `OTNetworkSimplex$clone()`

Method `new()`**:** Create a new OTNetworkSimplex object.

Usage:

```
OTNetworkSimplex$new(p = 2)
```

Arguments:

`p` [double] Power of the plan

Returns: A new 'OTNetworkSimplex' object.

Method `fit()`**:** Fit the OT plan

Usage:

```
OTNetworkSimplex$fit(muX0, muX1, C = NULL)
```

Arguments:

`muX0` [SparseHist or OTHist] Source histogram to move

`muX1` [SparseHist or OTHist] Target histogram

`C` [matrix or NULL] Cost matrix (without power p) between muX0 and muX1, if NULL pairwise_distances is called with Euclidean distance.

Returns: NULL

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`OTNetworkSimplex$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D.: Linear Programming and Network Flows, 4th edn., John Wiley & Sons, 2009.

Examples

```
## Define two dataset
X = stats::rnorm(2000)
Y = stats::rnorm(2000 , mean = 5 )
bw = base::c(0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## Find solution
ot = OTNetworkSimplex$new()
ot$fit( muX , muY )

print( sum(ot$plan) ) ## Must be equal to 1
print( ot$success )   ## If solve is success
print( sqrt(sum(ot$plan * ot$C)) ) ## Cost of plan
```

`pairwise_distances` *Pairwise distances*

Description

Compute the matrix of pairwise distances between a matrix X and a matrix Y

Usage

```
pairwise_distances(X,Y,metric)
```

Arguments

`X`	[matrix] A first matrix (samples in row, features in columns).
`Y`	[matrix] A second matrix (samples in row, features in columns). If Y = NULL, then pairwise distances is computed between X and X
`metric`	[string or callable] The metric used. If metric is a string, then metric is compiled (so faster). Available string are: "euclidean", "sqeuclidean" (Square of Euclidean distance), "logeulidean" (log of the Euclidean distance) and "chebyshev" (max). Callable must be a function taking two vectors and returning a double.

Value

distXY [matrix] Pairwise distances. distXY[i,j] is the distance between X[i,] and Y[j,]

Examples

```
X = matrix( stats::rnorm(200) , ncol = 100 , nrow = 2 )
Y = matrix( stats::rexp(300)  , ncol = 150 , nrow = 2 )

distXY = SBCK::pairwise_distances( X , Y )
```

`PPPDiffRef` *PPPDiffRef*

Description

Apply the diff w.r.t. a ref transformation.

Details

Transform a dataset such that all ‘lower‘ dimensions are replaced by the ‘ref‘ dimension minus the ‘lower‘; and all ‘upper‘ dimensions are replaced by ‘upper‘ minus ‘ref‘.

Super class

`SBCK::PrePostProcessing` -> `PPPDiffRef`

Public fields

`ref` [integer] The reference column

`lower` [vector integer] Dimensions lower than ref

`upper` [vector integer] Dimensions upper than ref

Methods

Public methods:

- `PPPDiffRef$new()`
- `PPPDiffRef$transform()`
- `PPPDiffRef$itransform()`
- `PPPDiffRef$clone()`

Method `new()`**:** Create a new PPPDiffRef object.

Usage:

`PPPDiffRef$new(ref, lower = NULL, upper = NULL, ...)`

Arguments:

`ref` The reference column

`lower` Dimensions lower than ref

`upper` Dimensions upper than ref

`...` Others arguments are passed to PrePostProcessing

Returns: A new ‘PPPDiffRef‘ object.

Method `transform()`**:** Apply the DiffRef transform.

Usage:

```
PPPDiffRef$transform(X)
```

Arguments:

`X` Data to transform

Returns: Xt a transformed matrix

Method `itransform()`**:** Apply the DiffRef inverse transform.

Usage:

```
PPPDiffRef$itransform(Xt)
```

Arguments:

`Xt` Data to transform

Returns: X a transformed matrix

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPDiffRef$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Parameters
size  = 2000
nfeat = 5
sign  = base::sample( base::c(-1,1) , nfeat - 1 , replace = TRUE )

## Build data
X     = matrix( stats::rnorm( n = size ) , ncol = 1 )
for( s in sign )
{
X = base::cbind( X , X[,1] + s * base::abs(matrix( stats::rnorm(n = size) , ncol = 1 )) )
}

## PPP
lower = which( sign ==  1 ) + 1
upper = which( sign == -1 ) + 1
ppp   = SBCK::PPPDiffRef$new( ref = 1 , lower = lower , upper = upper )
Xt    = ppp$transform(X)
Xti   = ppp$itransform(Xt)

print( base::max( base::abs( X - Xti ) ) )
```

PPPFunctionLink *PPPFunctionLink*

Description

Base class to build link function pre-post processing class. See also the PrePostProcessing documentation

Details

This class is used to define pre/post processing class with a link function and its inverse. See example.

Super class

`SBCK::PrePostProcessing` -> `PPPFunctionLink`

Methods

Public methods:

- `PPPFunctionLink$new()`
- `PPPFunctionLink$transform()`
- `PPPFunctionLink$itransform()`
- `PPPFunctionLink$clone()`

Method `new()`**:** Create a new PPPFunctionLink object.

Usage:

```
PPPFunctionLink$new(transform_, itransform_, cols = NULL, ...)
```

Arguments:

`transform_` The transform function

`itransform_` The inverse transform function

`cols` Columns to apply the link function

`...` Others arguments are passed to PrePostProcessing

Returns: A new 'PPPFunctionLink' object.

Method `transform()`**:** Apply the transform.

Usage:

```
PPPFunctionLink$transform(X)
```

Arguments:

`X` Data to transform

Returns: Xt a transformed matrix

Method `itransform()`**:** Apply the inverse transform.

Usage:

```
PPPFunctionLink$itransform(Xt)
```

Arguments:

`Xt` Data to transform

Returns: X a transformed matrix

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPFunctionLink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Start with data
XY = SBCK::dataset_like_tas_pr(2000)
X0 = XY$X0
X1 = XY$X1
Y0 = XY$Y0

## Define the link function
transform  = function(x) { return(x^3) }
itransform = function(x) { return(x^(1/3)) }

## And the PPP method
ppp = PPPFunctionLink$new( bc_method = CDFt , transform = transform ,
                                             itransform = itransform )

## And now the correction
## Bias correction
ppp$fit(Y0,X0,X1)
Z = ppp$predict(X1,X0)
```

`PPPLogLinLink` *PPPLogLinLink*

Description

Log linear link function. See also the PrePostProcessing documentation.

Details

Log linear link function. The transform is log(x) if 0 < x < 1, else x -1, and the inverse transform exp(x) if x < 0, else x + 1.

Super classes

SBCK::PrePostProcessing -> SBCK::PPPFunctionLink -> PPPLogLinLink

Methods

Public methods:

- PPPLogLinLink$new()
- PPPLogLinLink$clone()

Method `new()`**:** Create a new PPPLogLinLink object.

Usage:

```
PPPLogLinLink$new(s = 1e-05, cols = NULL, ...)
```

Arguments:

`s` The value where the function jump from exp to linear

`cols` Columns to apply the link function

`...` Others arguments are passed to PrePostProcessing

Returns: A new 'PPPLogLinLink' object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPLogLinLink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Start with data
XY = SBCK::dataset_like_tas_pr(2000)
X0 = XY$X0
X1 = XY$X1
Y0 = XY$Y0

## Define the PPP method
ppp = PPPLogLinLink$new( bc_method = CDFt , cols = 2 ,
                         pipe = list(PPPSSR),
                         pipe_kwargs = list(list(cols=2)) )

## And now the correction
## Bias correction
ppp$fit(Y0,X0,X1)
Z = ppp$predict(X1,X0)
```

PPPPreserveOrder *PPPPreserveOrder*

Description

Set an order between cols, and preserve it by swapping values after the correction

Details

Set an order between cols, and preserve it by swapping values after the correction

Super class

`SBCK::PrePostProcessing` -> `PPPPreserveOrder`

Methods

Public methods:

- `PPPPreserveOrder$new()`
- `PPPPreserveOrder$transform()`
- `PPPPreserveOrder$itransform()`
- `PPPPreserveOrder$clone()`

Method `new()`**:** Create a new PPPPreserveOrder object.

Usage:

```
PPPPreserveOrder$new(cols = NULL, ...)
```

Arguments:

`cols` The columns to keep the order

`...` Others arguments are passed to PrePostProcessing

Returns: A new 'PPPPreserveOrder' object.

Method `transform()`**:** nothing occur here

Usage:

```
PPPPreserveOrder$transform(X)
```

Arguments:

`X` Data to transform

Returns: Xt a transformed matrix

Method `itransform()`**:** sort along cols

Usage:

```
PPPPreserveOrder$itransform(Xt)
```

Arguments:

`Xt` Data to transform

Returns: X a transformed matrix

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPPreserveOrder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Build data
X = matrix( stats::rnorm( n = 20 ) , ncol = 2 )

## PPP
ppp   = SBCK::PPPPreserveOrder$new( cols = base::c(1,2) )
Xt    = ppp$transform(X) ## Nothing
Xti   = ppp$itransform(Xt) ## Order
```

`PPPSquareLink` *PPPSquareLink*

Description

Square link function. See also the PrePostProcessing documentation.

Details

Square link function. The transform is x^2, and the sign(x)*sqrt(abs(x)) its inverse.

Super classes

`SBCK::PrePostProcessing` -> `SBCK::PPPFunctionLink` -> `PPPSquareLink`

Methods

Public methods:

- `PPPSquareLink$new()`
- `PPPSquareLink$clone()`

Method `new()`**:** Create a new PPPSquareLink object.

