Package 'dcorBSS'

June 10, 2026

Title Distance-Correlation Based Methods for Blind Source Separation and Dependence Analysis

Version 1.0-0

Description Independent component analysis based on distance correlation, including a robust variant using the bowl transformation. The package provides user-facing implementations of distance covariance and distance correlation, including memory-efficient blockwise computations for large data sets. It includes a sequential ICA estimator based on minimizing distance correlation, as well as tools for analyzing serial dependence via distance autocorrelation, dependograms, and permutation-based tests. In addition, it provides functions for testing serial dependence based on distance correlation and the Hilbert–Schmidt independence criterion. The methodology is related to Matteson and Tsay (2017) <doi:10.1080/01621459.2016.1150851> and to the robust framework of Leyder et al. (2026) <doi:10.1007/s11634-026-00674-9>.

License GPL (>= 3)

Encoding UTF-8

LinkingTo Rcpp

Depends R (>= 4.1.0)

Imports dccpp, dHSIC, minqa, nloptr, stats, utils, Rcpp

Suggests energy, JADE, robustbase, knitr, rmarkdown

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Sarah Leyder [aut] (ORCID: <https://orcid.org/0000-0002-0828-0399>), Klaus Nordhausen [aut, cre] (ORCID: <https://orcid.org/0000-0002-3758-8501>)

Maintainer Klaus Nordhausen <klausnordhausenR@gmail.com>

VignetteBuilder knitr

Repository CRAN

Date/Publication 2026-06-10 07:30:09 UTC

Contents

`biloop_transform` *Biloop Transformation*

Description

Apply the biloop transformation to a numeric vector or matrix.

Usage

```
biloop_transform(
  X,
  c1 = 4,
  c2 = 4,
  do_scale = FALSE,
  location = NULL,
  scale = NULL
)
```

Arguments

`X`	A numeric vector or matrix. Rows correspond to observations and columns to variables.
`c1`	Positive scaling constant used before the hyperbolic tangent transformation. See details.
`c2`	Positive scaling constant applied to the first transformed coordinate. See details.
`do_scale`	Logical; if `TRUE`, standardize each variable before applying the biloop transformation.
`location`	Optional centering vector used when `do_scale = TRUE`. Defaults to `stats::median()`. Ignored otherwise.
`scale`	Optional scaling vector used when `do_scale = TRUE`. Defaults to `stats::mad()`. Ignored otherwise.

Details

The transformation maps each univariate variable to a two-dimensional nonlinear embedding $x \mapsto (u(x), v(x))$ based on two tangential circles. The data may first be robustly standardized (advised). For multivariate input, the transformation is applied columnwise and the transformed columns are concatenated.

$u(x)$ and $v(x)$ are defined as:

$$u(x) = \begin{cases} c_2(1 + \cos(2\pi \tanh(x/c_1) + \pi)), & x \geq 0 \\ -c_2(1 + \cos(2\pi \tanh(x/c_1) - \pi)), & x < 0 \end{cases}$$

$$v(x) = \sin(2\pi \tanh(x/c_2))$$

By default, no scaling is performed. If `do_scale = TRUE` and neither `location` nor `scale` is supplied, the columns are centered using `stats::median()` and scaled using `stats::mad()`. Alternatively, user-supplied location and scale vectors can be provided.

The biloop transformation can be useful when computing dependence measures on heavy-tailed or contaminated data.

Value

A numeric matrix with `nrow(X)` rows and `2 * ncol(X)` columns. If `X` is a vector, the result has two columns.

References

Leyder, S., Raymaekers, J., and Rousseeuw, P. J. (2026). Robust Distance Covariance. *International Statistical Review*, 94, 1–25. doi:10.1111/insr.70005

Examples

```
x <- rnorm(10)
biloop_transform(x)

X <- cbind(rnorm(10), rt(10, df = 3))
colnames(X) <- c("A","B")
biloop_transform(X, do_scale = TRUE)

# Robust scaling using median and Qn from robustbase
if (requireNamespace("robustbase", quietly = TRUE)) {
  loc <- apply(X, 2, median)
  scl <- apply(X, 2, robustbase::Qn)
  biloop_transform(X, do_scale = TRUE, location = loc, scale = scl)
}

# Robust distance correlation after the biloop transformation
if (requireNamespace("robustbase", quietly = TRUE)) {
  x <- rt(100, df = 3)
  y <- x^2 + rnorm(100, sd = 0.2)
  loc_x <- median(x)
  scl_x <- robustbase::Qn(x)
  loc_y <- median(y)
```

```
  scl_y <- robustbase::Qn(y)

  xb <- biloop_transform(x, do_scale = TRUE, location = loc_x, scale = scl_x)
  yb <- biloop_transform(y, do_scale = TRUE, location = loc_y, scale = scl_y)
  energy::dcor(xb, yb)
}
```

`bowl_transform` *Bowl Transformation*

Description

Apply the bowl transformation proposed for robust distance-correlation based independent component analysis. The transformation is bounded, one-to-one, and continuous, and maps far outlying observations close to the origin while preserving the key implication that zero distance correlation implies independence after transformation.

