dcor Documentation

Release 0.3

Author

Jun 06, 2019

Contents:

Distance covariance and distance correlation are dependency measures between random vectors introduced in [ASRB07].

This package provide functions for calculating several statistics related with distance covariance and distance correlation, including biased and unbiased estimators of both dependency measures.

CHAPTER 1

References

1.1 Installation

dcor is on PyPi and can be installed using `pip`:

```
pip install dcor
```

It is also available for `conda`:

```
conda install -c vnmabus dcor
```

1.1.1 Requirements

dcor is available in Python 3.5 or above and in Python 2.7, in all operating systems.

1.2 Theory

This section provides an explanation of the distance measures provided by this package (distance covariance and distance correlation). The package can be used without a deep understanding of the mathematics involved, so feel free to skip this chapter.

1.2.1 Distance covariance and distance correlation

Distance covariance and distance correlation are recently introduced dependency measures between random vectors [CSRB07]. Let X and Y be two random vectors with finite first moments, and let ϕ_X and ϕ_Y be the respective characteristic functions

$$\phi_X(t) = \mathbb{E}[e^{itX}]$$
$$\phi_Y(t) = \mathbb{E}[e^{itY}]$$

Let $\phi_{X,Y}$ be the joint characteristic function. Then, if X and Y take values in $\mathbb{R}^p$ and $\mathbb{R}^q$ respectively, the distance covariance between them $\mathcal{V}(X,Y)$, or $\mathrm{dCov}(X,Y)$, is the non-negative number defined by

$$\mathcal{V}^2(X,Y) = \int_{\mathbb{R}^{p+q}} |\phi_{X,Y}(t,s) - \phi_X(t)\phi_Y(s)|^2 w(t,s)dtds,$$

where $w(t,s) = (c_p c_q |t|_p^{1+p} |s|_q^{1+q})^{-1}$, $|\cdot|_d$ is the euclidean norm in $\mathbb{R}^d$ and $c_d = \frac{\pi^{(1+d)/2}}{\Gamma((1+d)/2)}$ is half the surface area of the unit sphere in $\mathbb{R}^d$. The distance correlation $\mathcal{R}(X,Y)$, or $\mathrm{dCor}(X,Y)$, is defined as

$$\mathcal{R}^2(X,Y) = \begin{cases} \frac{\mathcal{V}^2(X,Y)}{\mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y)} & \text{if } \mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y) > 0 \\ 0 & \text{if } \mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y) = 0. \end{cases}$$

1.2.1.1 Properties

The distance covariance has the following properties:

- $\mathcal{V}(X,Y) \geq 0$.
- $\mathcal{V}(X,Y) = 0$ if and only if X and Y are independent.
- $\mathcal{V}(X,Y) = \mathcal{V}(Y,X)$.
- $\mathcal{V}^2(\mathbf{a}_1 + b_1\mathbf{C}_1X, \mathbf{a}_2 + b_2\mathbf{C}_2Y) = |b_1b_2|\mathcal{V}^2(Y,X)$ for all constant real-valued vectors $\mathbf{a}_1, \mathbf{a}_2$, scalars b_1, b_2 and orthonormal matrices $\mathbf{C}_1, \mathbf{C}_2$.
- If the random vectors (X_1, Y_1) and (X_2, Y_2) are independent then

$$\mathcal{V}(X_1 + X_2, Y_1 + Y_2) \leq \mathcal{V}(X_1, Y_1) + \mathcal{V}(X_2, Y_2).$$

The distance correlation has the following properties:

- $0 \leq \mathcal{R}(X,Y) \leq 1$.
- $\mathcal{R}(X,Y) = 0$ if and only if X and Y are independent.
- If $\mathcal{R}(X,Y) = 1$ then there exists a vector $\mathbf{a}$, a nonzero real number b and an orthogonal matrix $\mathbf{C}$ such that $Y = \mathbf{a} + b\mathbf{C}X$.

1.2.1.2 Estimators

Distance covariance has an estimator with a simple form. Suppose that we have n observations of X and Y, denoted by x and y. We denote as x_i the i-th observation of x, and y_i the i-th observation of y. If we define $a_{ij} = |x_i - x_j|_p$ and $b_{ij} = |y_i - y_j|_q$, the corresponding double centered matrices are defined by $(A_{i,j})_{i,j=1}^n$ and $(B_{i,j})_{i,j=1}^n$

$$A_{i,j} = a_{i,j} - \frac{1}{n}\sum_{l=1}^n a_{il} - \frac{1}{n}\sum_{k=1}^n a_{kj} + \frac{1}{n^2}\sum_{k,l=1}^n a_{kl},$$
$$B_{i,j} = b_{i,j} - \frac{1}{n}\sum_{l=1}^n b_{il} - \frac{1}{n}\sum_{k=1}^n b_{kj} + \frac{1}{n^2}\sum_{k,l=1}^n b_{kl}.$$

Then

$$\mathcal{V}_n^2(x,y) = \frac{1}{n^2}\sum_{i,j=1}^n A_{i,j}B_{i,j}$$

is called the squared sample distance covariance, and it is an estimator of $\mathcal{V}^2(X, Y)$. The sample distance correlation $\mathcal{R}_n(x, y)$ can be obtained as the standardized sample covariance

$$\mathcal{R}_n^2(x,y) = \begin{cases} \frac{\mathcal{V}_n^2(x,y)}{\mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y)}, & \text{if } \mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y) > 0, \\ 0, & \text{if } \mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y) = 0. \end{cases}$$

These estimators have the following properties:

- $\mathcal{V}_n^2(x, y) \geq 0$
- $0 \leq \mathcal{R}_n^2(x, y) \leq 1$

In a similar way one can define an unbiased estimator $\Omega_n(x, y)$ of the squared distance covariance $\mathcal{V}^2(X, Y)$. Given the previous definitions of a_{ij} and b_{ij}, we define the U-centered matrices $(\widetilde{A}_{i,j})_{i,j=1}^n$ and $(\widetilde{B}_{i,j})_{i,j=1}^n$

$$\begin{aligned} \widetilde{A}_{i,j} &= \begin{cases} a_{i,j} - \frac{1}{n-2}\sum_{l=1}^n a_{il} - \frac{1}{n-2}\sum_{k=1}^n a_{kj} + \frac{1}{(n-1)(n-2)}\sum_{k,l=1}^n a_{kl}, & \text{if } i \neq j, \\ 0, & \text{if } i = j, \end{cases} \\ \widetilde{B}_{i,j} &= \begin{cases} b_{i,j} - \frac{1}{n-2}\sum_{l=1}^n b_{il} - \frac{1}{n-2}\sum_{k=1}^n b_{kj} + \frac{1}{(n-1)(n-2)}\sum_{k,l=1}^n b_{kl}, & \text{if } i \neq j, \\ 0, & \text{if } i = j. \end{cases} \end{aligned} \tag{1.1}$$

Then, $\Omega_n(x, y)$ is defined as

$$\Omega_n(x,y) = \frac{1}{n(n-3)}\sum_{i,j=1}^n \widetilde{A}_{i,j}\widetilde{B}_{i,j}.$$

We can also obtain an estimator of $\mathcal{R}^2(X, Y)$ using $\Omega_n(x, y)$, as we did with $\mathcal{V}_n^2(x, y)$. $\Omega_n(x, y)$ does not verify that $\Omega_n(x, y) \geq 0$, because sometimes could take negative values near 0. There is an algorithm that can compute $\Omega_n(x, y)$ for random variables with $O(n \log n)$ complexity [CHS16]. Since the estimator formulas explained above have complexity $O(n^2)$, this improvement is significant, specially for larger samples.

