Description  Multiscale Graph Correlation (MGC) is a framework developed by Vogelstein et al. (2019) <DOI:10.7554/eLife.41690> that extends global correlation procedures to be multiscale; consequently, MGC tests typically require far fewer samples than existing methods for a wide variety of dependence structures and dimensionalities, while maintaining computational efficiency. Moreover, MGC provides a simple and elegant multiscale characterization of the potentially complex latent geometry underlying the relationship.
**Description**

ConnCompLabel is a 1 pass implementation of connected components labelling. Here it is applied to identify disjunt patches within a distribution.

The raster matrix can be a raster of class 'asc' (adehabitat package), 'RasterLayer' (raster package) or 'SpatialGridDataFrame' (sp package).

**Usage**

ConnCompLabel(mat)
Arguments

**mat**

is a binary matrix of data with 0 representing background and 1 representing environment of interest. NA values are acceptable. The matrix can be a raster of class 'asc' (this & adehabitat package), 'RasterLayer' (raster package) or 'SpatialGridDataFrame' (sp package).

Value

A matrix of the same dim and class of mat in which unique components (individual patches) are numbered 1:n with 0 remaining background value.

Author(s)

Jeremy VanDerWal <jjvanderwal@gmail.com>

References


Examples

```r
# define a simple binary matrix
tmat = matrix(c( 0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1),nr=10,byrow=TRUE)

ccl.mat = ConnCompLabel(tmat)
ccl.mat
image(t(ccl.mat[10:1,]),col=c('grey',rainbow(length(unique(ccl.mat))-1)))
```

Description

A function to simulate data with the same mean that spreads as class id increases.
Usage

discr.sims.cross(
  n,
  d,
  K,
  signal.scale = 10,
  non.scale = 1,
  mean.scale = 0,
  rotate = FALSE,
  class.equal = TRUE,
  ind = FALSE
)

Arguments

n         the number of samples.
d         the number of dimensions.
K         the number of classes in the dataset.
signal.scale         the scaling for the signal dimension. Defaults to 10.
on.scale         the scaling for the non-signal dimensions. Defaults to 1.
mean.scale         whether the magnitude of the difference in the means between the two classes. If a mean scale is requested, d should be at least > K.
rotate         whether to apply a random rotation. Defaults to TRUE.
class.equal         whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/sum(K) for k=1:K. Defaults to TRUE.
ind         whether to sample x and y independently. Defaults to FALSE.

Author(s)

Eric Bridgeford

Examples

library(mgc)
sim <- discr.sims.cross(100, 3, 2)

---

discr.sims.exp Discriminability Exponential Simulation

Description

A function to simulate multi-class data with an Exponential class-mean trend.
Usage

```r
discr.sims.exp(
  n,
  d,
  K,
  signal.scale = 1,
  signal.lshift = 1,
  non.scale = 1,
  rotate = FALSE,
  class.equal = TRUE,
  ind = FALSE
)
```

Arguments

- **n**: the number of samples.
- **d**: the number of dimensions. The first dimension will be the signal dimension; the remainders noise.
- **K**: the number of classes in the dataset.
- **signal.scale**: the scaling for the signal dimension. Defaults to 1.
- **signal.lshift**: the location shift for the signal dimension between the classes. Defaults to 1.
- **non.scale**: the scaling for the non-signal dimensions. Defaults to 1.
- **rotate**: whether to apply a random rotation. Defaults to TRUE.
- **class.equal**: whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/\text{sum}(K) for k=1:K. Defaults to TRUE.
- **ind**: whether to sample x and y independently. Defaults to FALSE.

Author(s)

Eric Bridgeford

Description

A function to simulate data with the same mean that spreads as class id increases.
discr.sims.linear

Usage

discr.sims.fat_tails(
    n,  # the number of samples.
    d,  # the number of dimensions.
    K,  # the number of classes in the dataset.
    signal.scale = 1,  # the scaling for the signal dimension. Defaults to 1.
    rotate = FALSE,  # whether to apply a random rotation. Defaults to TRUE.
    class.equal = TRUE,  # whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/sum(K) for k=1:K. Defaults to TRUE.
    ind = FALSE  # whether to sample x and y independently. Defaults to FALSE.
)

Arguments

n the number of samples.
d the number of dimensions.
K the number of classes in the dataset.
signal.scale the scaling for the signal dimension. Defaults to 1.
rotate whether to apply a random rotation. Defaults to TRUE.
class.equal whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/sum(K) for k=1:K. Defaults to TRUE.
ind whether to sample x and y independently. Defaults to FALSE.

Author(s)

Eric Bridgeford

Examples

library(mgc)
sim <- discr.sims.fat_tails(100, 3, 2)

Discriminability Linear Simulation

A function to simulate multi-class data with a linear class-mean trend. The signal dimension is the dimension carrying all of the between-class difference, and the non-signal dimensions are noise.
Usage

discr.sims.linear(
  n,
  d,
  K,
  signal.scale = 1,
  signal.lshift = 1,
  non.scale = 1,
  rotate = FALSE,
  class.equal = TRUE,
  ind = FALSE
)

Arguments

n the number of samples.
d the number of dimensions. The first dimension will be the signal dimension; the remainders noise.
K the number of classes in the dataset.
signal.scale the scaling for the signal dimension. Defaults to 1.
signal.lshift the location shift for the signal dimension between the classes. Defaults to 1.
non.scale the scaling for the non-signal dimensions. Defaults to 1.
rotate whether to apply a random rotation. Defaults to TRUE.
class.equal whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/sum(K) for k=1:K. Defaults to TRUE.
ind whether to sample x and y independently. Defaults to FALSE.