Usage

```
bowl_transform(
  X,
  alpha = 0.9975,
  do_scale = FALSE,
  location = NULL,
  scale = NULL
)
```

Arguments

`X`	A numeric data matrix or an object coercible to a matrix. Rows are observations and columns are variables.
`alpha`	A probability level used to define the cutoff $q = \sqrt{\chi^2_{p,\alpha}}$, where p is the number of columns of `X`. Must satisfy `0 < alpha < 1`.
`do_scale`	Logical; if `TRUE`, standardize the columns of `X` before applying the bowl transformation.
`location`	Optional centering vector used when `do_scale = TRUE`. Defaults to `stats::median()`. Ignored otherwise.
`scale`	Optional scaling vector used when `do_scale = TRUE`. Defaults to `stats::mad()`. Ignored otherwise.

Details

By default, the input is not standardized before the transformation is applied. Optional centering and scaling can be requested through `do_scale = TRUE`.

Let x_i be an observation, $\|x_i\|$ its Euclidean norm, and $q = \sqrt{\chi^2_{p,\alpha}}$. Define

$$u_i = \tanh(\|x_i\|/q).$$

The transformed observation is then

$$\left(10u_i^2(1-u_i)^2 x_i,\ 10u_i^6(1-u_i)^2\right).$$

Value

A numeric matrix with `nrow(X)` rows and `ncol(X) + 1` columns. The first `ncol(X)` columns contain the transformed coordinates and the final column contains the additional radial component introduced by the bowl transformation.

References

Leyder, S., Raymaekers, J., Rousseeuw, P. J., Van Deuren, T., and Verdonck, T. (2026). Independent component analysis by robust distance correlation. *Advances in Data Analysis and Classification*. doi:10.1007/s11634-026-00674-9

Examples

```
X <- cbind(rnorm(10), rt(10, df = 4))

bowl_transform(X)
bowl_transform(X, do_scale = TRUE)
bowl_transform(X, alpha = 0.99)
```

`dacf_curve` *Distance Dependogram*

Description

Computes lagwise distance covariance or distance correlation values for a univariate or multivariate time series and returns an object that can be plotted as a dependogram using `plot()`.

Usage

```
dacf_curve(
  x,
  transform = c("none", "bowl", "biloop"),
  measure = c("dcov", "dcor"),
  lags = NULL,
  block_size = NULL,
  ...
)
```

Arguments

x	A numeric vector or numeric matrix containing a time series. If x is a matrix, rows are interpreted as time points and columns as components.
transform	Character string indicating whether the data should be transformed before computing the lagwise dependence measure. One of "none", "bowl", or "biloop".
measure	Character string specifying the lagwise dependence measure. Either "dcov" for distance covariance or "dcor" for distance correlation. The default is "dcov".
lags	Integer vector of lags to compute. If NULL, the default is 1:floor(n^(2/5)).
block_size	Either NULL or one positive integer. If NULL, the standard full computation is used. If positive, the blockwise computation is used with that block size.
...	Additional arguments passed to bowl_transform() or biloop_transform() when transform is not "none".

Details

The function computes either $\mathrm{dCov}(X_t, X_{t+k})$ or $\mathrm{dCor}(X_t, X_{t+k})$ for each lag k. The choice is controlled by measure.

If block_size = NULL, the standard full pairwise-distance computation is used at each lag. If block_size is positive, a blockwise computation is used. The block size does not need to divide n - k; the final block may be smaller.

If transform = "bowl" or transform = "biloop", the transformation is applied before computing the lagwise distance autocorrelations. For multivariate input, "biloop" is allowed but is generally not recommended.

The returned object is compatible with plot.sdt(), so the lagwise values can be shown as a barplot or line plot in the same style as the dependograms produced by dcor_serial_test() and hsic_serial_test().

Value

An object of class "sdt" containing the lagwise dependence values in estimate. It can be plotted with plot().