1.2.2 Partial distance covariance and partial distance correlation

Partial distance covariance and partial distance correlation are dependency measures between random vectors, based on distance covariance and distance correlation, in with the effect of a random vector is removed [CSR14]. The population partial distance covariance $\mathcal{V}^*(X, Y; Z)$, or $\text{pdCov}^*(X, Y; Z)$, between two random vectors X and Y with respect to a random vector Z is

$$\mathcal{V}^*(X,Y;Z) = \begin{cases} \mathcal{V}^2(X,Y) - \frac{\mathcal{V}^2(X,Z)\mathcal{V}^2(Y,Z)}{\mathcal{V}^2(Z,Z)} & \text{if } \mathcal{V}^2(Z,Z) \neq 0 \\ \mathcal{V}^2(X,Y) & \text{if } \mathcal{V}^2(Z,Z) = 0 \end{cases}$$

where $\mathcal{V}^2(\,\cdot\,,\,\cdot\,)$ is the squared distance covariance.

The corresponding partial distance correlation $\mathcal{R}^*(X, Y; Z)$, or $\text{pdCor}^*(X, Y; Z)$, is

$$\mathcal{R}^*(X,Y;Z) = \begin{cases} \frac{\mathcal{R}^2(X,Y) - \mathcal{R}^2(X,Z)\mathcal{R}^2(Y,Z)}{\sqrt{1-\mathcal{R}^4(X,Z)}\sqrt{1-\mathcal{R}^4(Y,Z)}} & \text{if } \mathcal{R}^4(X,Z) \neq 1 \text{ and } \mathcal{R}^4(Y,Z) \neq 1 \\ 0 & \text{if } \mathcal{R}^4(X,Z) = 1 \text{ or } \mathcal{R}^4(Y,Z) = 1 \end{cases}$$

where $\mathcal{R}(\,\cdot\,,\,\cdot\,)$ is the distance correlation.

1.2.2.1 Estimators

As in distance covariance and distance correlation, the U-centered distance matrices $\widetilde{A}_{i,j}$, $\widetilde{B}_{i,j}$ and $\widetilde{C}_{i,j}$ corresponding with the samples x, y and z taken from the random vectors X, Y and Z can be computed using using (1.1).

The set of all U-centered distance matrices is a Hilbert space with the inner product

$$\langle \widetilde{A}, \widetilde{B} \rangle = \frac{1}{n(n-3)} \sum_{i,j=1}^{n} \widetilde{A}_{i,j} \widetilde{B}_{i,j}.$$

Then, the projection of a sample x over z can be taken in this Hilbert space using the associated matrices, as

$$P_z(x) = \frac{\langle \widetilde{A}, \widetilde{C} \rangle}{\langle \widetilde{C}, \widetilde{C} \rangle} \widetilde{C}.$$

The complementary projection is then

$$P_{z^\perp}(x) = \widetilde{A} - P_z(x) = \widetilde{A} - \frac{\langle \widetilde{A}, \widetilde{C} \rangle}{\langle \widetilde{C}, \widetilde{C} \rangle} \widetilde{C}.$$

We can now define the sample partial distance covariance as

$$\mathcal{V}_n^*(x, y; z) = \langle P_{z^\perp}(x), P_{z^\perp}(y) \rangle$$

The sample distance correlation is defined as the cosine of the angle between the vectors $P_{z^\perp}(x)$ and $P_{z^\perp}(y)$

$$\mathcal{R}_n^*(x, y; z) = \begin{cases} \frac{\langle P_{z^\perp}(x), P_{z^\perp}(y) \rangle}{||P_{z^\perp}(x)|| ||P_{z^\perp}(y)||} & \text{if } ||P_{z^\perp}(x)|| ||P_{z^\perp}(y)|| \neq 0 \\ 0 & \text{if } ||P_{z^\perp}(x)|| ||P_{z^\perp}(y)|| = 0 \end{cases}$$

1.2.3 Energy distance

Energy distance is an statistical distance between random vectors $X, Y \in \mathbb{R}^d$ [CSR13], defined as

$$\mathcal{E}(X, Y) = 2\mathbb{E}(||X - Y||) - \mathbb{E}(||X - X'||) - \mathbb{E}(||Y - Y'||)$$

where X' and Y' are independent and identically distributed copies of X and Y, respectively.

It can be proved that, if the characteristic functions of X and Y are $\phi_X(t)$ and $\phi_Y(t)$ the energy distance can be alternatively written as

$$\mathcal{E}(X, Y) = \frac{1}{c_d} \int_{\mathbb{R}^d} \frac{|\phi_X(t) - \phi_Y(t)|^2}{||t||^{d+1}} dt$$

where again $c_d = \frac{\pi^{(1+d)/2}}{\Gamma((1+d)/2)}$ is half the surface area of the unit sphere in $\mathbb{R}^d$.

1.2.3.1 Estimator

Suppose that we have n_1 observations of X and n_2 observations of Y, denoted by x and y. We denote as x_i the i-th observation of x, and y_i the i-th observation of y. Then, an estimator of the energy distance is

$$\mathcal{E}_{\setminus\infty,\setminus\in}(x, y) = \frac{2}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} ||x_i - y_j|| - \frac{1}{n_1^2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_1} ||x_i - x_j|| - \frac{1}{n_2^2} \sum_{i=1}^{n_2} \sum_{j=1}^{n_2} ||y_i - y_j||$$

1.2.4 References

1.3 Comparison between Python's 'dcor' and R's 'energy'

The 'energy' package for R provides an implementation of the E-statistics in this package programmed by the original authors of these statistics.

This package is inspired by 'energy', and tries to bring the same functionality to a Python audience.

In this section, both packages are compared, to give an overview of the differences, and to make porting code between R and Python easier.

1.3.1 Table of energy-dcor equivalents

energy (R)	dcor (Python)	Notes
`dx <- dist(x)` `DX <- as.matrix(dx)`	`DX = dcor.distances.pairwise_distances(x)`	Not really part of 'energy', but the 'stats' package. In Python it returns a numpy array, while in R it returns a matrix object
`D_center(DX)`	`dcor.double_centered(DX)`	
`U_center(DX)`	`dcor.u_centered(DX)`	
	`dcor.mean_product(U, V)`	Provided for symmetry with *dcor.u_product*, although the implementation is very simple
`U_product(U, V)`	`dcor.u_product(U, V)`	
	`dcor.u_projection(U)`	
	`dcor.u_complementary_projection(U)`	
`dcov(x, y)`	`dcor.distance_covariance(x, y)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`dcov(x, y, index = 0.5)`	`dcor.distance_covariance(x, y, exponent = 0.5)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`dcor(x, y)`	`dcor.distance_correlation(x, y)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`dcor(x, y, index = 0.5)`	`dcor.distance_correlation(x, y, exponent = 0.5)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`DCOR(x, y)`	`dcor.distance_stats(x, y)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`DCOR(x, y, index = 0.5)`	`dcor.distance_stats(x, y, exponent = 0.5)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
`dcovU(x, y)`	`dcor.u_distance_covariance_sqr(x, y)`	In 'energy', the distance matrix can be computed beforehand. That is not currently possible in 'dcor'
	`dcor.u_distance_covariance_sqr(x, y, exponent = 0.5)`	
		In 'energy', the distance matrix can

1.4 API List

1.4.1 List of functions

A complete list of all functions provided by dcor.