Usage:

```
PPPSquareLink$new(cols = NULL, ...)
```

Arguments:

`cols` Columns to apply the link function

`...` Others arguments are passed to PrePostProcessing

Returns: A new ‘PPPSquareLink‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPSquareLink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Start with data
XY = SBCK::dataset_like_tas_pr(2000)
X0 = XY$X0
X1 = XY$X1
Y0 = XY$Y0

## Define the PPP method
ppp = PPPSquareLink$new( bc_method = CDFt , cols = 2 )

## And now the correction
## Bias correction
ppp$fit(Y0,X0,X1)
Z = ppp$predict(X1,X0)
```

`PPPSSR` *PPPSSR*

Description

Apply the SSR transformation.

Details

Apply the SSR transformation. The SSR transformation replace the 0 by a random values between 0 and the minimal non zero value (the threshold). The inverse transform replace all values lower than the threshold by 0. The threshold used for inverse transform is given by the keyword ‘isaved‘, which takes the value ‘Y0‘ (reference in calibration period), or ‘X0‘ (biased in calibration period), or ‘X1‘ (biased in projection period)

Super class

`SBCK::PrePostProcessing` -> `PPPSSR`

Public fields

`Xn` [vector] Threshold

Methods

Public methods:

- `PPPSSR$new()`
- `PPPSSR$transform()`
- `PPPSSR$itransform()`
- `PPPSSR$clone()`

Method `new()`**:** Create a new PPPSSR object.

Usage:

```
PPPSSR$new(cols = NULL, isaved = "Y0", ...)
```

Arguments:

`cols` Columns to apply the SSR

`isaved` Choose the threshold used for the inverse transform. Can be "Y0", "X0" and "X1".

`...` Others arguments are passed to PrePostProcessing

Returns: A new ‘PPPSSR‘ object.

Method `transform()`**:** Apply the SSR transform, i.e. all 0 are replaced by random values between 0 (excluded) and the minimal non zero value.

Usage:

```
PPPSSR$transform(X)
```

Arguments:

`X` Data to transform

Returns: Xt a transformed matrix

Method `itransform()`**:** Apply the inverse SSR transform, i.e. all values lower than the threshold found in the transform function are replaced by 0.

Usage:

```
PPPSSR$itransform(Xt)
```

Arguments:

`Xt` Data to transform

Returns: X a transformed matrix

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PPPSSR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Start with data
XY = SBCK::dataset_like_tas_pr(2000)
X0 = XY$X0
X1 = XY$X1
Y0 = XY$Y0

## Define the PPP method
ppp = PPPSSR$new( bc_method = CDFt , cols = 2 )

## And now the correction
## Bias correction
ppp$fit(Y0,X0,X1)
Z = ppp$predict(X1,X0)
```

PrePostProcessing *PrePostProcessing base class*

Description

Base class to pre/post process data before/after a bias correction

Details

This base class can be considered as the identity pre-post processing, and is used to be herited by others pre/post processing class. The key ideas are:
- A PrePostProcessing based class contains a bias correction method, initalized by the ‘bc_method‘ argument, always available for all herited class
- The ‘pipe‘ keyword is a list of pre/post processing class, applied one after the other.

Try with an example, start with a dataset similar to tas/pr:
»> XY = SBCK::dataset_like_tas_pr(2000)
»> X0 = XY$X0
»> X1 = XY$X1
»> Y0 = XY$Y0

The first column is Gaussian, but the second is an exponential law with a Dirac mass at 0, represented the 0 of precipitations. For a quantile mapping correction in the calibration period, we just apply
»> qm = SBCK::QM$new()
»> qm$fit(Y0,X0)
»> Z0 = qm$predict(X0)

Now, if we want to pre-post process with the SSR method (0 are replaced by random values between 0 (excluded) and the minimal non zero value), we write:
»> ppp = SBCK::PPPSSR$new(bc_method = QM , cols = 2)

»> ppp$fit(Y0,X0)
»> Z0 = ppp$predict(X0)

The SSR approach is applied only on the second column (the precipitation), and the syntax is the same than for a simple bias correction method.

Imagine now that we want to apply the SSR, and to ensure the positivity of CDFt for precipitation, we also want to use the LogLinLink pre-post processing method. This can be done with the following syntax:
»> ppp = PPPLogLinLink$new(bc_method = CDFt , cols = 2 ,
»> pipe = list(PPPSSR) ,
»> pipe_kwargs = list(list(cols = 2)))
»> ppp$fit(Y0,X0,X1)
»> Z = ppp$predict(X1,X0)

With this syntax, the pre processing operation is PPPLogLinLink$transform(PPPSSR$transform(data)) and post processing operation PPPSSR$itransform(PPPLogLinLink$itransform(bc_data)). So the formula can read from right to left (as the mathematical composition). Note it is equivalent to define:
»> ppp = PrePostProcessing$new(bc_method = CDFt,
»> pipe = list(PPPLogLinLink,PPPSSR),
»> pipe_kwargs = list(list(cols=2) , list(cols=2)))

Methods

Public methods:

- `PrePostProcessing$new()`
- `PrePostProcessing$transform()`
- `PrePostProcessing$itransform()`
- `PrePostProcessing$fit()`
- `PrePostProcessing$predict()`
- `PrePostProcessing$clone()`

Method `new()`**:** Create a new PrePostProcessing object.