Examples

```
x <- as.numeric(arima.sim(list(ar = 0.6), n = 200))
fit <- dacf_curve(x, lags = 1:10)
plot(fit)

y <- as.numeric(arima.sim(list(ar = 0.6), n = 200))
fit2 <- dacf_curve(y, lags = 1:10, block_size = 64L)
plot(fit2, type = "line")

# Multivariate example
set.seed(1)
X <- cbind(
  as.numeric(arima.sim(list(ar = 0.5), n = 200)),
  as.numeric(arima.sim(list(ar = -0.3), n = 200))
```

```
)
fit3 <- dacf_curve(X, lags = 1:8)
plot(fit3)
```

`dacor_large` *Distance autocorrelation with optional blockwise computation*

Description

Computes lag-`lag` distance autocorrelation for the rows of `x`. If `block_size = NULL`, the standard full pairwise-distance computation is used. If `block_size` is a positive integer, a blockwise algorithm is used that avoids allocating the full $(n - lag) \times (n - lag)$ distance matrices.

Usage

```
dacor_large(x, lag = 1L, block_size = NULL)
```

Arguments

`x`	A numeric vector or numeric matrix. Rows are observations ordered in time.
`lag`	A positive integer smaller than `nrow(x)`. The default is `1L`.
`block_size`	Either `NULL` or one positive integer. If `NULL`, the standard full computation is used. If positive, the blockwise computation is used with that block size.

Details

Vectors are treated as one-column matrices.

Distance autocorrelation is computed as the distance correlation between `x[1:(n-lag), ]` and `x[(1+lag):n, ]`.

If `block_size = NULL`, the required pairwise distances are computed using the standard full computation.

If `block_size` is a positive integer, the effective sample of size `n - lag` is partitioned into consecutive blocks of size at most `block_size`, and distances are accumulated block by block. Only one block pair is stored at a time, which can substantially reduce memory usage.

The block size does not need to divide `n - lag`. If `n - lag` is not a multiple of `block_size`, the final block is simply smaller.

The blockwise algorithm mainly reduces memory usage; the computational complexity remains of order $O((n - lag)^2)$.

Value

A numeric scalar giving the distance autocorrelation at lag `lag`.

References

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *Annals of Statistics*, **35**(6), 2769–2794. doi:10.1214/009053607000000505.

Székely, G. J. and Rizzo, M. L. (2009). Brownian distance covariance. *Annals of Applied Statistics*, **3**(4), 1236–1265. doi:10.1214/09-AOAS312.

Fokianos, K. and Pitsillou, M. (2017). Consistent testing for pairwise dependence in time series. *Technometrics*, **59**(2), 262–270. doi:10.1080/00401706.2016.1156024.

Examples

```
set.seed(1)
n <- 100
x <- matrix(rnorm(n * 2), n, 2)

r1 <- dacor_large(x, lag = 1L)
r2 <- dacor_large(x, lag = 1L, block_size = 32L)

r1
r2
all.equal(r1, r2, tolerance = 1e-10)
```

`dacov_large` *Distance autocovariance with optional blockwise computation*

Description

Computes lag-`lag` distance autocovariance for the rows of `x`. If `block_size = NULL`, the standard full pairwise-distance computation is used. If `block_size` is a positive integer, a blockwise algorithm is used that avoids allocating the full $(n - lag) \times (n - lag)$ distance matrices.

Usage

```
dacov_large(x, lag = 1L, block_size = NULL)

dacov2_large(x, lag = 1L, block_size = NULL)
```

Arguments

`x`	A numeric vector or numeric matrix. Rows are observations ordered in time.
`lag`	A positive integer smaller than `nrow(x)`. The default is `1L`.
`block_size`	Either `NULL` or one positive integer. If `NULL`, the standard full computation is used. If positive, the blockwise computation is used with that block size.

Details

`dacov_large()` returns the distance autocovariance and `dacov2_large()` returns the corresponding squared distance autocovariance.

Vectors are treated as one-column matrices.

The functions compare the segment `x[1:(n-lag), ]` with the lagged segment `x[(1+lag):n, ]` using distance covariance.

If `block_size = NULL`, all pairwise distances for these two segments are computed in the usual way.

If `block_size` is a positive integer, the same quantity is computed block by block. The effective sample size is `n - lag`, and the corresponding rows are split into consecutive blocks of size at most `block_size`. Distances are accumulated over all block pairs, so only one block pair needs to be stored at a time.

The block size does not need to divide `n - lag`. If `n - lag` is not a multiple of `block_size`, the final block is simply smaller.

The blockwise algorithm mainly reduces memory usage; the computational complexity remains of order $O((n - lag)^2)$.

Value

For `dacov_large()`, a numeric scalar giving the distance autocovariance at lag `lag`.

For `dacov2_large()`, a numeric scalar giving the squared distance autocovariance at lag `lag`.

References

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *Annals of Statistics*, **35**(6), 2769–2794. doi:10.1214/009053607000000505.

Székely, G. J. and Rizzo, M. L. (2009). Brownian distance covariance. *Annals of Applied Statistics*, **3**(4), 1236–1265. doi:10.1214/09-AOAS312.