1.4.1.1 Biased estimators for distance covariance and distance correlation

These functions compute the usual (biased) estimators for the distance covariance and distance correlation and their squares.

dcor.distance_covariance(x, y, *[, exponent])	Computes the usual (biased) estimator for the distance covariance between two random vectors.
dcor.distance_covariance_sqr(x, y, *[, exponent])	Computes the usual (biased) estimator for the squared distance covariance between two random vectors.
dcor.distance_correlation(x, y, *[, exponent])	Computes the usual (biased) estimator for the distance correlation between two random vectors.
dcor.distance_correlation_sqr(x, y, *[, ...])	Computes the usual (biased) estimator for the squared distance correlation between two random vectors.
dcor.distance_stats(x, y, *[, exponent])	Computes the usual (biased) estimators for the distance covariance and distance correlation between two random vectors, and the individual distance variances.
dcor.distance_stats_sqr(x, y, *[, exponent])	Computes the usual (biased) estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.

dcor.distance_covariance

distance_covariance(*x*, *y*, *, *exponent=1*)

Computes the usual (biased) estimator for the distance covariance between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Biased estimator of the distance covariance.

Return type numpy scalar

See also:

distance_covariance_sqr(), *u_distance_covariance_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_covariance(a, a) # doctest: +ELLIPSIS
7.2111025...
>>> dcor.distance_covariance(a, b)
1.0
>>> dcor.distance_covariance(b, b)
0.5
>>> dcor.distance_covariance(a, b, exponent=0.5)
0.6087614...
```

dcor.distance_covariance_sqr

distance_covariance_sqr(*x*, *y*, *, *exponent=1*)

Computes the usual (biased) estimator for the squared distance covariance between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Biased estimator of the squared distance covariance.

Return type numpy scalar

See also:

distance_covariance(), *u_distance_covariance_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
```

(continues on next page)

(continued from previous page)

```
>>> dcor.distance_covariance_sqr(a, a)
52.0
>>> dcor.distance_covariance_sqr(a, b)
1.0
>>> dcor.distance_covariance_sqr(b, b)
0.25
>>> dcor.distance_covariance_sqr(a, b, exponent=0.5) # doctest: +ELLIPSIS
0.3705904...
```

dcor.distance_correlation

distance_correlation(*x*, *y*, *, *exponent=1*)

Computes the usual (biased) estimator for the distance correlation between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Value of the biased estimator of the distance correlation.

Return type numpy scalar

See also:

distance_correlation_sqr(), *u_distance_correlation_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_correlation(a, a)
1.0
>>> dcor.distance_correlation(a, b) # doctest: +ELLIPSIS
0.5266403...
>>> dcor.distance_correlation(b, b)
1.0
>>> dcor.distance_correlation(a, b, exponent=0.5) # doctest: +ELLIPSIS
0.6703214...
```

dcor.distance_correlation_sqr

distance_correlation_sqr(*x*, *y*, ***, *exponent=1*)

Computes the usual (biased) estimator for the squared distance correlation between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Value of the biased estimator of the squared distance correlation.

Return type numpy scalar

See also:

distance_correlation(), *u_distance_correlation_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_correlation_sqr(a, a)
1.0
>>> dcor.distance_correlation_sqr(a, b) # doctest: +ELLIPSIS
0.2773500...
>>> dcor.distance_correlation_sqr(b, b)
1.0
>>> dcor.distance_correlation_sqr(a, b, exponent=0.5) # doctest: +ELLIPSIS
0.4493308...
```

dcor.distance_stats

distance_stats(*x*, *y*, ***, *exponent=1*)

Computes the usual (biased) estimators for the distance covariance and distance correlation between two random vectors, and the individual distance variances.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Distance covariance, distance correlation, distance variance of the first random vector and distance variance of the second random vector.

Return type Stats

See also:

distance_covariance(), *distance_correlation()*

Notes

It is less efficient to compute the statistics separately, rather than using this function, because some computations can be shared.

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_stats(a, a) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=7.2111025..., correlation_xy=1.0,
variance_x=7.2111025..., variance_y=7.2111025...)
>>> dcor.distance_stats(a, b) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=1.0, correlation_xy=0.5266403...,
variance_x=7.2111025..., variance_y=0.5)
>>> dcor.distance_stats(b, b) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=0.5, correlation_xy=1.0, variance_x=0.5,
variance_y=0.5)
>>> dcor.distance_stats(a, b, exponent=0.5) # doctest: +ELLIPSIS
...                                 # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=0.6087614..., correlation_xy=0.6703214...,
variance_x=1.6495217..., variance_y=0.5)
```

dcor.distance_stats_sqr

distance_stats_sqr(*x*, *y*, *, *exponent=1*)

Computes the usual (biased) estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Squared distance covariance, squared distance correlation, squared distance variance of the first random vector and squared distance variance of the second random vector.

Return type Stats

See also:

distance_covariance_sqr(), *distance_correlation_sqr()*

Notes

It is less efficient to compute the statistics separately, rather than using this function, because some computations can be shared.

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_stats_sqr(a, a) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=52.0, correlation_xy=1.0, variance_x=52.0,
variance_y=52.0)
>>> dcor.distance_stats_sqr(a, b) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=1.0, correlation_xy=0.2773500...,
variance_x=52.0, variance_y=0.25)
>>> dcor.distance_stats_sqr(b, b) # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=0.25, correlation_xy=1.0, variance_x=0.25,
variance_y=0.25)
>>> dcor.distance_stats_sqr(a, b, exponent=0.5) # doctest: +ELLIPSIS
...                                 # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=0.3705904..., correlation_xy=0.4493308...,
variance_x=2.7209220..., variance_y=0.25)
```

1.4.1.2 Unbiased and bias-corrected estimators for distance covariance and distance correlation

These functions compute the unbiased estimators for the square of the distance covariance and the bias corrected estimator for the square of the distance correlation. As these estimators are signed, no functions are provided for taking the square root.

dcor.u_distance_covariance_sqr(x, y, *[, ...])	Computes the unbiased estimator for the squared distance covariance between two random vectors.

Continued on next page

Table 2 – continued from previous page

dcor.u_distance_correlation_sqr(x, y, *[, ...])	Computes the bias-corrected estimator for the squared distance correlation between two random vectors.
dcor.u_distance_stats_sqr(x, y, *[, exponent])	Computes the unbiased estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.

dcor.u_distance_covariance_sqr

u_distance_covariance_sqr (*x*, *y*, *, *exponent=1*)

Computes the unbiased estimator for the squared distance covariance between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Value of the unbiased estimator of the squared distance covariance.