Author(s)

Eric Bridgeford

discr.sims.radial  Discriminability Radial Simulation

Description

A function to simulate data with the same mean with radial symmetry as class id increases.
Usage

discr.sims.radial(
  n,
  d,
  K,
  er.scale = 0.1,
  r = 1,
  class.equal = TRUE,
  ind = FALSE
)

Arguments

n  the number of samples.

d  the number of dimensions.

K  the number of classes in the dataset.

er.scale  the scaling for the error of the samples. Defaults to 0.1.

r  the radial spacing between each class. Defaults to 1.

class.equal  whether the number of samples/class should be equal, with each class having a prior of 1/K, or inequal, in which each class obtains a prior of k/sum(K) for k=1:K. Defaults to TRUE.

ind  whether to sample x and y independently. Defaults to FALSE.

Author(s)

Eric Bridgeford

Examples

library(mgc)
sim <- discr.sims.radial(100, 3, 2)

---

discr.stat  Discriminability Statistic

Description

A function for computing the discriminability from a distance matrix and a set of associated labels.
Usage

discr.stat(
    X,
    Y,
    is.dist = FALSE,
    dist.xfm = mgc.distance,
    dist.params = list(method = "euclidean"),
    dist.return = NULL,
    remove.isolates = TRUE
)

Arguments

X
is interpreted as:

\[ n \times d \] \textbf{data matrix} \quad X is a data matrix with \( n \) samples in \( d \) dimensions, if flag is.dist=False.

\[ n \times n \] \textbf{distance matrix} \quad X is a distance matrix. Use flag is.dist=TRUE.

Y
[\( n \)] a vector containing the sample ids for our \( n \) samples.

is.dist
a boolean indicating whether your \( X \) input is a distance matrix or not. Defaults to FALSE.

dist.xfm
if is.dist == FALSE, a distance function to transform \( X \). If a distance function is passed, it should accept an [\( n \times d \)] matrix of \( n \) samples in \( d \) dimensions and return a [\( n \times n \)] distance matrix, which can be either the default output, an item castable to a distance matrix, or . See mgc.distance for details.

dist.params
a list of trailing arguments to pass to the distance function specified in dist.xfm. Defaults to list(method='euclidean').

dist.return
the return argument for the specified dist.xfm containing the distance matrix. Defaults to FALSE.

is.null(dist.return)
use the return argument directly from dist.xfm as the distance matrix. Should be an object castable to a [\( n \times n \)] matrix. You can verify whether this is the case by looking at as.matrix(do.call(dist.xfm,list(X,<trailing_args>))

is.character(dist.return) | is.integer(dist.return) use dist.xfm[[dist.return]] as the distance matrix. Should be a [\( n \times n \)] matrix.

remove.isolates
remove isolated samples from the dataset. Isolated samples are samples with only one instance of their class appearing in the Y vector. Defaults to TRUE.

Value

A list containing the following:

- \texttt{discr} \quad the discriminability statistic.
- \texttt{rdf} \quad the rdfs for each sample.

Details

For more details see the help vignette: vignette("discriminability",package = "mgc")
discr.test.one_sample

Author(s)
Eric Bridgeford

References

Examples

```r
sim <- discr.sims.linear(100, 10, K=2)
X <- sim$X; Y <- sim$Y
discr.stat(X, Y)$discr
```

discr.test.one_sample  Discriminability One Sample Permutation Test

Description
A function that performs a one-sample test for whether the discriminability differs from random chance.

Usage

```r
discr.test.one_sample(
  X,
  Y,
  is.dist = FALSE,
  dist.xfm = mgc.distance,
  dist.params = list(method = "euclidean"),
  dist.return = NULL,
  remove.isolates = TRUE,
  nperm = 500,
  no_cores = 1
)
```

Arguments

- **X** is interpreted as:
  - a [n x d] data matrix X is a data matrix with n samples in d dimensions, if flag is.dist=FALSE.
  - a [n x n] distance matrix X is a distance matrix. Use flag is.dist=TRUE.

- **Y** [n] a vector containing the sample ids for our n samples.

- **is.dist** a boolean indicating whether your X input is a distance matrix or not. Defaults to FALSE.
if is.dist == FALSE, a distance function to transform X. If a distance function is passed, it should accept an \([n \times d]\) matrix of \(n\) samples in \(d\) dimensions and return a \([n \times n]\) distance matrix as the $D$ return argument. See mgc.distance for details.

dist.params a list of trailing arguments to pass to the distance function specified in dist.xfm. Defaults to list(method='euclidean').

dist.return the return argument for the specified dist.xfm containing the distance matrix. Defaults to FALSE.

is.null(dist.return) use the return argument directly from dist.xfm as the distance matrix. Should be a \([n \times n]\) matrix.

is.character(dist.return) | is.integer(dist.return) use dist.xfm[['dist.return']] as the distance matrix. Should be a \([n \times n]\) matrix.

remove.isolates remove isolated samples from the dataset. Isolated samples are samples with only one instance of their class appearing in the Y vector. Defaults to TRUE.

nperm the number of permutations to perform. Defaults to 500.

no_cores the number of cores to use for permutation test. Defaults to 1.

Value

A list containing the following:

- stat the discriminability of the data.
- null the discriminability scores under the null, computed via permutation.
- p.value the pvalue associated with the permutation test.

Details

Performs a test of whether an observed discriminability is significantly different from chance, as described in Bridgeford et al. (2019). With $\hat{D}_X$ the sample discriminability of $X$:

$$H_0 : D_X = D_0$$

and:

$$H_A : D_X > D_0$$

where $D_0$ is the discriminability that would be observed by random chance.