Fokianos, K. and Pitsillou, M. (2017). Consistent testing for pairwise dependence in time series. *Technometrics*, **59**(2), 262–270. doi:10.1080/00401706.2016.1156024.

Examples

```
set.seed(1)
n <- 100
x <- matrix(rnorm(n * 2), n, 2)

v1 <- dacov2_large(x, lag = 1L)
v2 <- dacov2_large(x, lag = 1L, block_size = 32L)

v1
v2
all.equal(v1, v2, tolerance = 1e-10)

a1 <- dacov_large(x, lag = 2L)
a2 <- dacov_large(x, lag = 2L, block_size = 32L)

a1
```

```
a2
all.equal(a1, a2, tolerance = 1e-10)
```

---

dcorICA *Independent Component Analysis via Distance Correlation*

Description

Estimate an unmixing matrix by sequentially minimizing a distance-correlation criterion on whitened data. The method can be run either with the ordinary distance correlation criterion or with a robustified criterion based on the bowl transformation.

Usage

```
dcorICA(
  X,
  mu = NULL,
  scatter = NULL,
  transform = c("none", "bowl"),
  alpha = 0.9975,
  sweeps = ncol(X) + 1,
  seed = NULL,
  block_size = NULL,
  optimizer = c("cobyla", "bobyqa"),
  control = list()
)
```

Arguments

X	A numeric data matrix or an object coercible to a matrix. Rows are observations and columns are variables.
mu	Optional location vector used for whitening and centering. If NULL, the column means of X are used.
scatter	Optional positive definite scatter matrix used for whitening. If NULL, the sample covariance matrix is used.
transform	Character string specifying the dependence criterion. Possible values are "none" for ordinary distance correlation and "bowl" for the bowl-transformed robust criterion.
alpha	Probability level passed to bowl_transform() when transform = "bowl".
sweeps	Number of sweeps through the sequential optimization scheme.
seed	Optional integer random seed for the random sweep permutations.
block_size	Either NULL or one positive integer. If NULL, the standard full distance-correlation computation is used in the objective. If positive, the blockwise computation is used with that block size.

`optimizer` Character string specifying the optimizer to use. Either `"cobyla"` for `nloptr::cobyla()` or `"bobyqa"` for `minqa::bobyqa()`. Defaults to `"cobyla"`.

`control` A named list of control parameters passed to `nloptr::cobyla()` or `minqa::bobyqa()`.

Details

The returned object has class `"bss"` so that common extractor methods from package **JADE**, such as `JADE::bss.components()` and `stats::coef()` can be used directly.

The algorithm first whitens the observations using the supplied location and scatter estimates. It then searches for an orthogonal rotation that minimizes the distance correlation between one component and the remaining components, proceeding sequentially until all components have been extracted.

The dependence criterion is evaluated using `dcor_large()`. If `block_size = NULL`, the standard full pairwise-distance computation is used. If `block_size` is a positive integer, the same criterion is computed blockwise in order to reduce memory usage. The block size does not need to divide the sample size; if it does not, the final block is simply smaller.

When the sample mean and sample covariance matrix are used for whitening and the ordinary, non-transformed distance-correlation criterion is minimized, the resulting procedure is closely related in spirit to the distance covariance ICA method of Matteson and Tsay (2017). That paper develops ICA based on distance covariance and distance correlation under classical whitening. When robust location and scatter estimates such as MCD are used for whitening and the bowl-transformed distance-correlation criterion is minimized, the procedure corresponds to the robust approach referred to as PICARD (Leyder et al. 2026).

Value

An object of class `"bss"`, implemented as a list with components:

W Estimated unmixing matrix in the convention $S = (X - \mu)W^T$.

S Estimated independent components.

mu Location vector used for centering.

transform Selected dependence criterion.

alpha Bowl-transform tuning parameter.

block_size Block size used in the distance-correlation criterion, or `NULL` if the standard full computation was used.

convergence Optimization diagnostics from the sequential optimization steps.

References

Matteson, D. S. and Tsay, R. S. (2017). Independent component analysis via distance covariance. *Journal of the American Statistical Association*, **112**(518), 623–637. doi:10.1080/01621459.2016.1150851

Leyder, S., Raymaekers, J., Rousseeuw, P. J., Van Deuren, T., and Verdonck, T. (2026). Independent component analysis by robust distance correlation. *Advances in Data Analysis and Classification*. doi:10.1007/s11634-026-00674-9

See Also

bowl_transform(), dcor_large(), JADE::bss.components(), stats::coef()

