# Package ‘flintyR’

October 13, 2022

**Title** Flexible and Interpretable Non-Parametric Tests of Exchangeability

**Version** 0.0.2

**Date** 2021-09-27

**Description** Given a multivariate dataset and some knowledge about the dependencies between its features, it is important to ensure the observations or individuals are exchangeable before fitting a model to the data in order to make inferences from it, or assigning randomized treatments in order to estimate treatment effects. This package provides a flexible nonparametric test of exchangeability, allowing the user to specify the feature dependencies by hand. It can be used directly to evaluate whether a sample is exchangeable, and can also be piped into larger procedures that require exchangeable samples as outputs (e.g., clustering or community detection). See Aw, Spence and Song (2021+) for the accompanying paper.

**License** GPL (>= 3)

**Imports** Rcpp (>= 1.0.6), doParallel, foreach, assertthat, testthat, stats

**Suggests** rmarkdown, knitr, devtools, tidyverse

**LinkingTo** Rcpp, RcppArmadillo

**URL** `https://alanaw1.github.io/flintyR/`

**BugReports** `https://github.com/alanaw1/flintyR/issues`

**RoxygenNote** 7.1.1

**Config/testthat/edition** 3

**NeedsCompilation** yes

**Author** Alan Aw [cre, aut] (<`https://orcid.org/0000-0001-9455-7878`>), Jeffrey Spence [ctb]

**Maintainer** Alan Aw <`alanaw1@berkeley.edu`>

**Repository** CRAN

**Date/Publication** 2021-09-28 12:40:07 UTC

# R topics documented:

---

## `flintyR-package` *Flexible and Interpretable Non-Parametric Tests of Exchangeability*

---

### Description

Given a multivariate dataset and some knowledge about the dependencies between its features, it is important to ensure the observations or individuals are exchangeable before fitting a model to the data in order to make inferences from it, or assigning randomized treatments in order to estimate treatment effects. This package provides a flexible non-parametric test of exchangeability, allowing the user to specify the feature dependencies by hand. It can be used directly to evaluate whether a sample is exchangeable, and can also be piped into larger procedures that require exchangeable samples as outputs (e.g., clustering or community detection). See Aw, Spence and Song (2021+) for the accompanying paper.

## Package Content

Index of help topics:

| | |
|---|---|
| `blockGaussian` | `Approximate p-value for Test of Exchangeability (Assuming Large N and P with Block Dependencies)` |
| `blockLargeP` | `Approximate p-value for Test of Exchangeability (Assuming Large P with Block Dependencies)` |
| `blockPermute` | `p-value Computation for Test of Exchangeability with Block Dependencies` |
| `buildForward` | `Map from Indices to Label Pairs` |
| `buildReverse` | `Map from Label Pairs to Indices` |
| `cacheBlockPermute1` | `Resampling Many V Statistics (Version 1)` |
| `cacheBlockPermute2` | `Resampling Many V Statistics (Version 2)` |
| `cachePermute` | `Permutation by Caching Distances` |
| `distDataLargeP` | `Asymptotic p-value of Exchangeability Using Distance Data` |
| `distDataPValue` | `A Non-parametric Test for Exchangeability and Homogeneity (Distance List Version)` |
| `distDataPermute` | `p-value Computation for Test of Exchangeability Using Distance Data` |
| `flintyR-package` | `Flexible and Interpretable Non-Parametric Tests of Exchangeability` |
| `getBinVStat` | `V Statistic for Binary Matrices` |
| `getBlockCov` | `Covariance Computations Between Pairs of Distances (Block Dependencies Case)` |
| `getChi2Weights` | `Get Chi Square Weights` |
| `getCov` | `Covariance Computations Between Pairs of Distances (Independent Case)` |
| `getHammingDistance` | `A Hamming Distance Vector Calculator` |
| `getLpDistance` | `A l_p^p Distance Vector Calculator` |
| `getPValue` | `A Non-parametric Test for Exchangeability and Homogeneity` |
| `getRealVStat` | `V Statistic for Real Matrices` |
| `hamming_bitwise` | `Fast Bitwise Hamming Distance Vector Computation` |
| `indGaussian` | `Approximate p-value for Test of Exchangeability (Assuming Large N and P)` |
| `indLargeP` | `Approximate p-value for Test of Exchangeability (Assuming Large P)` |
| `lp_distance` | `Fast l_p^p Distance Vector Computation` |
| `naiveBlockPermute1` | `Resampling V Statistic (Version 1)` |
| `naiveBlockPermute2` | `Resampling V Statistic (Version 2)` |
| `weightedChi2P` | `Tail Probability for Chi Square Convolution Random Variable` |

## Maintainer

NA

## Author(s)

NA

---

`blockGaussian` *Approximate p-value for Test of Exchangeability (Assuming Large N and P with Block Dependencies)*

---

## Description

Computes the large $(N, P)$ asymptotic p-value for dataset $\mathbf{X}$, assuming its $P$ features are independent within specified blocks.