Author(s)

Eric Bridgeford

References

Examples

```r
## Not run:
require(mgc)
n = 100; d=5

# simulation with a large difference between the classes
# meaning they are more discriminable
sim <- discr.sims.linear(n=n, d=d, K=2, signal.lshift=10)
X <- sim$X; Y <- sim$Y

# p-value is small
discr.test.one_sample(X, Y)$p.value

## End(Not run)
```

discr.test.two_sample  Discriminability Two Sample Permutation Test

Description

A function that takes two sets of paired data and tests of whether or not the data is more, less, or non-equally discriminable between the set of paired data.

Usage

```r
discr.test.two_sample(
  X1,
  X2,
  Y,
  dist.xfm = mgc.distance,
  dist.params = list(method = "euclidian"),
  dist.return = NULL,
  remove.isolates = TRUE,
  nperm = 500,
  no_cores = 1,
  alt = "greater"
)
```

Arguments

- `X1` is interpreted as a `[n x d]` data matrix with `n` samples in `d` dimensions. Should NOT be a distance matrix.
- `X2` is interpreted as a `[n x d]` data matrix with `n` samples in `d` dimensions. Should NOT be a distance matrix.
- `Y` is a vector containing the sample ids for our `n` samples. Should be matched such that `Y[i]` is the corresponding label for `X1[i,]` and `X2[i,]`.
The `discr.test_two_sample` function performs a two-sample test for whether the discriminability is different for that of one dataset vs another, as described in Bridgeford et al. (2019). With $\hat{D}_{X_1}$, the sample discriminability of one approach, and $\hat{D}_{X_2}$ the sample discriminability of another approach:

$$H_0 : \hat{D}_{X_1} = \hat{D}_{X_2}$$

and:

$$H_A : \hat{D}_{X_1} > \hat{D}_{X_2}$$

Also implemented are tests of $<$ and $\neq$.

**Author(s)**

Eric Bridgeford
References


Examples

```r
## Not run:
require(mgc)
require(MASS)

n = 100; d=5

# generate two subjects truths; true difference btwn
# subject 1 (column 1) and subject 2 (column 2)
mus <- cbind(c(0, 0), c(1, 1))
Sigma <- diag(2)  # dimensions are independent

# first dataset X1 contains less noise than X2
X1 <- do.call(rbind, lapply(1:dim(mus)[2],
                      function(k) {mvrnorm(n=50, mus[,k], 0.5*Sigma)})))
X2 <- do.call(rbind, lapply(1:dim(mus)[2],
                      function(k) {mvrnorm(n=50, mus[,k], 2*Sigma)})))
Y <- do.call(c, lapply(1:2, function(i) rep(i, 50)))

# X1 should be more discriminable, as less noise
discr.test.two_sample(X1, X2, Y, alt="greater")$p.value  # p-value is small

## End(Not run)
```

description

Transform the distance matrices, with column-wise ranking if needed.

Usage

```r
mgc.dist.xfm(X, Y, option = "mgc", optionRk = TRUE)
```

Arguments

- **X**  
  
  [nxn] is a distance matrix

- **Y**  
  
  [nxn] is a second distance matrix

- **option**  
  
  is a string that specifies which global correlation to build up-on. Defaults to mgc.
  
  'mgc' use the MGC global correlation.
  
  'dcor' use the dcor global correlation.
`mantel` use the mantel global correlation.

`rank` use the rank global correlation.

`optionRk` is a string that specifies whether ranking within column is computed or not. If `option='rank'`, ranking will be performed regardless of the value specified by `optionRk`. Defaults to `TRUE`.

**Value**

A list containing the following:

A

[n x n] the centered distance matrix for X.

B

[n x n] the centered distance matrix for Y.

RX

[n x n] the column-rank matrices of X.

RY

[n x n] the column-rank matrices of Y.

**Author(s)**

C. Shen

**Examples**

```r
library(mgc)

n=200; d=2
data <- mgc.sims.linear(n, d)
Dx <- as.matrix(dist(data$X), nrow=n); Dy <- as.matrix(dist(data$Y), nrow=n)
dt <- mgc.dist.xfm(Dx, Dy)
```

**Description**

A function that returns a distance matrix given a collection of observations.

**Usage**

```r
mgc.distance(X, method = "euclidean")
```

**Arguments**

X

[n x d] a data matrix for d samples of d variables.

method
the method for computing distances. Defaults to 'euclidean'. See `dist` for details. Also includes a 'ohe' option, which one-hot-encodes the matrix when computing distances.
**Value**

a \([n \times n]\) distance matrix indicating the pairwise distances between all samples passed in.

**Author(s)**

Eric Bridgeford

---

**mgc.ksample**  
*MGC K Sample Testing*

---

**Description**

MGC K Sample Testing provides a wrapper for MGC Sample testing under the constraint that the Ys here are categorical labels with K possible sample ids. This function uses a 0-1 loss for the Ys (one-hot-encoding).

**Usage**

```r
gc.ksample(X, Y, mgc.opts = list(), ...)
```

**Arguments**

- **X**
  - is interpreted as:
    - a \([n \times d]\) **data matrix** X is a data matrix with \(n\) samples in \(d\) dimensions, if flag `is.dist.X=FALSE`.
    - a \([n \times n]\) **distance matrix** X is a distance matrix. Use flag `is.dist.X=TRUE`.

- **Y**
  - \([n]\) the labels of the samples with \(K\) unique labels.

- **mgc.opts**
  - Arguments to pass to MGC, as a named list. See `mgc.test` for details. Do not pass arguments for `is.dist.Y, dist.xfm.Y, dist.params.Y, nor dist.return.Y, as they will be ignored.