Examples

```
n <- 400
S <- cbind(runif(n), rnorm(n), rchisq(n, df = 3))

A <- matrix(rnorm(ncol(S)^2), ncol = ncol(S))
X <- tcrossprod(S, A)

fit_default <- dcorICA(X, seed = 1)
fit_block <- dcorICA(X, seed = 1, block_size = 128L)
fit_bobyqa <- dcorICA(X, seed = 1, optimizer = "bobyqa")

fit_default$W
fit_block$W
fit_bobyqa$W
head(fit_default$S)

if (requireNamespace("JADE", quietly = TRUE)) {
  coef(fit_default)
  plot(fit_default)
}

# PICARD: Robust whitening + bowl criterion
if (requireNamespace("robustbase", quietly = TRUE)) {
  mcd <- robustbase::covMcd(X)
  fit_picard <- dcorICA(
    X,
    mu = mcd$center,
    scatter = mcd$cov,
    transform = "bowl",
    seed = 1
  )
  if (requireNamespace("JADE", quietly = TRUE)) {
    JADE::MD(fit_picard$W, A)
  }
}
```

---

dcor_large *Distance correlation with optional blockwise computation*

Description

Computes the distance correlation between x and y. If block_size = NULL, the standard full pairwise-distance computation is used. If block_size is a positive integer, a blockwise algorithm is used that avoids allocating the full $n \times n$ distance matrices.

Usage

```
dcor_large(x, y, block_size = NULL)
```

Arguments

x	A numeric vector or numeric matrix. Rows are observations.
y	A numeric vector or numeric matrix. Rows are observations.
block_size	Either NULL or one positive integer. If NULL, the standard full computation is used, except in the univariate case where dccpp is used. If positive, the blockwise computation is used with that block size.

Details

If block_size = NULL and both inputs are univariate, the computation is delegated to the optimized dccpp implementation. In this special case, the computational complexity is of order $O(n \log n)$, whereas the standard and blockwise multivariate implementations both have quadratic complexity.

Vectors are treated as one-column matrices.

Distance correlation is obtained by combining squared distance covariance terms for (x, y), (x, x), and (y, y).

If block_size is a positive integer, the same quantities are computed block by block. The rows of x and y are partitioned into consecutive blocks of size at most block_size. Distances are then computed only for one block pair at a time, and the contributions needed for distance covariance are accumulated across all block pairs. This can greatly reduce memory requirements for large n.

The block size does not need to divide the sample size. If the sample size is not a multiple of block_size, the final block is simply smaller.

The blockwise computation is intended for larger data sets where forming the full pairwise distance matrices may be memory intensive. It computes the same sample quantity as the standard version, up to possible small differences from floating-point arithmetic. It mainly reduces memory usage; the computational complexity remains of order $O(n^2)$.

Smaller block sizes use less memory but may be somewhat slower. Larger block sizes use more memory and approach the standard full computation.

Value

A numeric scalar, the distance correlation.

References

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *Annals of Statistics*, **35**(6), 2769–2794. doi:10.1214/009053607000000505.

Székely, G. J. and Rizzo, M. L. (2009). Brownian distance covariance. *Annals of Applied Statistics*, **3**(4), 1236–1265. doi:10.1214/09-AOAS312.

Chaudhuri, A. and Hu, W. (2019). A fast algorithm for computing distance correlation. *Computational Statistics & Data Analysis*, **135**, 15–24. doi:10.1016/j.csda.2019.01.016.

Examples

```
n <- 100
x <- matrix(rnorm(n * 3), n, 3)
y <- matrix(rnorm(n * 2), n, 2)

r1 <- dcor_large(x, y)
r2 <- dcor_large(x, y, block_size = 32L)

r1
r2
all.equal(r1, r2, tolerance = 1e-10)
```

`dcor_serial_test` *Test Serial Dependence Using Distance Covariance or Distance Correlation*

Description

Test whether a univariate time series exhibits serial dependence using a portmanteau-type test based on lagwise distance covariance or distance correlation statistics.

Usage

```
dcor_serial_test(
  x,
  transform = c("none", "bowl", "biloop"),
  measure = c("dcov", "dcor"),
  type = c("FP", "BP"),
  kernel = "bartlett",
  B = 499L,
  lags = NULL,
  seed = NULL,
  block_size = NULL,
  ...
)
```

Arguments

`x`	A numeric vector, or a one-column numeric matrix, containing a univariate time series.
`transform`	Character string indicating whether the data should be transformed before computing the lagwise dependence measure. One of `"none"`, `"bowl"`, or `"biloop"`.
`measure`	Character string specifying the lagwise dependence measure. Either `"dcov"` for distance covariance or `"dcor"` for distance correlation. The default is `"dcov"`.
`type`	Character string specifying the portmanteau statistic. Either `"BP"` for a Box–Pierce type statistic or `"FP"` for the kernel-weighted Fokianos-Pitsillou statistic.

kernel	Character string specifying the kernel used when type = "FP". Currently only "bartlett" is implemented.
B	Number of permutation resamples used to approximate the null distribution.
lags	Integer vector of lags to include in the test statistic. If NULL, the default is 1:floor(n^(2/5)). It is recommended to use a vector of the form 1:m, where m is the maximum number of lags.
seed	Optional random seed.
block_size	Either NULL or one positive integer. If NULL, the standard full computation is used for the lagwise distance covariance or distance correlation values. If positive, the blockwise computation is used with that block size.
...	Additional arguments passed to bowl_transform() or biloop_transform() when transform is not "none".