Return type numpy scalar

See also:

distance_covariance(), *distance_covariance_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.u_distance_covariance_sqr(a, a) # doctest: +ELLIPSIS
42.6666666...
>>> dcor.u_distance_covariance_sqr(a, b) # doctest: +ELLIPSIS
-2.6666666...
>>> dcor.u_distance_covariance_sqr(b, b) # doctest: +ELLIPSIS
0.6666666...
>>> dcor.u_distance_covariance_sqr(a, b, exponent=0.5) # doctest: +ELLIPSIS
-0.2996598...
```

dcor.u_distance_correlation_sqr

u_distance_correlation_sqr(*x*, *y*, ***, *exponent=1*)

Computes the bias-corrected estimator for the squared distance correlation between two random vectors.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Value of the bias-corrected estimator of the squared distance correlation.

Return type numpy scalar

See also:

distance_correlation(), *distance_correlation_sqr()*

Notes

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.u_distance_correlation_sqr(a, a)
1.0
>>> dcor.u_distance_correlation_sqr(a, b)
-0.5
>>> dcor.u_distance_correlation_sqr(b, b)
1.0
>>> dcor.u_distance_correlation_sqr(a, b, exponent=0.5)
... # doctest: +ELLIPSIS
-0.4050479...
```

dcor.u_distance_stats_sqr

u_distance_stats_sqr(*x*, *y*, ***, *exponent=1*)

Computes the unbiased estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

Returns Squared distance covariance, squared distance correlation, squared distance variance of the first random vector and squared distance variance of the second random vector.

Return type Stats

See also:

u_distance_covariance_sqr(), *u_distance_correlation_sqr()*

Notes

It is less efficient to compute the statistics separately, rather than using this function, because some computations can be shared.

The algorithm uses the fast distance covariance algorithm proposed in [BHS16] when possible.

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.u_distance_stats_sqr(a, a) # doctest: +ELLIPSIS
...                     # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=42.6666666..., correlation_xy=1.0,
variance_x=42.6666666..., variance_y=42.6666666...)
>>> dcor.u_distance_stats_sqr(a, b) # doctest: +ELLIPSIS
...                     # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=-2.6666666..., correlation_xy=-0.5,
variance_x=42.6666666..., variance_y=0.6666666...)
>>> dcor.u_distance_stats_sqr(b, b) # doctest: +ELLIPSIS
...                     # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=0.6666666..., correlation_xy=1.0,
variance_x=0.6666666..., variance_y=0.6666666...)
>>> dcor.u_distance_stats_sqr(a, b, exponent=0.5) # doctest: +ELLIPSIS
...                                   # doctest: +NORMALIZE_WHITESPACE
Stats(covariance_xy=-0.2996598..., correlation_xy=-0.4050479...,
variance_x=0.8209855..., variance_y=0.6666666...)
```

1.4.1.3 Affinely invariant distance correlation

These functions compute the estimators for the affinely invariant distance correlation, a variant of distance correlation that is invariant by invertible affine transformations of the input parameters.

dcor.distance_correlation_af_inv_sqr(x, y)	Square of the affinely invariant distance correlation.
dcor.distance_correlation_af_inv(x, y)	Affinely invariant distance correlation.

dcor.distance_correlation_af_inv_sqr

distance_correlation_af_inv_sqr (x, y)

Square of the affinely invariant distance correlation.

Computes the estimator for the square of the affinely invariant distance correlation between two random vectors.

> **Warning:** The return value of this function is undefined when the covariance matrix of x or y is singular.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns Value of the estimator of the squared affinely invariant distance correlation.

Return type numpy scalar

See also:

distance_correlation(), u_distance_correlation()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 3, 2, 5],
...               [5, 7, 6, 8],
...               [9, 10, 11, 12],
...               [13, 15, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_correlation_af_inv_sqr(a, a)
1.0
>>> dcor.distance_correlation_af_inv_sqr(a, b) # doctest: +ELLIPSIS
0.5773502...
>>> dcor.distance_correlation_af_inv_sqr(b, b)
1.0
```

dcor.distance_correlation_af_inv

distance_correlation_af_inv (x, y)

Affinely invariant distance correlation.

Computes the estimator for the affinely invariant distance correlation between two random vectors.

Warning: The return value of this function is undefined when the covariance matrix of x or y is singular.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns Value of the estimator of the squared affinely invariant distance correlation.

Return type numpy scalar

See also:

distance_correlation(), u_distance_correlation()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 3, 2, 5],
...               [5, 7, 6, 8],
...               [9, 10, 11, 12],
...               [13, 15, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> dcor.distance_correlation_af_inv(a, a)
1.0
>>> dcor.distance_correlation_af_inv(a, b) # doctest: +ELLIPSIS
0.7598356...
>>> dcor.distance_correlation_af_inv(b, b)
1.0
```

1.4.1.4 Partial distance covariance and partial distance correlation

These functions compute the estimators for the partial distance covariance and partial distance correlation.

dcor.partial_distance_covariance(x, y, z)	Partial distance covariance estimator.
dcor.partial_distance_correlation(x, y, z)	Partial distance correlation estimator.

dcor.partial_distance_covariance

partial_distance_covariance (x, y, z)

Partial distance covariance estimator.

Compute the estimator for the partial distance covariance of the random vectors corresponding to x and y with respect to the random variable corresponding to z.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **z** (*array_like*) – Random vector with respect to which the partial distance covariance is computed. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns Value of the estimator of the partial distance covariance.

Return type numpy scalar

See also:

partial_distance_correlation()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1], [0], [0], [1]])
>>> c = np.array([[1, 3, 4],
...               [5, 7, 8],
...               [9, 11, 15],
...               [13, 15, 16]])
>>> dcor.partial_distance_covariance(a, a, c) # doctest: +ELLIPSIS
0.0024298...
>>> dcor.partial_distance_covariance(a, b, c)
0.0347030...
>>> dcor.partial_distance_covariance(b, b, c)
0.4956241...
```

dcor.partial_distance_correlation

partial_distance_correlation(x, y, z)

Partial distance correlation estimator.

Compute the estimator for the partial distance correlation of the random vectors corresponding to x and y with respect to the random variable corresponding to z.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **z** (*array_like*) – Random vector with respect to which the partial distance correlation is computed. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns Value of the estimator of the partial distance correlation.

Return type numpy scalar

See also:

partial_distance_covariance()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1], [1], [2], [2], [3]])
>>> b = np.array([[1], [2], [1], [2], [1]])
>>> c = np.array([[1], [2], [2], [1], [2]])
>>> dcor.partial_distance_correlation(a, a, c)
1.0
>>> dcor.partial_distance_correlation(a, b, c) # doctest: +ELLIPSIS
-0.5...
>>> dcor.partial_distance_correlation(b, b, c)
1.0
>>> dcor.partial_distance_correlation(a, c, c)
0.0
```

1.4.1.5 Energy distance

The following function is an estimator for the energy distance between two random vectors.

dcor.energy_distance(x, y, *[, exponent])	Computes the estimator for the energy distance of the random vectors corresponding to x and y.

dcor.energy_distance

energy_distance(*x*, *y*, *, *exponent=1*)

Computes the estimator for the energy distance of the random vectors corresponding to x and y. Both random vectors must have the same number of components.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$.

Returns Value of the estimator of the energy distance.

Return type numpy scalar

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
```

(continues on next page)

(continued from previous page)

```
...                [9, 10, 11, 12],
...                [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...                [0, 1, 1, 1],
...                [1, 1, 1, 1]])
>>> dcor.energy_distance(a, a)
0.0
>>> dcor.energy_distance(a, b) # doctest: +ELLIPSIS
20.5780594...
>>> dcor.energy_distance(b, b)
0.0
```

A different exponent for the Euclidean distance in the range $(0, 2)$ can be used:

```
>>> dcor.energy_distance(a, a, exponent=1.5)
0.0
>>> dcor.energy_distance(a, b, exponent=1.5)
... # doctest: +ELLIPSIS
99.7863955...
>>> dcor.energy_distance(b, b, exponent=1.5)
0.0
```

1.4.1.6 Homogeneity test

The following functions are used to test if random vectors have the same distribution.

dcor.homogeneity.energy_test_statistic(x, y, *)	Homogeneity statistic.
dcor.homogeneity.energy_test(*args[, ...])	Test of homogeneity based on the energy distance.

dcor.homogeneity.energy_test_statistic

energy_test_statistic (*x*, *y*, *, *exponent=1*)

Homogeneity statistic.