- **...**
  - trailing args.

**Value**

A list containing the following:

- **p.value** P-value of MGC
- **stat** is the sample MGC statistic within \([-1, 1]\)
- **pLocalCorr** P-value of the local correlations by double matrix index
- **localCorr** the local correlations
- **optimalScale** the optimal scale identified by MGC

**Author(s)**

Eric Bridgeford
References


Examples

```r
## Not run:
library(mgc)
library(MASS)

n = 100; d = 2
# simulate 100 samples, where first 50 have mean [0,0] and second 50 have mean [1,1]
Y <- c(replicate(n/2, 0), replicate(n/2, 1))
X <- do.call(rbind, lapply(Y, function(y) {
    return(rnorm(d) + y)
}))
# p value is small
mgc.ksample(X, Y, mgc.opts=list(nperm=100))$p.value

## End(Not run)
```

---

`mgc.localcorr`  \hspace{1cm} **MGC Local Correlations**

**Description**

Compute all local correlation coefficients in $O(n^2 \log n)$

**Usage**

```r
mgc.localcorr(
  X,
  Y,
  is.dist.X = FALSE,
  dist.xfm.X = mgc.distance,
  dist.params.X = list(method = "euclidean"),
  dist.return.X = NULL,
  is.dist.Y = FALSE,
  dist.xfm.Y = mgc.distance,
  dist.params.Y = list(method = "euclidean"),
  dist.return.Y = NULL,
  option = "mgc"
)
```
Arguments

\(X\) is interpreted as:

a \([n \times d]\) data matrix \(X\) is a data matrix with \(n\) samples in \(d\) dimensions, if flag is.dist.X=FALSE.

a \([n \times n]\) distance matrix \(X\) is a distance matrix. Use flag is.dist.X=TRUE.

\(Y\) is interpreted as:

a \([n \times d]\) data matrix \(Y\) is a data matrix with \(n\) samples in \(d\) dimensions, if flag is.dist.Y=FALSE.

a \([n \times n]\) distance matrix \(Y\) is a distance matrix. Use flag is.dist.Y=TRUE.

is.dist.X a boolean indicating whether your \(X\) input is a distance matrix or not. Defaults to FALSE.

dist.xfm.X if is.dist == FALSE, a distance function to transform \(X\). If a distance function is passed, it should accept an \([n \times d]\) matrix of \(n\) samples in \(d\) dimensions and return a \([n \times n]\) distance matrix as the $D$ return argument. See mgc.distance for details.

dist.params.X a list of trailing arguments to pass to the distance function specified in dist.xfm.X. Defaults to list(method='euclidean').

dist.return.X the return argument for the specified dist.xfm.X containing the distance matrix. Defaults to FALSE.

is.null(dist.return) use the return argument directly from dist.xfm as the distance matrix. Should be a \([n \times n]\) matrix.

is.character(dist.return) | is.integer(dist.return) use dist.xfm.X[[dist.return]] as the distance matrix. Should be a \([n \times n]\) matrix.

is.dist.Y a boolean indicating whether your \(Y\) input is a distance matrix or not. Defaults to FALSE.

dist.xfm.Y if is.dist == FALSE, a distance function to transform \(Y\). If a distance function is passed, it should accept an \([n \times d]\) matrix of \(n\) samples in \(d\) dimensions and return a \([n \times n]\) distance matrix as the dist.return.Y return argument. See mgc.distance for details.

dist.params.Y a list of trailing arguments to pass to the distance function specified in dist.xfm.Y. Defaults to list(method='euclidean').

dist.return.Y the return argument for the specified dist.xfm.Y containing the distance matrix. Defaults to FALSE.

is.null(dist.return) use the return argument directly from dist.xfm.Y(Y) as the distance matrix. Should be a \([n \times n]\) matrix.

is.character(dist.return) | is.integer(dist.return) use dist.xfm.Y(Y)[[dist.return]] as the distance matrix. Should be a \([n \times n]\) matrix.

option is a string that specifies which global correlation to build up-on. Defaults to 'mgc'.

'mgc' use the MGC global correlation.
'dcor' use the dcor global correlation.
'mantel' use the mantel global correlation.
'rank' use the rank global correlation.
Value

A list contains the following:

- **corr**: consists of all local correlations within \([-1,1]\) by double matrix index
- **varX**: contains all local variances for X.
- **varY**: contains all local variances for X.

**Author(s)**

C. Shen

**Examples**

```r
library(mgc)

n=200; d=2
data <- mgc.sims.linear(n, d)
lcor <- mgc.localcorr(data$X, data$Y)
```

---

**mgc.localcorr.driver**  
*Driver for MGC Local Correlations*

**Description**

Driver for MGC Local Correlations

**Usage**

```r
mgc.localcorr.driver(DX, DY, option = "mgc")
```

**Arguments**

- **DX**: the first distance matrix.
- **DY**: the second distance matrix.
- **option**: is a string that specifies which global correlation to build up-on. Defaults to 'mgc'.
  - 'mgc' use the MGC global correlation.
  - 'dcor' use the dcor global correlation.
  - 'mantel' use the mantel global correlation.
  - 'rank' use the rank global correlation.
Value
A list contains the following:

corr consists of all local correlations within [-1,1] by double matrix index
varX contains all local variances for X.
varY contains all local variances for X.

Author(s)
C. Shen

---

mgc.sims.2ball Sample from Unit 2-Ball

Description
Sample from the 2-ball in d-dimensions.