Details

For each lag k, the procedure computes a dependence measure D_k between X_t and X_{t+k}. The measure is either distance covariance or distance correlation.

If type = "BP", the test statistic is

$$T_{BP} = n \sum_{k \in \mathcal{L}} D_k^2.$$

If type = "FP", the test statistic is

$$T_{H96} = \sum_{k \in \mathcal{L}} (n-k) K^2(k/m) D_k^2,$$

where K is a kernel function and m is the number of selected lags. Currently, the Bartlett kernel is used:

$$K(z) = \begin{cases} 1 - |z|, & |z| \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

The null distribution is approximated by random permutations of the observed series. This targets the null hypothesis of serial independence. The procedure is simple and broadly applicable, but it is computationally more intensive than classical autocorrelation-based tests.

The procedure assumes that the time series is completely observed. Missing observations should therefore be handled before applying the test.

In practical applications, a larger number of permutation resamples is recommended to obtain stable and reliable p-values. The default B = 499 is intended for exploratory use, whereas values such as B = 2000 or larger are recommended for reporting final results.

If transform = "bowl" or transform = "biloop", the series is first transformed and the distance correlation is then computed on the transformed data. By default, scaling is enabled for these transformations, unless the user explicitly supplies do_scale = FALSE.

Robust scaling can be supplied through the location and scale arguments. For example, for univariate data one may use the sample median together with robustbase::Qn.

Value

An object of classes "sdt" and "htest".

References

Fokianos, K. and Pitsillou, M (2017). Consistent testing for pairwise dependence in time series. *Technometrics*, **59**(2), 262–270 doi:10.1080/00401706.2016.1156024

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *Annals of Statistics*, **35**(6), 2769–2794. doi:10.1214/009053607000000505.

Székely, G. J. and Rizzo, M. L. (2009). Brownian distance covariance. *Annals of Applied Statistics*, **3**(4), 1236–1265. doi:10.1214/09-AOAS312.

Examples

```
x <- rnorm(200)
dcor_serial_test(x, B = 99)

y <- as.numeric(arima.sim(list(ar = 0.6), n = 200))

dcor_serial_test(y, B = 99)
dcor_serial_test(y, B = 99, measure = "dcor")
dcor_serial_test(y, B = 99, type = "FP")
dcor_serial_test(y, B = 99, transform = "biloop")


# robust scaling example
ymat <- matrix(y, ncol = 1)
loc <- apply(ymat, 2, stats::median)
sc <- apply(ymat, 2, robustbase::Qn)

dcor_serial_test(
  y,
  transform = "bowl",
  B = 99,
  location = loc,
  scale = sc
)

dcor_serial_test(y, B = 99, block_size = 64L)
```

`dcov_large` *Distance covariance with optional blockwise computation*

Description

Computes distance covariance between `x` and `y`. If `block_size = NULL`, the standard full pairwise-distance computation is used. If `block_size` is a positive integer, a blockwise algorithm is used that avoids allocating the full $n \times n$ distance matrices.

Usage

```
dcov_large(x, y, block_size = NULL)

dcov2_large(x, y, block_size = NULL)
```

Arguments

x	A numeric vector or numeric matrix. Rows are observations.
y	A numeric vector or numeric matrix. Rows are observations.
block_size	Either NULL or one positive integer. If NULL, the standard full computation is used, except in the univariate case where dccpp is used. If positive, the blockwise computation is used with that block size.

Details

dcov_large() returns the distance covariance and thus behaves like dcov. dcov2_large() returns the corresponding squared distance covariance.

Vectors are treated as one-column matrices.

The functions compute the biased V-statistic version of distance covariance based on pairwise Euclidean distances between the rows of x and y.

If block_size = NULL, all pairwise distances are computed in the usual way. In the special case where both x and y are univariate, the optimized dccpp implementation is used instead. This implementation has computational complexity of order $O(n \log n)$, making it preferable to the blockwise approach in the univariate setting.

If block_size is a positive integer, the same quantity is computed block by block. The observations are split into consecutive blocks of size at most block_size, and pairwise distances are accumulated over all block pairs. This reduces memory usage because only distances for one block pair are stored at a time rather than the full distance matrices.