Usage:

```
PrePostProcessing$new(
  bc_method = NULL,
  bc_method_kwargs = list(),
  pipe = list(),
  pipe_kwargs = list()
)
```

Arguments:

`bc_method` The bias correction method

`bc_method_kwargs` Dict of keyword arguments passed to bc_method

`pipe` list of others PrePostProcessing class to pipe

`pipe_kwargs` list of list of keyword arguments passed to each elements of pipe

Returns: A new ‘PrePostProcessing‘ object.

Method `transform()`**:** Transformation applied to data before the bias correction. Just the identity for this class

Usage:

```
PrePostProcessing$transform(X)
```

Arguments:

`X` [matrix: n_samples * n_features]

Returns: Xt [matrix: n_samples * n_features]

Method `itransform()`**:** Transformation applied to data after the bias correction. Just the identity for this class

Usage:

```
PrePostProcessing$itransform(Xt)
```

Arguments:

`Xt` [matrix: n_samples * n_features]

Returns: X [matrix: n_samples * n_features]

Method `fit()`**:** Apply the pre processing and fit the bias correction method. If X1 is NULL, the method is considered as stationary

Usage:

```
PrePostProcessing$fit(Y0, X0, X1 = NULL)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction, apply pre-processing before, and post-processing after

Usage:

```
PrePostProcessing$predict(X1 = NULL, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features or NULL] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL (and vice-versa), else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PrePostProcessing$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Start with data
XY = SBCK::dataset_like_tas_pr(2000)
X0 = XY$X0
X1 = XY$X1
Y0 = XY$Y0

## Define pre/post processing method
ppp = PrePostProcessing$new( bc_method = CDFt,
                             pipe = list(PPPLogLinLink,PPPSSR),
                             pipe_kwargs = list( list(cols=2) , list(cols=2) ) )

## Bias correction
ppp$fit(Y0,X0,X1)
Z = ppp$predict(X1,X0)
```

QDM *QDM (Quantile delta mapping method)*

Description

Perform a bias correction.

Details

Mix of delta and quantile method

Methods

Public methods:

- `QDM$new()`
- `QDM$fit()`
- `QDM$predict()`
- `QDM$clone()`

Method `new()`**:** Create a new QDM object.

Usage:

```
QDM$new(delta = "additive", ...)
```

Arguments:

`delta` [character or list] If character : "additive" or "multiplicative". If a list is given, delta[[1]] is the delta transform operator, and delta[[2]] its inverse.

`...` [] Named arguments passed to quantile mapping

Returns: A new ‘QDM‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

`QDM$fit(Y0, X0, X1)`

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

`QDM$predict(X1, X0 = NULL)`

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`QDM$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Cannon, A. J., Sobie, S. R., and Murdock, T. Q.: Bias correction of simulated precipitation by quantile mapping: how well do methods preserve relative changes in quantiles and extremes?, J. Climate, 28, 6938–6959, https://doi.org/10.1175/JCLI-D-14- 00754.1, 2015.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bias correction
## Step 1 : construction of the class QDM
qdm = SBCK::QDM$new()
## Step 2 : Fit the bias correction model
qdm$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction, Z is a list containing
## corrections
Z = qdm$predict(X1,X0)
Z$Z0 ## Correction in calibration period
Z$Z1 ## Correction in projection period
```

QM *Quantile Mapping method*

Description

Perform an univariate bias correction of X0 with respect to Y0

Details

Correction is applied margins by margins.

Public fields

`distX0` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

`distY0` [ROOPSD distribution or a list of them] Describe the law of each margins. A list permit to use different laws for each margins. Default is ROOPSD::rv_histogram.

`n_features` [integer] Numbers of features

`tol` [double] Floatting point tolerance

Methods

Public methods:

- `QM$new()`
- `QM$fit()`
- `QM$predict()`
- `QM$clone()`

Method `new()`**:** Create a new QM object.

Usage:

```
QM$new(distX0 = ROOPSD::rv_histogram, distY0 = ROOPSD::rv_histogram, ...)
```

Arguments:

`distX0` [ROOPSD distribution or a list of them] Describe the law of model

`distY0` [ROOPSD distribution or a list of them] Describe the law of observations