Usage
mgc.sims.2ball(n, d, r = 1, cov.scale = 0)

Arguments

n the number of samples.
d the number of dimensions.
r the radius of the 2-ball. Defaults to 1.
cov.scale if desired, sample from 2-ball with error sigma. Defaults to NaN, which has no noise.

Value
the points sampled from the ball, as a [n,d] array.

Author(s)
Eric Bridgeford

Examples

library(mgc)
# sample 100 points from 3-d 2-ball with radius 2
X <- mgc.sims.2ball(100, 3, 2)
Description

Sample from the 2-sphere in d-dimensions.

Usage

mgc.sims.2sphere(n, d, r, cov.scale = 0)

Arguments

- `n`  
  the number of samples.
- `d`  
  the number of dimensions.
- `r`  
  the radius of the 2-ball. Defaults to 1.
- `cov.scale`  
  if desired, sample from 2-ball with error sigma. Defaults to 0, which has no noise.

Value

the points sampled from the sphere, as a \([n, d]\) array.

Author(s)

Eric Bridgeford

Examples

library(mgc)
# sample 100 points from 3-d 2-sphere with radius 2
X <- mgc.sims.2sphere(100, 3, 2)

Description

A function for Generating a cubic simulation.
Usage

```r
gc.sims.cubic(
  n,
  d,
  eps = 80,
  ind = FALSE,
  a = -1,
  b = 1,
  c.coef = c(-12, 48, 128),
  s = 1/3
)
```

Arguments

- **n**: the number of samples for the simulation.
- **d**: the number of dimensions for the simulation setting.
- **eps**: the noise level for the simulation. Defaults to 80.
- **ind**: whether to sample x and y independently. Defaults to FALSE.
- **a**: the lower limit for the range of the data matrix. Defaults to -1.
- **b**: the upper limit for the range of the data matrix. Defaults to 1.
- **c.coef**: the coefficients for the cubic function, where the first value is the first order coefficient, the second value the quadratic coefficient, and the third the cubic coefficient. Defaults to c(-12, 48, 128).
- **s**: the scaling for the center of the cubic. Defaults to 1/3.

Value

A list containing the following:

- **X**: \([n, d]\) the data matrix with \(n\) samples in \(d\) dimensions.
- **Y**: \([n]\) the response array.

Details

Given: \(w_i = \frac{1}{d}\) is a weight-vector that scales with the dimensionality. Simulates \(n\) points from \(\text{Linear}(X, Y) \in \mathbb{R}^d \times \mathbb{R}\), where:

\[
X \sim U(a, b)^d
\]

\[
Y = c_3 \left( w^T X - s \right)^3 + c_2 \left( w^T X - s \right)^2 + c_1 \left( w^T X - s \right) + \kappa \epsilon
\]

and \(\kappa = 1\) if \(d = 1\), and 0 otherwise controls the noise for higher dimensions.

Author(s)

Eric Bridgeford
Examples

```r
library(mgc)
result <- mgc.sims.cubic(n=100, d=10) # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```

mgc.sims.exp

### Exponential Simulation

#### Description

A function for Generating an exponential simulation.

#### Usage

```r
mgc.sims.exp(n, d, eps = 10, ind = FALSE, a = 0, b = 3)
```

#### Arguments

- **n**: the number of samples for the simulation.
- **d**: the number of dimensions for the simulation setting.
- **eps**: the noise level for the simulation. Defaults to 10.
- **ind**: whether to sample x and y independently. Defaults to FALSE.
- **a**: the lower limit for the range of the data matrix. Defaults to 0.
- **b**: the upper limit for the range of the data matrix. Defaults to 3.

#### Value

A list containing the following:

- **X**: [n, d] the data matrix with n samples in d dimensions.
- **Y**: [n] the response array.

#### Details

Given: \( w_i = \frac{1}{d} \) is a weight-vector that scales with the dimensionality. Simulates \( n \) points from

\[
\begin{align*}
X & \sim U(a, b)^d \\
Y & = e^{w^T X} + \kappa \epsilon
\end{align*}
\]

and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions.

#### Author(s)

Eric Bridgeford
Examples

```r
library(mgc)
result <- mgc.sims.exp(n=100, d=10)  # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```

---

**mgc.sims.joint**  
*Joint Normal Simulation*

**Description**

A function for generating a joint-normal simulation.

**Usage**

```r
mgc.sims.joint(n, d, eps = 0.5)
```

**Arguments**

- `n`: the number of samples for the simulation.
- `d`: the number of dimensions for the simulation setting.
- `eps`: the noise level for the simulation. Defaults to 0.5.

**Value**

A list containing the following:

- `X [n,d]`: the data matrix with `n` samples in `d` dimensions.

**Details**

Given: \( \rho = \frac{1}{2}d \), \( I_d \) is the identity matrix of size \( d \times d \), \( J_d \) is the matrix of ones of size \( d \times d \). Simulates \( n \) points from Joint − Normal \( (X, Y) \in \mathbb{R}^d \times \mathbb{R}^d \), where:

\[
(X, Y) \sim N(0, \Sigma)
\]

\[
\Sigma = [I_d; \rho J_d; \rho J_d; (1 + \varepsilon \kappa) I_d]
\]

and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions. For more details see the help vignette: `vignette("sims", package = "mgc")`