The block size does not need to divide the sample size. If n is not a multiple of block_size, the final block simply contains the remaining observations and is processed in the same way as the other blocks.

The blockwise result is intended to agree with the standard computation up to numerical differences due to floating-point arithmetic. It mainly reduces memory usage; the computational complexity remains of order $O(n^2)$.

Smaller block sizes reduce peak memory usage but may increase computation time due to additional block overhead. Larger block sizes are closer to the standard full computation in memory behavior.

Value

For dcov_large(), a numeric scalar giving the distance covariance.

For dcov2_large(), a numeric scalar giving the squared distance covariance.

References

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *Annals of Statistics*, **35**(6), 2769–2794. doi:10.1214/009053607000000505.

Székely, G. J. and Rizzo, M. L. (2009). Brownian distance covariance. *Annals of Applied Statistics*, **3**(4), 1236–1265. doi:10.1214/09-AOAS312.

Chaudhuri, A. and Hu, W. (2019). A fast algorithm for computing distance correlation. *Computational Statistics & Data Analysis*, **135**, 15–24. doi:10.1016/j.csda.2019.01.016.

Examples

```
n <- 100
x <- matrix(rnorm(n * 3), n, 3)
y <- matrix(rnorm(n * 2), n, 2)

v1 <- dcov2_large(x, y)
v2 <- dcov2_large(x, y, block_size = 32L)

v1
v2
all.equal(v1, v2, tolerance = 1e-10)

d1 <- dcov_large(x, y)
d2 <- dcov_large(x, y, block_size = 32L)

d1
d2
all.equal(d1, d2, tolerance = 1e-10)

# Compare with energy::dcov()
if (requireNamespace("energy", quietly = TRUE)) {

 d_energy <- energy::dcov(x, y)
 d_large  <- dcov_large(x, y)

 d_energy
 d_large

 all.equal(d_energy, d_large, tolerance = 1e-10)
}
```

`hsic_serial_test` *Test Serial Dependence Using HSIC*

Description

Test whether a univariate time series exhibits serial dependence using a portmanteau-type test based on Hilbert-Schmidt Independence Criterion (HSIC) statistics over a set of lags.

Usage

```
hsic_serial_test(
  x,
  B = 499L,
  lags = NULL,
  normalize = TRUE,
  type = c("H96", "BP"),
  kernel = "bartlett",
  seed = NULL
)
```

Arguments

x	A numeric vector, or a one-column numeric matrix, containing a univariate time series.
B	Number of permutation resamples used to approximate the null distribution.
lags	Integer vector of lags to include in the test statistic. If NULL, the default is 1:floor(n^(2/5)). It is recommended to use a vector of the form 1:m, where m is the maximum number of lags.
normalize	Logical; if TRUE, the test statistic is based on the normalized HSIC values computed by nHSIC. If FALSE, the unnormalized HSIC values returned by dhsic are used.
type	Character string specifying the portmanteau statistic. Either "BP" for a Box–Pierce type statistic or "H96" for a kernel-weighted Hong-type statistic.
kernel	Character string specifying the kernel used when type = "H96". Currently only "bartlett" is implemented.
seed	Optional random seed.

Details

For each lag k, the procedure computes a dependence measure D_k between X_t and X_{t+k}. The measure D_k is either HSIC or normalized HSIC, depending on the value of normalize. By default, normalize = TRUE.

If type = "BP", the test statistic is the Box–Pierce type statistic

$$T_{BP} = n \sum_{k \in \mathcal{L}} D_k^2,$$

where $\mathcal{L}$ is the set of selected lags.

If type = "H96", the test statistic is the kernel-weighted Hong-type statistic

$$T_{H96} = n \sum_{k \in \mathcal{L}} K^2(k/m) D_k^2,$$

where K is a kernel function and m is the number of selected lags. Currently, the Bartlett kernel is used:

$$K(z) = \begin{cases} 1 - |z|, & |z| \le 1 \\ 0, & \text{otherwise} \end{cases}$$

The null distribution is approximated by random permutations of the observed series. This targets the null hypothesis of serial independence.

The procedure assumes that the time series is completely observed. Missing observations should therefore be handled before applying the test.

In practical applications, a larger number of permutation resamples is recommended to obtain stable and reliable p-values. The default `B = 499` is intended for exploratory use, whereas values such as `B = 2000` or larger are recommended for reporting final results.

Value

An object of classes `"sdt"` and `"htest"` with components:

statistic Observed test statistic.

p.value Permutation p-value.

method Description of the procedure.

data.name Name of the supplied data object.

estimate HSIC or normalized HSIC values at the selected lags.

parameter Number of resamples, number of lags used, whether normalization was used, and the statistic type.

null.value Null hypothesis value.

alternative The alternative hypothesis.

lags Integer vector of lags used in the test.

measure_name Name of the dependence measure.

type Type of portmanteau statistic.