Computes the statistic for homogeneity based on the energy distance, for random vectors corresponding to x and y.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$.

Returns Value of the statistic for homogeneity based on the energy distance.

Return type numpy scalar

See also:

`energy_distance()`

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...               [0, 1, 1, 1],
...               [1, 1, 1, 1]])
>>> dcor.homogeneity.energy_test_statistic(a, a)
0.0
>>> dcor.homogeneity.energy_test_statistic(a, b) # doctest: +ELLIPSIS
35.2766732...
>>> dcor.homogeneity.energy_test_statistic(b, b)
0.0
```

dcor.homogeneity.energy_test

energy_test(**args*, *num_resamples=0*, *exponent=1*, *random_state=None*)

Test of homogeneity based on the energy distance.

Compute the test of homogeneity based on the energy distance, for an arbitrary number of random vectors.

The test is a permutation test where the null hypothesis is that all random vectors have the same distribution.

Parameters

- ***args** (*array_like*) – Random vectors. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **num_resamples** (*int*) – Number of permutations resamples to take in the permutation test.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$.
- **random_state** (*{None, int, array_like, numpy.random.RandomState}*) – Random state to generate the permutations.

Returns Results of the hypothesis test.

Return type HypothesisTest

See also:

`energy_distance()`

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...               [0, 1, 1, 1],
```

(continues on next page)

(continued from previous page)

```
...                [1, 1, 1, 1]])
>>> c = np.array([[1000, 0, 0, 1000],
...                [0, 1000, 1000, 1000],
...                [1000, 1000, 1000, 1000]])
>>> dcor.homogeneity.energy_test(a, a)
HypothesisTest(p_value=1.0, statistic=0.0)
>>> dcor.homogeneity.energy_test(a, b) # doctest: +ELLIPSIS
HypothesisTest(p_value=1.0, statistic=35.2766732...)
>>> dcor.homogeneity.energy_test(b, b)
HypothesisTest(p_value=1.0, statistic=0.0)
>>> dcor.homogeneity.energy_test(a, b, num_resamples=5, random_state=0)
HypothesisTest(p_value=0.1666666..., statistic=35.2766732...)
>>> dcor.homogeneity.energy_test(a, b, num_resamples=5, random_state=6)
HypothesisTest(p_value=0.3333333..., statistic=35.2766732...)
>>> dcor.homogeneity.energy_test(a, c, num_resamples=7, random_state=0)
HypothesisTest(p_value=0.125, statistic=4233.8935035...)
```

A different exponent for the Euclidean distance in the range $(0, 2)$ can be used:

```
>>> dcor.homogeneity.energy_test(a, b, exponent=1.5) # doctest: +ELLIPSIS
HypothesisTest(p_value=1.0, statistic=171.0623923...)
```

1.4.1.7 Independence test

The following functions are used to test if two random vectors are independent.

dcor.independence.distance_covariance_test(x, y, *)	Test of distance covariance independence.
dcor.independence.distance_correlation_t_statistic(x, y)	Transformation of the bias corrected version of distance correlation used in `distance_correlation_t_test()`.
dcor.independence.distance_correlation_t_test(x, y)	Test of independence for high dimension based on convergence to a Student t distribution.

dcor.independence.distance_covariance_test

distance_covariance_test(*x*, *y*, *, *num_resamples=0*, *exponent=1*, *random_state=None*)

Test of distance covariance independence.

Compute the test of independence based on the distance covariance, for two random vectors.

The test is a permutation test where the null hypothesis is that the two random vectors are independent.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **exponent** (*float*) – Exponent of the Euclidean distance, in the range $(0, 2)$. Equivalently, it is twice the Hurst parameter of fractional Brownian motion.

- **num_resamples** (*int*) – Number of permutations resamples to take in the permutation test.
- **random_state** (*{None, int, array_like, numpy.random.RandomState}*) – Random state to generate the permutations.

Returns Results of the hypothesis test.

Return type HypothesisTest

See also:

`distance_covariance()`

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...               [0, 1, 1, 1],
...               [1, 1, 1, 1],
...               [1, 1, 0, 1]])
>>> dcor.independence.distance_covariance_test(a, a)
HypothesisTest(p_value=1.0, statistic=208.0)
>>> dcor.independence.distance_covariance_test(a, b)
...                                       # doctest: +ELLIPSIS
HypothesisTest(p_value=1.0, statistic=11.75323056...)
>>> dcor.independence.distance_covariance_test(b, b)
HypothesisTest(p_value=1.0, statistic=1.3604610...)
>>> dcor.independence.distance_covariance_test(a, b,
... num_resamples=5, random_state=0)
HypothesisTest(p_value=0.5, statistic=11.7532305...)
>>> dcor.independence.distance_covariance_test(a, b,
... num_resamples=5, random_state=13)
HypothesisTest(p_value=0.3333333..., statistic=11.7532305...)
>>> dcor.independence.distance_covariance_test(a, a,
... num_resamples=7, random_state=0)
HypothesisTest(p_value=0.125, statistic=208.0)
```

dcor.independence.distance_correlation_t_statistic

distance_correlation_t_statistic(*x*, *y*)

Transformation of the bias corrected version of distance correlation used in *distance_correlation_t_test()*.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns T statistic.

Return type numpy scalar

See also:

distance_correlation_t_test()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...               [0, 1, 1, 1],
...               [1, 1, 1, 1],
...               [1, 1, 0, 1]])
>>> with np.errstate(divide='ignore'):
...     dcor.independence.distance_correlation_t_statistic(a, a)
inf
>>> dcor.independence.distance_correlation_t_statistic(a, b)
...                                       # doctest: +ELLIPSIS
-0.4430164...
>>> with np.errstate(divide='ignore'):
...     dcor.independence.distance_correlation_t_statistic(b, b)
inf
```

dcor.independence.distance_correlation_t_test

distance_correlation_t_test(*x*, *y*)

Test of independence for high dimension based on convergence to a Student t distribution. The null hypothesis is that the two random vectors are independent.

Parameters

- **x** (*array_like*) – First random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.
- **y** (*array_like*) – Second random vector. The columns correspond with the individual random variables while the rows are individual instances of the random vector.

Returns Results of the hypothesis test.