**Author(s)**

Eric Bridgeford

**Examples**

```r
library(mgc)
result <- mgc.sims.joint(n=100, d=10)  # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```
mgc.sims.linear

Linear Simulation

Description

A function for Generating a linear simulation.

Usage

mgc.sims.linear(n, d, eps = 1, ind = FALSE, a = -1, b = 1)

Arguments

n
the number of samples for the simulation.

d
the number of dimensions for the simulation setting.

eps
the noise level for the simulation. Defaults to 1.

ind
whether to sample x and y independently. Defaults to FALSE.

a
the lower limit for the range of the data matrix. Defaults to -1.

b
the upper limit for the range of the data matrix. Defaults to 1.

Value

a list containing the following:

X
[n, d] the data matrix with n samples in d dimensions.

Y
[n] the response array.

Details

Given: \( w_i = \frac{1}{i} \) is a weight-vector that scales with the dimensionality. Simulates \( n \) points from \( \text{Linear}(X, Y) \in \mathbb{R}^d \times \mathbb{R} \), where:

\[
X \sim U(a, b)^d \\
Y = w^T X + \kappa \epsilon
\]

and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions.

Author(s)

Eric Bridgeford

Examples

library(mgc)
result <- mgc.sims.linear(n=100, d=10)  # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
Description
A function for generating a quadratic simulation.

Usage
mgc.sims.quad(n, d, eps = 0.5, ind = FALSE, a = -1, b = 1)

Arguments
- `n`: the number of samples for the simulation.
- `d`: the number of dimensions for the simulation setting.
- `eps`: the noise level for the simulation. Defaults to 0.5.
- `ind`: whether to sample x and y independently. Defaults to FALSE.
- `a`: the lower limit for the data matrix. Defaults to -1.
- `b`: the upper limit for the data matrix. Defaults to 1.

Value
a list containing the following:
- `X`: [n, d] the data matrix with n samples in d dimensions.
- `Y`: [n] the response array.

Details
Given: \( w_i = \frac{1}{d} \) is a weight-vector that scales with the dimensionality. Simulates n points from \( \text{Quadratic}(X, Y) \in \mathbb{R}^d \times \mathbb{R} \) where:
\[
X \sim U(a, b)^d
\]
\[
Y = (w^T X)^2 + \kappa \epsilon \mathcal{N}(0, 1)
\]
and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions.
For more details see the help vignette: vignette("sims", package = "mgc")

Author(s)
Eric Bridgeford

Examples
library(mgc)
result <- mgc.sims.quad(n=100, d=10) # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
Description

A function for Generating a spiral simulation.

Usage

```r
mgc.sims.spiral(n, d, eps = 0.4, a = 0, b = 5)
```

Arguments

- `n` the number of samples for the simulation.
- `d` the number of dimensions for the simulation setting.
- `eps` the noise level for the simulation. Defaults to 0.5.
- `a` the lower limit for the data matrix. Defaults -1.
- `b` the upper limit for the data matrix. Defaults to 1.

Value

A list containing the following:

- `X` \([n,d]\) the data matrix with \(n\) samples in \(d\) dimensions.
- `Y` \([n]\) the response array.

Details

Given: \(U \sim U(a, b)\) a random variable. Simulates \(n\) points from \(Spiral(X, Y) \in \mathbb{R}^d \times \mathbb{R}\) where:

\[
X_i = \begin{cases} 
U \cos(\pi U) & \text{if } i = d, \\
U \sin(\pi U) \cos(\pi U) & \text{otherwise}
\end{cases}
\]

\[Y = U \sin(\pi U) + \epsilon pN(0, 1)\]

For more details see the help vignette: `vignette("sims", package = "mgc")`

Author(s)

Eric Bridgeford

Examples

```r
library(mgc)
result <- mgc.sims.spiral(n=100, d=10) # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```
**mgc.sims.step**  

**Step Function Simulation**

**Description**

A function for generating a step function simulation.

**Usage**

```r
mgc.sims.step(n, d, eps = 1, ind = FALSE, a = -1, b = 1)
```

**Arguments**

- `n`: the number of samples for the simulation.
- `d`: the number of dimensions for the simulation setting.
- `eps`: the noise level for the simulation. Defaults to 1.
- `ind`: whether to sample x and y independently. Defaults to `FALSE`.
- `a`: the lower limit for the data matrix. Defaults to -1.
- `b`: the upper limit for the data matrix. Defaults to -1.

**Value**

A list containing the following:

- `X`: `[n,d]` the data matrix with `n` samples in `d` dimensions.
- `Y`: `[n]` the response array.

**Details**

Given: \( w_i = \frac{1}{d} \) is a weight-vector that scales with the dimensionality. Simulates \( n \) points from \( \text{Step}(X, Y) \in \mathbb{R}^d \times \mathbb{R} \) where:

\[
X \sim U(a, b)^d
\]

\[
Y = I\{w^T X > 0\} + \kappa \epsilon \mathcal{N}(0, 1)
\]

and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions.