References

Hong, Y. (1996). Consistent testing for serial correlation of unknown form. *Econometrica*, **64**(4), 837–864. doi:10.2307/2171847.

Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. *Algorithmic Learning Theory*, 63–77. doi:10.1007/11564089_7.

Examples

```
x <- rnorm(80)
hsic_serial_test(x, B = 99)

y <- as.numeric(arima.sim(list(ar = 0.6), n = 80))

# Default: normalized HSIC with BP-type statistic
hsic_serial_test(y, B = 99, lags = 1:3)


# Unnormalized HSIC with BP-type statistic
hsic_serial_test(y, B = 99, normalize = FALSE)
```

```
# Normalized HSIC with kernel-weighted H96-type statistic
hsic_serial_test(y, B = 99, type = "H96")

# Compare both statistic types
hsic_serial_test(y, B = 99, type = "BP")
hsic_serial_test(y, B = 99, type = "H96")
```

`nHSIC` *Normalized Hilbert-Schmidt Independence Criterion*

Description

Computes a normalized version of the Hilbert-Schmidt Independence Criterion (HSIC) between two random vectors.

Usage

```
nHSIC(x, y)
```

Arguments

`x`	A numeric vector or numeric matrix. Rows correspond to observations.
`y`	A numeric vector or numeric matrix. Rows correspond to observations.

Details

The normalization rescales HSIC to reduce the influence of marginal scale effects and yields a quantity analogous in spirit to a correlation measure.

The normalized HSIC is defined as

$$\mathrm{nHSIC}(X,Y) = \sqrt{\frac{\mathrm{HSIC}(X,Y)}{\sqrt{\mathrm{HSIC}(X,X)\mathrm{HSIC}(Y,Y)}}}.$$

Here, $\mathrm{HSIC}(X,Y)$ denotes the empirical HSIC value computed using `dhsic`.

Vectors are treated as one-column matrices.

The function computes empirical HSIC values for `(x, y)`, `(x, x)`, and `(y, y)` and combines them according to the normalization formula above.

If the normalization denominator is numerically non-positive, the function returns `0`.

Small negative values caused by floating-point arithmetic are truncated to zero before taking the square root.

Value

A numeric scalar giving the normalized HSIC value.

References

Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. *Algorithmic Learning Theory*, 63–77. doi:10.1007/11564089_7.

Examples

```
set.seed(1)

n <- 200

x <- matrix(rnorm(n), ncol = 1)
y <- x^2 + 0.3 * rnorm(n)

# Standard HSIC
dHSIC::dhsic(x, y)$dHSIC

# Normalized HSIC
nHSIC(x, y)

# Independent variables
z <- matrix(rnorm(n), ncol = 1)

dHSIC::dhsic(x, z)$dHSIC
nHSIC(x, z)
```

---

sdt *Serial Dependence Test Objects and Dependograms*

Description

Objects of class "sdt" are returned by dcor_serial_test() and hsic_serial_test(). They inherit from class "htest" and contain additional components for lagwise dependence measures, which can be plotted as dependograms using plot().

Usage

```
## S3 method for class 'sdt'
plot(
  x,
  type = c("bar", "line"),
  xlab = "Lag",
  ylab = NULL,
  main = NULL,
  add_h0 = TRUE,
  pch = 19,
  lty = 1,
  ...
)
```

Arguments

x	An object of class "sdt".
type	Type of dependogram. Either "bar" for a barplot or "line" for a line plot.
xlab	Label for the x-axis.
ylab	Label for the y-axis. If omitted, a default based on the stored dependence measure is used.
main	Main title. If omitted, a default title is used.
add_h0	Logical; if TRUE, add a horizontal reference line at zero.
pch	Plotting character used for type = "line".
lty	Line type used for type = "line".
...	Further arguments passed to graphics::barplot() or graphics::plot().

Details

An object of class "sdt" contains the usual "htest" components, together with:

estimate Named vector of lagwise dependence values.

lags Integer vector of lags used in the test.

measure_name Character string naming the dependence measure, for example "dCor" or "HSIC".

The plot method displays the lagwise dependence values either as a barplot or as a line plot. For kernel-weighted tests such as type = "H96", the plot shows the unweighted lagwise dependence values, not the weighted contributions to the test statistic.

Value

The plot method is called for its side effect of producing a plot and returns the object invisibly.

Examples

```
x <- as.numeric(arima.sim(list(ar = 0.6), n = 100))
fit <- dcor_serial_test(x, B = 99, lags = 1:10)
plot(fit)

fit2 <- hsic_serial_test(x, B = 99, lags = 1:10)
plot(fit2, type = "line")
```

Index