Return type HypothesisTest

See also:

distance_correlation_t_statistic()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
```

(continues on next page)

(continued from previous page)

```
...                [9, 10, 11, 12],
...                [13, 14, 15, 16]])
>>> b = np.array([[1, 0, 0, 1],
...                [0, 1, 1, 1],
...                [1, 1, 1, 1],
...                [1, 1, 0, 1]])
>>> with np.errstate(divide='ignore'):
...     dcor.independence.distance_correlation_t_test(a, a)
...                                          # doctest: +ELLIPSIS
HypothesisTest(p_value=0.0, statistic=inf)
>>> dcor.independence.distance_correlation_t_test(a, b)
...                                          # doctest: +ELLIPSIS
HypothesisTest(p_value=0.6327451..., statistic=-0.4430164...)
>>> with np.errstate(divide='ignore'):
...     dcor.independence.distance_correlation_t_test(b, b)
...                                          # doctest: +ELLIPSIS
HypothesisTest(p_value=0.0, statistic=inf)
```

1.4.1.8 Internal computations

These functions are used for computing the estimators of the squared distance covariance, and are also provided by this package.

dcor.double_centered(a, *[, out])	Return a copy of the matrix a which is double centered.
dcor.u_centered(a, *[, out])	Return a copy of the matrix a which is U-centered.
dcor.mean_product(a, b)	Average of the elements for an element-wise product of two matrices.
dcor.u_product(a, b)	Inner product in the Hilbert space of U-centered distance matrices.
dcor.u_projection(a)	Return the orthogonal projection function over a.
dcor.u_complementary_projection(a)	Return the orthogonal projection function over $a^{\perp}$.

dcor.double_centered

double_centered(*a*, *, *out=None*)

Return a copy of the matrix a which is double centered.

A matrix is double centered if both the sum of its columns and the sum of its rows are 0.

In order to do that, for every element its row and column averages are subtracted, and the total average is added.

Thus, if the element in the i-th row and j-th column of the original matrix a is $a_{i,j}$, then the new element will be

$$\tilde{a}_{i,j} = a_{i,j} - \frac{1}{N}\sum_{l=1}^{N} a_{il} - \frac{1}{N}\sum_{k=1}^{N} a_{kj} + \frac{1}{N^2}\sum_{k=1}^{N} a_{kj}.$$

Parameters

- **a** (*(N, N) array_like*) – Original matrix.
- **out** (*None or array_like*) – If not None, specifies where to return the resulting array. This array should allow non integer numbers.

Returns Double centered matrix.

Return type (N, N) ndarray

See also:

u_centered()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2], [3, 4]])
>>> dcor.double_centered(a)
array([[0., 0.],
       [0., 0.]])
>>> b = np.array([[1, 2, 3], [2, 4, 5], [3, 5, 6]])
>>> dcor.double_centered(b)
array([[ 0.44444444, -0.22222222, -0.22222222],
       [-0.22222222,  0.11111111,  0.11111111],
       [-0.22222222,  0.11111111,  0.11111111]])
>>> c = np.array([[1., 2., 3.], [2., 4., 5.], [3., 5., 6.]])
>>> dcor.double_centered(c, out=c)
array([[ 0.44444444, -0.22222222, -0.22222222],
       [-0.22222222,  0.11111111,  0.11111111],
       [-0.22222222,  0.11111111,  0.11111111]])
>>> c
array([[ 0.44444444, -0.22222222, -0.22222222],
       [-0.22222222,  0.11111111,  0.11111111],
       [-0.22222222,  0.11111111,  0.11111111]])
```

dcor.u_centered

u_centered(*a*, *, *out=None*)

Return a copy of the matrix a which is U-centered.

If the element of the i-th row and j-th column of the original matrix a is $a_{i,j}$, then the new element will be

$$\tilde{a}_{i,j} = \begin{cases} a_{i,j} - \frac{1}{n-2}\sum_{l=1}^{n} a_{il} - \frac{1}{n-2}\sum_{k=1}^{n} a_{kj} + \frac{1}{(n-1)(n-2)}\sum_{k=1}^{n} a_{kj}, & \text{if } i \neq j, \\ 0, & \text{if } i = j. \end{cases}$$

Parameters

- **a** (*(N, N) array_like*) – Original matrix.
- **out** (*None or array_like*) – If not None, specifies where to return the resulting array. This array should allow non integer numbers.

Returns U-centered matrix.

Return type (N, N) ndarray

See also:

double_centered()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3], [2, 4, 5], [3, 5, 6]])
>>> dcor.u_centered(a)
array([[ 0. ,  0.5, -1.5],
       [ 0.5,  0. , -4.5],
       [-1.5, -4.5,  0. ]])
>>> b = np.array([[1., 2., 3.], [2., 4., 5.], [3., 5., 6.]])
>>> dcor.u_centered(b, out=b)
array([[ 0. ,  0.5, -1.5],
       [ 0.5,  0. , -4.5],
       [-1.5, -4.5,  0. ]])
>>> b
array([[ 0. ,  0.5, -1.5],
       [ 0.5,  0. , -4.5],
       [-1.5, -4.5,  0. ]])
```

Note that when the matrix is 1x1 or 2x2, the formula performs a division by 0

```
>>> import warnings
>>> b = np.array([[1, 2], [3, 4]])
>>> with warnings.catch_warnings():
...     warnings.simplefilter("ignore")
...     dcor.u_centered(b)
array([[ 0., nan],
       [nan,  0.]])
```

dcor.mean_product

mean_product(*a*, *b*)

Average of the elements for an element-wise product of two matrices.

If the matrices are square it is

$$\frac{1}{n^2}\sum_{i,j=1}^{n} a_{i,j}b_{i,j}.$$

Parameters

- **a** (*array_like*) – First input array to be multiplied.
- **b** (*array_like*) – Second input array to be multiplied.

Returns Average of the product.

Return type numpy scalar

See also:

u_product()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 4], [1, 2, 4], [1, 2, 4]])
>>> b = np.array([[1, .5, .25], [1, .5, .25], [1, .5, .25]])
>>> dcor.mean_product(a, b)
1.0
>>> dcor.mean_product(a, a)
7.0
```

If the matrices involved are not square, but have the same dimensions, the average of the product is still well defined

```
>>> c = np.array([[1, 2], [1, 2], [1, 2]])
>>> dcor.mean_product(c, c)
2.5
```

dcor.u_product

u_product (a, b)

Inner product in the Hilbert space of U-centered distance matrices.

This inner product is defined as

$$\frac{1}{n(n-3)} \sum_{i,j=1}^{n} a_{i,j} b_{i,j}$$

Parameters

- **a** (*array_like*) – First input array to be multiplied.
- **b** (*array_like*) – Second input array to be multiplied.

Returns Inner product.