For more details see the help vignette: `vignette("sims",package = "mgc")`

**Author(s)**

Eric Bridgeford

**Examples**

```r
library(mgc)
result <- mgc.sims.step(n=100, d=10)  # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```
mgc.sims.ubern  Uncorrelated Bernoulli Simulation

Description

A function for generating an uncorrelated bernoulli simulation.

Usage

mgc.sims.ubern(n, d, eps = 0.5, p = 0.5)

Arguments

n  the number of samples for the simulation.
d  the number of dimensions for the simulation setting.
eps  the noise level for the simulation. Defaults to 0.5.
p  the bernoulli probability.

Value

a list containing the following:

X  [n, d] the data matrix with n samples in d dimensions.
Y  [n] the response array.

Details

Given:  \( w_i = \frac{1}{d} \) is a weight-vector that scales with the dimensionality. Simulates \( n \) points from  \( W_{shape}(X, Y) \in \mathbb{R}^d \times \mathbb{R} \) where:

\[
U \sim Bern(p) \\
X \sim Bern(p)^d + \epsilon N(0, I_d) \\
Y = (2U - 1)w^TX + \epsilon N(0, 1)
\]

For more details see the help vignette: vignette("sims",package = "mgc")

Author(s)

Eric Bridgeford

Examples

library(mgc)
result <- mgc.sims.ubern(n=100, d=10)  # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
Description

A function for generating a W-shaped simulation.

Usage

\texttt{mgc.sims.wshape(n, d, eps = 0.5, ind = FALSE, a = -1, b = 1)}

Arguments

\begin{itemize}
  \item \texttt{n} the number of samples for the simulation.
  \item \texttt{d} the number of dimensions for the simulation setting.
  \item \texttt{eps} the noise level for the simulation. Defaults to 0.5.
  \item \texttt{ind} whether to sample \texttt{x} and \texttt{y} independently. Defaults to \texttt{FALSE}.
  \item \texttt{a} the lower limit for the data matrix. Defaults to \texttt{-1}.
  \item \texttt{b} the upper limit for the data matrix. Defaults to \texttt{1}.
\end{itemize}

Value

a list containing the following:
\begin{itemize}
  \item \texttt{X} \texttt{[n,d]} the data matrix with \texttt{n} samples in \texttt{d} dimensions.
  \item \texttt{Y} \texttt{[n]} the response array.
\end{itemize}

Details

Given: \( w_i = \frac{1}{i} \) is a weight-vector that scales with the dimensionality. Simulates \( n \) points from \( W - \text{shape}(X, Y) \in \mathbb{R}^d \times \mathbb{R} \) where:

\begin{align*}
  U & \sim U(a, b)^d \\
  X & \sim U(a, b)^d \\
  Y & = \left[ (w^T X)^2 - \frac{1}{2} + \frac{w^T U}{500} \right] + \kappa \epsilon N(0, 1)
\end{align*}

and \( \kappa = 1 \) if \( d = 1 \), and 0 otherwise controls the noise for higher dimensions.

For more details see the help vignette: \texttt{vignette("sims", package = "mgc")}

Author(s)

Eric Bridgeford
Examples

```r
library(mgc)
result <- mgc.sims.wshape(n=100, d=10) # simulate 100 samples in 10 dimensions
X <- result$X; Y <- result$Y
```

### mgc.stat

**MGC Test**

#### Description

The main function that computes the MGC measure between two datasets: It first computes all local correlations, then use the maximal statistic among all local correlations based on thresholding.

#### Usage