`...` [] kwargsX0 or kwargsY0, arguments passed to distX0 and distY0

Returns: A new ‘QM‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
QM$fit(Y0 = NULL, X0 = NULL)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
QM$predict(X0)
```

Arguments:

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix] Return the corrections of X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
QM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Panofsky, H. A. and Brier, G. W.: Some applications of statistics to meteorology, Mineral Industries Extension Services, College of Mineral Industries, Pennsylvania State University, 103 pp., 1958.

Wood, A. W., Leung, L. R., Sridhar, V., and Lettenmaier, D. P.: Hydrologic Implications of Dynamical and Statistical Approaches to Downscaling Climate Model Outputs, Clim. Change, 62, 189–216, https://doi.org/10.1023/B:CLIM.0000013685.99609.9e, 2004.

Déqué, M.: Frequency of precipitation and temperature extremes over France in an anthropogenic scenario: Model results and statistical correction according to observed values, Global Planet. Change, 57, 16–26, https://doi.org/10.1016/j.gloplacha.2006.11.030, 2007.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period

## Bias correction
## Step 1 : construction of the class QM
qm = SBCK::QM$new()
## Step 2 : Fit the bias correction model
qm$fit( Y0 , X0 )
## Step 3 : perform the bias correction, Z0 is the correction of
## X0 with respect to the estimation of Y0
Z0 = qm$predict(X0)

# ## But in fact the laws are known, we can fit parameters:
distY0 = list( ROOPSD::Exponential , ROOPSD::Normal )
distX0 = list( ROOPSD::Normal , ROOPSD::Exponential )
qm_fix = SBCK::QM$new( distY0 = distY0 , distX0 = distX0 )
qm_fix$fit( Y0 , X0 )
Z0 = qm_fix$predict(X0)
```

QMrs *Quantile Mapping RankShuffle method*

Description

Perform a multivariate bias correction of X with respect to Y

Details

Dependence is corrected with multi_schaake_shuffle.

Super class

`SBCK::QM` -> `QMrs`

Public fields

`irefs` [vector of int] Indexes for shuffle. Defaults is base::c(1)

Methods

Public methods:

- `QMrs$new()`
- `QMrs$fit()`
- `QMrs$predict()`
- `QMrs$clone()`

Method `new()`**:** Create a new QMrs object.

Usage:

```
QMrs$new(irefs = base::c(1), ...)
```

Arguments:

`irefs` [vector of int] Indexes for shuffle. Defaults is base::c(1) model

`...` [] all others arguments are passed to QM class.

Returns: A new ‘QMrs‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
QMrs$fit(Y0, X0)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

`QMrs$predict(X0)`

Arguments:

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix] Return the corrections of X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`QMrs$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Vrac, M.: Multivariate bias adjustment of high-dimensional climate simulations: the Rank Resampling for Distributions and Dependences (R2 D2) bias correction, Hydrol. Earth Syst. Sci., 22, 3175–3196, https://doi.org/10.5194/hess-22-3175-2018, 2018.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period

## Bias correction
## Step 1 : construction of the class QMrs
qmrs = SBCK::QMrs$new()
## Step 2 : Fit the bias correction model
qmrs$fit( Y0 , X0 )
## Step 3 : perform the bias correction
Z0 = qmrs$predict(X0)
```

`R2D2` *R2D2 (Rank Resampling for Distributions and Dependences) method*

Description

Perform a multivariate (non stationary) bias correction.

Details

Use rankshuffle in calibration and projection period with CDFt

Super class

SBCK::CDFt -> R2D2

Public fields

`irefs` [vector of int] Indexes for shuffle. Defaults is base::c(1)

Methods

Public methods:

- `R2D2$new()`
- `R2D2$fit()`
- `R2D2$predict()`
- `R2D2$clone()`

Method `new()`**:** Create a new R2D2 object.

Usage:

```
R2D2$new(irefs = base::c(1), ...)
```

Arguments:

`irefs` [vector of int] Indexes for shuffle. Defaults is base::c(1) model

`...` [] all others arguments are passed to CDFt class.

Returns: A new ‘R2D2‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
R2D2$fit(Y0, X0, X1)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
R2D2$predict(X1, X0 = NULL)
```

Arguments:

`X1` [matrix: n_samples * n_features] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
R2D2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Vrac, M.: Multivariate bias adjustment of high-dimensional climate simulations: the Rank Resampling for Distributions and Dependences (R2 D2) bias correction, Hydrol. Earth Syst. Sci., 22, 3175–3196, https://doi.org/10.5194/hess-22-3175-2018, 2018.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period

## Bias correction
## Step 1 : construction of the class R2D2
r2d2 = SBCK::R2D2$new()
## Step 2 : Fit the bias correction model
r2d2$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction
Z = r2d2$predict(X1,X0)
```

RBC — *RBC (Random Bias Correction) method*

Description

Perform a multivariate bias correction of X with respect to Y randomly.

Details

Only for comparison.

Methods

Public methods:

- `RBC$new()`
- `RBC$fit()`
- `RBC$predict()`
- `RBC$clone()`

Method `new()`**:** Create a new RBC object.

Usage:

```
RBC$new()
```

Returns: A new ‘RBC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

`RBC$fit(Y0, X0, X1 = NULL)`

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

`X1` [matrix: n_samples * n_features] Model in projection, can be NULL for stationary BC method

Returns: NULL

Method `predict()`**:** Predict the correction. Use named keywords to use stationary or non-stationary method.

Usage:

`RBC$predict(X1 = NULL, X0 = NULL)`

Arguments:

`X1` [matrix: n_samples * n_features or NULL] Model in projection

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix or list] Return the matrix of correction of X1 if X0 is NULL, else return a list containing Z1 and Z0, the corrections of X1 and X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`RBC$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Three bivariate random variables (rnorm and rexp are inverted between ref
## and bias)
XY = SBCK::dataset_gaussian_exp_2d(2000)
X0 = XY$X0 ## Biased in calibration period
Y0 = XY$Y0 ## Reference in calibration period
X1 = XY$X1 ## Biased in projection period


## Bias correction
## Step 1 : construction of the class RBC
rbc = SBCK::RBC$new()
## Step 2 : Fit the bias correction model
rbc$fit( Y0 , X0 , X1 )
## Step 3 : perform the bias correction
Z = rbc$predict(X1,X0)
## Z$Z0 # BC of X0
## Z$Z1 # BC of X1
```

`SBCK` *SBCK*

Description

Statistical Bias Correction Kit

Author(s)

Yoann Robin Maintainer: Yoann Robin <yoann.robin.k@gmail.com>

`SchaakeShuffle` *ShaakeShuffle class*

Description

Perform the Schaake Shuffle

Details

as fit/predict mode

Methods

Public methods:

- `SchaakeShuffle$new()`
- `SchaakeShuffle$fit()`
- `SchaakeShuffle$predict()`
- `SchaakeShuffle$clone()`

Method `new()`**:** Create a new ShaakeShuffle object.

Usage:

```
SchaakeShuffle$new(Y0 = NULL)
```

Arguments:

`Y0` [vector] The reference vector

Returns: A new ‘ShaaleShuffle‘ object.

Method `fit()`**:** Fit the model

Usage:

```
SchaakeShuffle$fit(Y0)
```

Arguments:

`Y0` [vector] The reference vector

Returns: NULL

Method `predict()`**:** Fit the model

Usage:

`SchaakeShuffle$predict(X0)`

Arguments:

`X0` [vector] The vector to apply shuffle

Returns: Z0 [vector] data shuffled

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`SchaakeShuffle$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
X0 = matrix( stats::runif(20) , ncol = 2 )
Y0 = matrix( stats::runif(20) , ncol = 2 )
ss = SchaakeShuffle$new()
ss$fit(Y0)
Z0 = ss$predict(X0)
```

`SchaakeShuffleMultiRef` *ShaakeShuffleMultiRef class*

Description

Match the rank structure of X with them of Y by reordering X.

Details

Can keep multiple features to keep the structure of X.

Public fields

`cond_cols` [vector of integer] The conditioning columns

`lag_search` [integer] Number of lag to take into account

`lag_keep` [integer] Number of lag to keep

`Y0` [matrix] Reference data

Methods

Public methods:

- SchaakeShuffleMultiRef$new()
- SchaakeShuffleMultiRef$fit()
- SchaakeShuffleMultiRef$predict()
- SchaakeShuffleMultiRef$clone()

Method `new()`**:** Create a new ShaakeShuffleMultiRef object.

Usage:

```
SchaakeShuffleMultiRef$new(lag_search, lag_keep, cond_cols = base::c(1))
```

Arguments:

`lag_search` [integer] Number of lag to take into account

`lag_keep` [integer] Number of lag to keep

`cond_cols` [vector of integer] The conditioning columns

Returns: A new ‘ShaaleShuffleMultiRef‘ object.

Method `fit()`**:** Fit the model

Usage:

```
SchaakeShuffleMultiRef$fit(Y0)
```

Arguments:

`Y0` [vector] The reference vector

Returns: NULL

Method `predict()`**:** Fit the model

Usage:

```
SchaakeShuffleMultiRef$predict(X0)
```

Arguments:

`X0` [vector] The vector to apply shuffle

Returns: Z0 [vector] data shuffled

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SchaakeShuffleMultiRef$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
X0 = matrix( stats::runif(50) , ncol = 2 )
Y0 = matrix( stats::runif(50) , ncol = 2 )
ssmr = SchaakeShuffleMultiRef$new( lag_search = 3 , lag_keep = 1 , cond_cols = 1 )
ssmr$fit(Y0)
Z0 = ssmr$predict(X0)
```

`SchaakeShuffleRef` *ShaakeShuffleRef class*

Description

Match the rank structure of X with them of Y by reordering X.

Details

Fix one features to keep the structure of X.

Super class

`SBCK::SchaakeShuffle` -> `SchaakeShuffleRef`

Public fields

`ref` [integer] Reference

Methods

Public methods:

- `SchaakeShuffleRef$new()`
- `SchaakeShuffleRef$fit()`
- `SchaakeShuffleRef$predict()`
- `SchaakeShuffleRef$clone()`

Method `new()`**:** Create a new ShaakeShuffleRef object.

Usage:

```
SchaakeShuffleRef$new(ref, Y0 = NULL)
```

Arguments:

`ref` [integer] Reference

`Y0` [vector] The reference vector

Returns: A new ‘ShaaleShuffleRef‘ object.