Return type numpy scalar

See also:

mean_product()

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[  0.,   3.,  11.,   6.],
...               [  3.,   0.,   8.,   3.],
...               [ 11.,   8.,   0.,   5.],
...               [  6.,   3.,   5.,   0.]])
>>> b = np.array([[  0.,  13.,  11.,   3.],
...               [ 13.,   0.,   2.,  10.],
...               [ 11.,   2.,   0.,   8.],
...               [  3.,  10.,   8.,   0.]])
>>> u_a = dcor.u_centered(a)
>>> u_a
array([[ 0., -2.,  1.,  1.],
       [-2.,  0.,  1.,  1.],
```

(continues on next page)

(continued from previous page)

```
       [ 1.,  1.,  0., -2.],
       [ 1.,  1., -2.,  0.]])
>>> u_b = dcor.u_centered(b)
>>> u_b
array([[ 0.        ,  2.66666667,  2.66666667, -5.33333333],
       [ 2.66666667,  0.        , -5.33333333,  2.66666667],
       [ 2.66666667, -5.33333333,  0.        ,  2.66666667],
       [-5.33333333,  2.66666667,  2.66666667,  0.        ]])
>>> dcor.u_product(u_a, u_a)
6.0
>>> dcor.u_product(u_a, u_b)
-8.0
```

Note that the formula is well defined as long as the matrices involved are square and have the same dimensions, even if they are not in the Hilbert space of U-centered distance matrices

```
>>> dcor.u_product(a, a)
132.0
```

Also the formula produces a division by 0 for 3x3 matrices

```
>>> import warnings
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> with warnings.catch_warnings():
...     warnings.simplefilter("ignore")
...     dcor.u_product(b, b)
inf
```

dcor.u_projection

u_projection(a)

Return the orthogonal projection function over a.

The function returned computes the orthogonal projection over a in the Hilbert space of U-centered distance matrices.

The projection of a matrix B over a matrix A is defined as

$$\operatorname{proj}_A(B) = \begin{cases} \frac{\langle A,B\rangle}{\langle A,A\rangle} A, & \text{if} \langle A, A\rangle \neq 0, \\ 0, & \text{if} \langle A, A\rangle = 0. \end{cases}$$

where $\langle \cdot\,, \cdot \rangle$ is the scalar product in the Hilbert space of U-centered distance matrices, given by the function *u_product()*.

Parameters **a** (*array_like*) – U-centered distance matrix.

Returns Function that receives a U-centered distance matrix and computes its orthogonal projection over a.

Return type callable

See also:

u_complementary_projection(), *u_centered()*

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[  0.,   3.,  11.,   6.],
...               [  3.,   0.,   8.,   3.],
...               [ 11.,   8.,   0.,   5.],
...               [  6.,   3.,   5.,   0.]])
>>> b = np.array([[  0.,  13.,  11.,   3.],
...               [ 13.,   0.,   2.,  10.],
...               [ 11.,   2.,   0.,   8.],
...               [  3.,  10.,   8.,   0.]])
>>> u_a = dcor.u_centered(a)
>>> u_a
array([[ 0., -2.,  1.,  1.],
       [-2.,  0.,  1.,  1.],
       [ 1.,  1.,  0., -2.],
       [ 1.,  1., -2.,  0.]])
>>> u_b = dcor.u_centered(b)
>>> u_b
array([[ 0.        ,  2.66666667,  2.66666667, -5.33333333],
       [ 2.66666667,  0.        , -5.33333333,  2.66666667],
       [ 2.66666667, -5.33333333,  0.        ,  2.66666667],
       [-5.33333333,  2.66666667,  2.66666667,  0.        ]])
>>> proj_a = dcor.u_projection(u_a)
>>> proj_a(u_a)
array([[ 0., -2.,  1.,  1.],
       [-2.,  0.,  1.,  1.],
       [ 1.,  1.,  0., -2.],
       [ 1.,  1., -2.,  0.]])
>>> proj_a(u_b)
array([[-0.        ,  2.66666667, -1.33333333, -1.33333333],
       [ 2.66666667, -0.        , -1.33333333, -1.33333333],
       [-1.33333333, -1.33333333, -0.        ,  2.66666667],
       [-1.33333333, -1.33333333,  2.66666667, -0.        ]])
```

The function gives the correct result if
$langle A, A$
$rangle = 0$.

```
>>> proj_null = dcor.u_projection(np.zeros((4, 4)))
>>> proj_null(u_a)
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

dcor.u_complementary_projection

u_complementary_projection(*a*)

Return the orthogonal projection function over $a^{\perp}$.

The function returned computes the orthogonal projection over $a^{\perp}$ (the complementary projection over a) in the Hilbert space of U-centered distance matrices.

The projection of a matrix B over a matrix $A^{\perp}$ is defined as

$$\text{proj}_{A^{\perp}}(B) = B - \text{proj}_{A}(B)$$

Parameters **a** (*array_like*) – U-centered distance matrix.

Returns Function that receives a U-centered distance matrices and computes its orthogonal projection over $a^{\perp}$.

Return type callable

See also:

u_projection(), *u_centered()*

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[  0.,   3.,  11.,   6.],
...               [  3.,   0.,   8.,   3.],
...               [ 11.,   8.,   0.,   5.],
...               [  6.,   3.,   5.,   0.]])
>>> b = np.array([[  0.,  13.,  11.,   3.],
...               [ 13.,   0.,   2.,  10.],
...               [ 11.,   2.,   0.,   8.],
...               [  3.,  10.,   8.,   0.]])
>>> u_a = dcor.u_centered(a)
>>> u_a
array([[ 0., -2.,  1.,  1.],
       [-2.,  0.,  1.,  1.],
       [ 1.,  1.,  0., -2.],
       [ 1.,  1., -2.,  0.]])
>>> u_b = dcor.u_centered(b)
>>> u_b
array([[ 0.        ,  2.66666667,  2.66666667, -5.33333333],
       [ 2.66666667,  0.        , -5.33333333,  2.66666667],
       [ 2.66666667, -5.33333333,  0.        ,  2.66666667],
       [-5.33333333,  2.66666667,  2.66666667,  0.        ]])
>>> proj_a = dcor.u_complementary_projection(u_a)
>>> proj_a(u_a)
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> proj_a(u_b)
array([[ 0.0000000e+00, -4.4408921e-16,  4.0000000e+00, -4.0000000e+00],
       [-4.4408921e-16,  0.0000000e+00, -4.0000000e+00,  4.0000000e+00],
       [ 4.0000000e+00, -4.0000000e+00,  0.0000000e+00, -4.4408921e-16],
       [-4.0000000e+00,  4.0000000e+00, -4.4408921e-16,  0.0000000e+00]])
>>> proj_null = dcor.u_complementary_projection(np.zeros((4, 4)))
>>> proj_null(u_a)
array([[ 0., -2.,  1.,  1.],
       [-2.,  0.,  1.,  1.],
       [ 1.,  1.,  0., -2.],
       [ 1.,  1., -2.,  0.]])
```

1.4.1.9 Compute distance matrices

These functions are used for computing distance matrices.

dcor.distances.pairwise_distances(x[, y, ...])	Pairwise distance between points.

dcor.distances.pairwise_distances

pairwise_distances (*x*, *y=None*, ***, *exponent=1*)

Pairwise distance between points.

Return the pairwise distance between points in two sets, or in the same set if only one set is passed.

Parameters

- **x** (*array_like*) – An $n \times m$ array of n observations in a m-dimensional space.
- **y** (*array_like*) – An $l \times m$ array of l observations in a m-dimensional space. If None, the distances will be computed between the points in x.
- **exponent** (*float*) – Exponent of the Euclidean distance.

Returns A $n \times l$ matrix where the (i, j)-th entry is the distance between $x[i]$ and $y[j]$.