```r
mgc.stat(
  X,
  Y,
  is.dist.X = FALSE,
  dist.xfm.X = mgc.distance,
  dist.params.X = list(method = "euclidean"),
  dist.return.X = NULL,
  is.dist.Y = FALSE,
  dist.xfm.Y = mgc.distance,
  dist.params.Y = list(method = "euclidean"),
  dist.return.Y = NULL,
  option = "mgc"
)
```

#### Arguments

- **X**
  
  is interpreted as:

  - a **[n x d]** data matrix: X is a data matrix with n samples in d dimensions, if flag **is.dist.X**=FALSE.
  
  - a **[n x n]** distance matrix: X is a distance matrix. Use flag **is.dist.X**=TRUE.

- **Y**
  
  is interpreted as:

  - a **[n x d]** data matrix: Y is a data matrix with n samples in d dimensions, if flag **is.dist.Y**=FALSE.
  
  - a **[n x n]** distance matrix: Y is a distance matrix. Use flag **is.dist.Y**=TRUE.

- **is.dist.X**
  
  a boolean indicating whether your X input is a distance matrix or not. Defaults to FALSE.

- **dist.xfm.X**
  
  if **is.dist** == FALSE, a distance function to transform X. If a distance function is passed, it should accept an [n x d] matrix of n samples in d dimensions and return a [n x n] distance matrix as the $D return argument. See mgc.distance for details.
dist.params.X  a list of trailing arguments to pass to the distance function specified in dist.xfm.X. 
Defaults to list(method='euclidean').

dist.return.X  the return argument for the specified dist.xfm.X containing the distance matrix. 
Defaults to FALSE.
   is.null(dist.return)  use the return argument directly from dist.xfm.X as the 
distance matrix. Should be a [n x n] matrix.
   is.character(dist.return) | is.integer(dist.return)  use dist.xfm.X[[dist.return]]
   as the distance matrix. Should be a [n x n] matrix.

is.dist.Y  a boolean indicating whether your Y input is a distance matrix or not. Defaults 
to FALSE.

dist.xfm.Y  if is.dist == FALSE, a distance function to transform Y. If a distance function 
is passed, it should accept an [n x d] matrix of n samples in d dimensions and 
return a [n x n] distance matrix as the dist.return.Y return argument. See 
mgc.distance for details.

dist.params.Y  a list of trailing arguments to pass to the distance function specified in dist.xfm.Y.
Defaults to list(method='euclidean').

dist.return.Y  the return argument for the specified dist.xfm.Y containing the distance matrix. 
Defaults to FALSE.
   is.null(dist.return)  use the return argument directly from dist.xfm.Y(Y) 
as the distance matrix. Should be a [n x n] matrix.
   is.character(dist.return) | is.integer(dist.return)  use dist.xfm.Y(Y)[[dist.return]]
   as the distance matrix. Should be a [n x n] matrix.

option  is a string that specifies which global correlation to build up-on. Defaults to 
'mgc'.
   'mgc'  use the MGC global correlation.
   'dcor'  use the decor global correlation.
   'mantel'  use the mantel global correlation.
   'rank'  use the rank global correlation.

Value
A list containing the following:

stat  is the sample MGC statistic within [-1,1]
localCorr  the local correlations
optimalScale  the optimal scale identified by MGC
option  specifies which global correlation was used

Author(s)
C. Shen and Eric Bridgeford

References
Joshua T. Vogelstein, et al. "Discovering and deciphering relationships across disparate data modal-
Examples

library(mgc)

n=200; d=2
data <- mgc.sims.linear(n, d)
mgc.stat.res <- mgc.stat(data$X, data$Y)

mgc.test

MGC Permutation Test

Description

Test of Dependence using MGC Approach.

Usage

mgc.test(
  X,
  Y,
  is.dist.X = FALSE,
  dist.xfm.X = mgc.distance,
  dist.params.X = list(method = "euclidean"),
  dist.return.X = NULL,
  is.dist.Y = FALSE,
  dist.xfm.Y = mgc.distance,
  dist.params.Y = list(method = "euclidean"),
  dist.return.Y = NULL,
  nperm = 1000,
  option = "mgc",
  no_cores = 1
)

Arguments

X is interpreted as:

\textbf{a \,[n \times d] data matrix} \ X is a data matrix with \ n \ samples in \ d \ dimensions, if flag \ is.dist.X=FALSE.

\textbf{a \,[n \times n] distance matrix} \ X is a distance matrix. Use flag \ is.dist.X=TRUE.

Y is interpreted as:

\textbf{a \,[n \times d] data matrix} \ Y is a data matrix with \ n \ samples in \ d \ dimensions, if flag \ is.dist.Y=FALSE.

\textbf{a \,[n \times n] distance matrix} \ Y is a distance matrix. Use flag \ is.dist.Y=TRUE.

\textbf{is.dist.X} \ a boolean indicating whether your \ X \ input is a distance matrix or not. Defaults to FALSE.
if is.dist == FALSE, a distance function to transform \( X \). If a distance function is passed, it should accept an \([n \times d]\) matrix of \( n \) samples in \( d \) dimensions and return a \([n \times n]\) distance matrix as the $D$ return argument. See mgc.distance for details.

A list of trailing arguments to pass to the distance function specified in dist.xfm.X. Defaults to list(method='euclidean').

the return argument for the specified dist.xfm.X containing the distance matrix. Defaults to FALSE.

is.null(dist.return) use the return argument directly from dist.xfm as the distance matrix. Should be a \([n \times n]\) matrix.

is.character(dist.return) use dist.xfm.X[[dist.return]] as the distance matrix. Should be a \([n \times n]\) matrix.

a boolean indicating whether your \( Y \) input is a distance matrix or not. Defaults to FALSE.

if is.dist == FALSE, a distance function to transform \( Y \). If a distance function is passed, it should accept an \([n \times d]\) matrix of \( n \) samples in \( d \) dimensions and return a \([n \times n]\) distance matrix as the dist.return.Y return argument. See mgc.distance for details.

a list of trailing arguments to pass to the distance function specified in dist.xfm.Y. Defaults to list(method='euclidean').

the return argument for the specified dist.xfm.Y containing the distance matrix. Defaults to FALSE.

is.null(dist.return) use the return argument directly from dist.xfm.Y(Y) as the distance matrix. Should be a \([n \times n]\) matrix.

is.character(dist.return) use dist.xfm.Y(Y)[[dist.return]] as the distance matrix. Should be a \([n \times n]\) matrix.

specifies the number of replicates to use for the permutation test. Defaults to 1000.

is a string that specifies which global correlation to build up-on. Defaults to 'mgc'.

'mgc' use the MGC global correlation.

'dcor' use the decor global correlation.

'mantel' use the mantel global correlation.

'rank' use the rank global correlation.

the number of cores to use for the permutations. Defaults to 1.

A list containing the following:

P-value of MGC

is the sample MGC statistic within \([-1,1]\)

P-value of the local correlations by double matrix index.

the local correlations

the optimal scale identified by MGC

specifies which global correlation was used
Details

A test of independence using the MGC approach, described in Vogelstein et al. (2019). For $X \sim F_X$, $Y \sim F_Y$:

$$H_0 : F_X \neq F_Y$$

and:

$$H_A : F_X = F_Y$$

Note that one should avoid report positive discovery via minimizing individual p-values of local correlations, unless corrected for multiple hypotheses.

For details on usage see the help vignette: `vignette("mgc", package = "mgc")`

Author(s)

Eric Bridgeford and C. Shen

References


Examples

```r
## Not run:
library(mgc)
n = 100; d = 2
data <- mgc.sims.linear(n, d)
# note: on real data, one would put nperm much higher (at least 100)
# nperm is set to 10 merely for demonstration purposes
result <- mgc.test(data$X, data$Y, nperm=10)
## End(Not run)
```
Index

ConnCompLabel, 2
discr.sims.cross, 3
discr.sims.exp, 4
discr.sims.fat_tails, 5
discr.sims.linear, 6
discr.sims.radial, 7
discr.stat, 8
discr.test.one_sample, 10
discr.test.two_sample, 12
dist, 15
mgc.dist.xfm, 14
mgc.distance, 9, 11, 13, 15, 18, 31, 32, 34
mgc.ksample, 16
mgc.localcorr, 17
mgc.localcorr.driver, 19
mgc.sims.2ball, 20
mgc.sims.2sphere, 21
mgc.sims.cubic, 21
mgc.sims.exp, 23
mgc.sims.joint, 24
mgc.sims.linear, 25
mgc.sims.quad, 26
mgc.sims.spiral, 27
mgc.sims.step, 28
mgc.sims.ubern, 29
mgc.sims.wshape, 30
mgc.stat, 31
mgc.test, 16, 33