Method `fit()`**:** Fit the model

Usage:

```
SchaakeShuffleRef$fit(Y0)
```

Arguments:

`Y0` [vector] The reference vector

Returns: NULL

Method `predict()`**:** Fit the model

Usage:

```
SchaakeShuffleRef$predict(X0)
```

Arguments:

`X0` [vector] The vector to apply shuffle

Returns: Z0 [vector] data shuffled

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SchaakeShuffleRef$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
X0 = matrix( stats::runif(20) , ncol = 2 )
Y0 = matrix( stats::runif(20) , ncol = 2 )
ss = SchaakeShuffleRef$new( ref = 1 )
ss$fit(Y0)
Z0 = ss$predict(X0)
```

`schaake_shuffle` *schaake_shuffle function*

Description

Apply the Schaake shuffle to transform the rank of X0 such that its correspond to the rank of Y0

Usage

```
schaake_shuffle(Y0,X0)
```

Arguments

`Y0`	[vector] The reference vector
`X0`	[vector] The vector to transform the rank

Value

Z0 [vector] X shuffled.

Examples

```
X0 = stats::runif(10)
Y0 = stats::runif(10)
Z0 = SBCK::schaake_shuffle( Y0 , X0 )
```

Shift *Shift*

Description

Class to shift a dataset.

Format

R6Class object.

Details

Transform autocorrelations to intervariables correlations

Value

Object of R6Class

Methods

`new(lag,method,ref,)` This method is used to create object of this class with `Shift`

`transform(X)` Method to shift a dataset

`inverse(Xs)` Method to inverse the shift of a dataset

Public fields

`lag` [integer] max lag for autocorrelations

Active bindings

`method` [character] If inverse is by row or column.

`ref` [integer] reference column/row to inverse shift.

Methods

Public methods:

- `Shift$new()`
- `Shift$transform()`
- `Shift$inverse()`
- `Shift$clone()`

Method `new()`: Create a new Shift object.

Usage:

```
Shift$new(lag, method = "row", ref = 1)
```

Arguments:

`lag` [integer] max lag for autocorrelations

`method` [character] If "row" inverse by row, else by column

`ref` [integer] starting point for inverse transform

Returns: A new ‘Shift‘ object.

Method `transform()`**:** Shift the data

Usage:

```
Shift$transform(X)
```

Arguments:

`X` [matrix: n_samples * n_features] Data to shift

Returns: [matrix] Matrix shifted

Method `inverse()`**:** Inverse the shift of the data

Usage:

```
Shift$inverse(Xs)
```

Arguments:

`Xs` [matrix] Data Shifted

Returns: [matrix] Matrix un shifted

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Shift$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
X = base::t(matrix( 1:20 , nrow = 2 , ncol = 10 ))

sh = Shift$new(1)
Xs = sh$transform(X)
Xi = sh$inverse(Xs)
```

`SlopeStoppingCriteria` *Slope stopping criteria*

Description

Class which send a stop signal when a time series stay constant.

Details

Test the slope.

Public fields

`minit` [integer] Minimal number of iterations. At least 3.

`maxit` [integer] Maximal number of iterations.

`nit` [integer] Number of iterations.

`tol` [float] Tolerance to control if slope is close to zero

`stop` [bool] If we stop

`criteria` [vector] State of criteria

`slope` [vector] Values of slope

Methods

Public methods:

- `SlopeStoppingCriteria$new()`
- `SlopeStoppingCriteria$reset()`
- `SlopeStoppingCriteria$append()`
- `SlopeStoppingCriteria$clone()`

Method `new()`**:** Create a new SlopeStoppingCriteria object.

Usage:

```
SlopeStoppingCriteria$new(minit, maxit, tol)
```

Arguments:

`minit` [integer] Minimal number of iterations. At least 3.

`maxit` [integer] Maximal number of iterations.

`tol` [float] Tolerance to control if slope is close to zero

Returns: A new 'SlopeStoppingCriteria' object.

Method `reset()`**:** Reset the class

Usage:

```
SlopeStoppingCriteria$reset()
```

Returns: NULL

Method `append()`**:** Add a new value

Usage:

```
SlopeStoppingCriteria$append(value)
```

Arguments:

`value` [double] New metrics

Returns: NULL

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SlopeStoppingCriteria$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
stop_slope = SlopeStoppingCriteria$new( 20 , 500 , 1e-3 )
x = 0
while(!stop_slope$stop)
{
stop_slope$append(base::exp(-x))
x = x + 0.1
}
print(stop_slope$nit)
```

SparseHist *SparseHist*

Description

Return the Rcpp Class SparseHistBase initialized

Usage

```
SparseHist(X, bin_width = NULL, bin_origin = NULL)
```

Arguments

X	[matrix] Dataset to find the SparseHist
bin_width	[vector] Width of a bin for each dimension
bin_origin	[vector] Coordinate of the "0" bin

Value

[SparseHist] SparseHist class

Examples