Return type numpy ndarray

Examples

```
>>> import numpy as np
>>> import dcor
>>> a = np.array([[1, 2, 3, 4],
...               [5, 6, 7, 8],
...               [9, 10, 11, 12],
...               [13, 14, 15, 16]])
>>> b = np.array([[16, 15, 14, 13],
...               [12, 11, 10, 9],
...               [8, 7, 6, 5],
...               [4, 3, 2, 1]])
>>> dcor.distances.pairwise_distances(a)
array([[ 0.,  8., 16., 24.],
       [ 8.,  0.,  8., 16.],
       [16.,  8.,  0.,  8.],
       [24., 16.,  8.,  0.]])
>>> dcor.distances.pairwise_distances(a, b)
array([[24.41311123, 16.61324773,  9.16515139,  4.47213595],
       [16.61324773,  9.16515139,  4.47213595,  9.16515139],
       [ 9.16515139,  4.47213595,  9.16515139, 16.61324773],
       [ 4.47213595,  9.16515139, 16.61324773, 24.41311123]])
```

1.5 Internal documentation

1.5.1 List of modules

dcor._dcor_internals	Internal functions for distance covariance and correlation.
dcor._dcor	Distance correlation and covariance.
dcor._energy	Energy distance functions
dcor._pairwise	Functions to compute a pairwise dependency measure.
dcor._partial_dcor	Functions for computing partial distance covariance and correlation
dcor._utils	Utility functions
dcor.distances	Distance functions between sets of points.
dcor.homogeneity	Functions for testing homogeneity of several distributions.
dcor.independence	Functions for testing independence of several distributions.

1.5.1.1 dcor._dcor_internals

Internal functions for distance covariance and correlation.

The functions in this module are used for performing computations related with distance covariance and correlation.

Functions

`double_centered`(a, *[, out])	Return a copy of the matrix a which is double centered.
`mean_product`(a, b)	Average of the elements for an element-wise product of two matrices.
`u_centered`(a, *[, out])	Return a copy of the matrix a which is U-centered.
`u_complementary_projection`(a)	Return the orthogonal projection function over $a^{\perp}$.
`u_product`(a, b)	Inner product in the Hilbert space of U-centered distance matrices.
`u_projection`(a)	Return the orthogonal projection function over a.

1.5.1.2 dcor._dcor

Distance correlation and covariance.

This module contains functions to compute statistics related to the distance covariance and distance correlation [BSRB07].

References

Functions

`distance_correlation`(x, y, *[, exponent])	Computes the usual (biased) estimator for the distance correlation between two random vectors.

Continued on next page

Table 12 – continued from previous page

distance_correlation_af_inv(x, y)	Affinely invariant distance correlation.
distance_correlation_af_inv_sqr(x, y)	Square of the affinely invariant distance correlation.
distance_correlation_sqr(x, y, *[, exponent])	Computes the usual (biased) estimator for the squared distance correlation between two random vectors.
distance_covariance(x, y, *[, exponent])	Computes the usual (biased) estimator for the distance covariance between two random vectors.
distance_covariance_sqr(x, y, *[, exponent])	Computes the usual (biased) estimator for the squared distance covariance between two random vectors.
distance_stats(x, y, *[, exponent])	Computes the usual (biased) estimators for the distance covariance and distance correlation between two random vectors, and the individual distance variances.
distance_stats_sqr(x, y, *[, exponent])	Computes the usual (biased) estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.
u_distance_correlation_sqr(x, y, *[, exponent])	Computes the bias-corrected estimator for the squared distance correlation between two random vectors.
u_distance_covariance_sqr(x, y, *[, exponent])	Computes the unbiased estimator for the squared distance covariance between two random vectors.
u_distance_stats_sqr(x, y, *[, exponent])	Computes the unbiased estimators for the squared distance covariance and squared distance correlation between two random vectors, and the individual squared distance variances.

Classes

Stats(covariance_xy, correlation_xy, ...)	

1.5.1.3 dcor._energy

Energy distance functions

Functions

energy_distance(x, y, *[, exponent])	Computes the estimator for the energy distance of the random vectors corresponding to x and y.

1.5.1.4 dcor._pairwise

Functions to compute a pairwise dependency measure.

Functions

pairwise(function, x[, y, pool, is_symmetric])	Computes a dependency measure between each pair of elements.

1.5.1.5 dcor._partial_dcor

Functions for computing partial distance covariance and correlation

Functions

`partial_distance_correlation`(x, y, z)	Partial distance correlation estimator.
`partial_distance_covariance`(x, y, z)	Partial distance covariance estimator.

1.5.1.6 dcor._utils

Utility functions

1.5.1.7 dcor.distances

Distance functions between sets of points.

This module provide functions that compute the distance between one or two sets of points. The Scipy implementation is used when the conversion to a double precision floating point number will not cause loss of precision.

Functions

`pairwise_distances`(x[, y, exponent])	Pairwise distance between points.

1.5.1.8 dcor.homogeneity

Functions for testing homogeneity of several distributions.

The functions in this module provide methods for testing if the samples generated from two random vectors have the same distribution.

Functions

`energy_test`(*args[, num_resamples, ...])	Test of homogeneity based on the energy distance.
`energy_test_statistic`(x, y, *[, exponent])	Homogeneity statistic.

1.5.1.9 dcor.independence

Functions for testing independence of several distributions.

The functions in this module provide methods for testing if the samples generated from two random vectors are independent.

Functions

`distance_covariance_test`(x, y, *[, ...])	Test of distance covariance independence.

Continued on next page

Table 19 – continued from previous page

`partial_distance_covariance_test`(x, y, z, ...)	partial_distance_covariance_test(x, y, z, num_resamples=0, exponent=1, random_state=None)

dcor is developed on Github. Please report issues there as well.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[ASRB07] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 12 2007. URL: http://dx.doi.org/10.1214/009053607000000505, doi:10.1214/009053607000000505.

[CHS16] Xiaoming Huo and Gábor J. Székely. Fast computing for distance covariance. *Technometrics*, 58(4):435–447, 2016. URL: http://dx.doi.org/10.1080/00401706.2015.1054435, arXiv:http://dx.doi.org/10.1080/00401706.2015.1054435, doi:10.1080/00401706.2015.1054435.

[CSR13] Gábor J. Székely and Maria L. Rizzo. Energy statistics: a class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249 – 1272, 2013. URL: http://www.sciencedirect.com/science/article/pii/S0378375813000633, doi:10.1016/j.jspi.2013.03.018.

[CSR14] Gábor J. Székely and Maria L. Rizzo. Partial distance correlation with methods for dissimilarities. *The Annals of Statistics*, 42(6):2382–2412, 12 2014. URL: https://doi.org/10.1214/14-AOS1255, doi:10.1214/14-AOS1255.

[CSRB07] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 12 2007. URL: http://dx.doi.org/10.1214/009053607000000505, doi:10.1214/009053607000000505.

[BHS16] Xiaoming Huo and Gábor J. Székely. Fast computing for distance covariance. *Technometrics*, 58(4):435–447, 2016. URL: http://dx.doi.org/10.1080/00401706.2015.1054435, arXiv:http://dx.doi.org/10.1080/00401706.2015.1054435, doi:10.1080/00401706.2015.1054435.

[BSRB07] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 12 2007. URL: http://dx.doi.org/10.1214/009053607000000505, doi:10.1214/009053607000000505.

Python Module Index

d

Index