```
## Data
X = base::matrix( stats::rnorm( n = 10000 ) , nrow = 5000 , ncol = 2 )
muX = SparseHist(X)

print(muX$p) ## Vector of probabilities
print(muX$c) ## Matrix of coordinates of each bins
print(muX$argwhere(X)) ## Index of bins of dataset X
```

TSMBC *TSMBC (Time Shifted Multivariate Bias Correction)*

Description

Perform a bias correction of auto-correlation

Details

Correct auto-correlation with a shift approach.

Public fields

`shift` [Shift class] Shift class to shift data.

`bc_method` [SBCK::BC_method] Underlying bias correction method.

Active bindings

`method` [character] If inverse is by row or column, see class Shift

`ref` [integer] reference column/row to inverse shift, see class

Methods

Public methods:

- `TSMBC$new()`
- `TSMBC$fit()`
- `TSMBC$predict()`
- `TSMBC$clone()`

Method `new()`**:** Create a new TSMBC object.

Usage:

```
TSMBC$new(lag, bc_method = OTC, method = "row", ref = "middle", ...)
```

Arguments:

`lag` [integer] max lag of autocorrelation

`bc_method` [SBCK::BC_METHOD] bias correction method to use after shift of data, default is OTC

`method` [character] If inverse is by row or column, see class Shift

`ref` [integer] reference column/row to inverse shift, see class Shift. Default is 0.5 * (lag+1)

`...` [] All others arguments are passed to bc_method

Returns: A new ‘TSMBC‘ object.

Method `fit()`**:** Fit the bias correction method

Usage:

```
TSMBC$fit(Y0, X0)
```

Arguments:

`Y0` [matrix: n_samples * n_features] Observations in calibration

`X0` [matrix: n_samples * n_features] Model in calibration

Returns: NULL

Method `predict()`**:** Predict the correction

Usage:

```
TSMBC$predict(X0)
```

Arguments:

`X0` [matrix: n_samples * n_features or NULL] Model in calibration

Returns: [matrix] Return the corrections of X0

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TSMBC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Robin, Y. and Vrac, M.: Is time a variable like the others in multivariate statistical downscaling and bias correction?, Earth Syst. Dynam. Discuss. [preprint], https://doi.org/10.5194/esd-2021-12, in review, 2021.

Examples

```
## arima model parameters
modelX0 = list( ar = base::c(  0.6 , 0.2 , -0.1 ) )
modelY0 = list( ar = base::c( -0.3 , 0.4 , -0.2 ) )

## arima random generator
rand.genX0 = function(n){ return(stats::rnorm( n , mean = 0.2 , sd = 1   )) }
rand.genY0 = function(n){ return(stats::rnorm( n , mean = 0   , sd = 0.7 )) }

## Generate two AR processes
X0 = stats::arima.sim( n = 1000 , model = modelX0 , rand.gen = rand.genX0 )
Y0 = stats::arima.sim( n = 1000 , model = modelY0 , rand.gen = rand.genY0 )
X0 = as.vector( X0 )
Y0 = as.vector( Y0 + 5 )

## And correct it with 30 lags
tsbc = SBCK::TSMBC$new( 30 )
tsbc$fit( Y0 , X0 )
Z0 = tsbc$predict(X0)
```

wasserstein *wasserstein distance*

Description

Compute wasserstein distance between two dataset or SparseHist X and Y

Usage

```
wasserstein(X, Y, p = 2, ot = SBCK::OTNetworkSimplex$new())
```

Arguments

X	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
Y	[matrix or SparseHist] If matrix, dim = (nrow = n_samples, ncol = n_features)
p	[float] Power of the metric (default = 2)
ot	[Optimal transport solver]

Value

[float] value of distance

References

Wasserstein, L. N. (1969). Markov processes over denumerable products of spaces describing large systems of automata. Problems of Information Transmission, 5(3), 47-52.

Examples

```
X = base::cbind( stats::rnorm(2000) , stats::rnorm(2000)  )
Y = base::cbind( stats::rnorm(2000,mean=10)  , stats::rnorm(2000) )
bw = base::c(0.1,0.1)
muX = SBCK::SparseHist( X , bw )
muY = SBCK::SparseHist( Y , bw )

## The four are equals
w2 = SBCK::wasserstein(X,Y)
w2 = SBCK::wasserstein(muX,Y)
w2 = SBCK::wasserstein(X,muY)
w2 = SBCK::wasserstein(muX,muY)
```

where *where function*

Description

This function return a vector / matrix / array of the same shape than cond / x / y such that if(cond) values are x, and else y.

Usage

```
where(cond,x,y)
```

Arguments

cond	[vector/matrix/array] Boolean values
x	[vector/matrix/array] Values if cond is TRUE
y	[vector/matrix/array] Values if cond is FALSE

Value

z [vector/matrix/array].

Examples

```
x = base::seq( -2 , 2 , length = 100 )
y = where( x < 1 , x , exp(x) ) ## y = x if x < 1, else exp(x)
```

Index