## Usage

```
blockGaussian(X, block_boundaries, block_labels, p)
```

## Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform test of exchangeability |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts. |
| `block_labels` | Length $P$ vector recording the block label of each feature. |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

## Details

This is the large $N$ and large $P$ asymptotics of the permutation test.

Dependencies: getBinVStat, getRealVStat, getBlockCov, getChi2Weights

## Value

The asymptotic p-value

---

## `blockLargeP` *Approximate p-value for Test of Exchangeability (Assuming Large P with Block Dependencies)*

---

### Description

Computes the large $P$ asymptotic p-value for dataset $\mathbf{X}$, assuming its $P$ features are independent within specified blocks.

### Usage

```
blockLargeP(X, block_boundaries, block_labels, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform test of exchangeability |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts. |
| `block_labels` | Length $P$ vector recording the block label of each feature. |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

This is the large $P$ asymptotics of the permutation test.

Dependencies: getBinVStat, getRealVStat, getChi2Weights, weightedChi2P, getBlockCov

### Value

The asymptotic p-value

---

## `blockPermute` *p-value Computation for Test of Exchangeability with Block Dependencies*

---

### Description

Generates a block permutation p-value. Uses a heuristic to decide whether to use distance caching or simple block permutations.

### Usage

```
blockPermute(X, block_boundaries = NULL, block_labels = NULL, nruns, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform permutation resampling |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts. Default is NULL. |
| `block_labels` | Length $P$ vector recording the block label of each feature. Default is NULL. |
| `nruns` | The resampling number (use at least 1000) |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

Dependencies: buildForward, buildReverse, cachePermute, cacheBlockPermute1, cacheBlockPermute2, getHammingDistance, getLpDistance, naiveBlockPermute1, naiveBlockPermute2

### Value

The block permutation p-value

---

## `buildForward` *Map from Indices to Label Pairs*

### Description

Builds a map from indexes to pairs of labels. This is for caching distances, to avoid recomputing Hamming distances especially when dealing with high-dimensional (large $P$) matrices.

### Usage

```
buildForward(N)
```

### Arguments

| | |
|---|---|
| `N` | Sample size, i.e., nrow($\mathbf{X}$) |

### Details

Dependencies: None

### Value

$N \times N$ matrix whose entries record the index corresponding to the pair of labels (indexed by the matrix dims)

---

## `buildReverse` *Map from Label Pairs to Indices*

---

### Description

Builds a map from pairs of labels to indexes. This is for caching distances, to avoid recomputing Hamming distances especially when dealing with high-dimensional (large $P$) matrices.

### Usage

```
buildReverse(N)
```

### Arguments

| | |
|---|---|
| `N` | Sample size, i.e., nrow($\mathbf{X}$) |

### Details

Dependencies: None

### Value

$N \times N$ matrix whose entries record the index corresponding to the pair of labels (indexed by the matrix dims)

---

## `cacheBlockPermute1` *Resampling Many V Statistics (Version 1)*

---

### Description

Generates a block permutation distribution of $V$ statistic. Precomputes distances and some indexing arrays to quickly generate samples from the block permutation distribution of the $V$ statistic of $\mathbf{X}$.

### Usage

```
cacheBlockPermute1(X, block_labels, nruns, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform permutation resampling |
| `block_labels` | Length $P$ vector recording the block label of each feature |
| `nruns` | The resampling number (use at least 1000) |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

## Details

This version is with block labels specified.

Dependencies: buildForward, buildReverse, cachePermute, getHammingDistance, getLpDistance

## Value

A vector of resampled values of the $V$ statistic

---

# `cacheBlockPermute2` *Resampling Many V Statistics (Version 2)*

## Description

Generates a block permutation distribution of $V$ statistic. Precomputes distances and some indexing arrays to quickly generate samples from the block permutation distribution of the $V$ statistic of $\mathbf{X}$.

## Usage

```
cacheBlockPermute2(X, block_boundaries, nruns, p = 2)
```

## Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform permutation resampling |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts |
| `nruns` | The resampling number (use at least 1000) |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

## Details

This version is with block boundaries specified.

Dependencies: buildForward, buildReverse, cachePermute, getHammingDistance, getLpDistance

## Value

A vector of resampled values of the $V$ statistic

---

## cachePermute *Permutation by Caching Distances*

---

### Description

What do you do when you have to compute pairwise distances many times, and those damn distances take a long time to compute? Answer: You cache the distances and permute the underlying sample labels!

### Usage

```
cachePermute(dists, forward, reverse)
```

### Arguments

| | |
|---|---|
| `dists` | $\binom{N}{2}$ by $B$ matrix, with each column containing the distances (ex: Hamming, $l_p^p$) for the block |
| `forward` | $N \times N$ matrix mapping the pairs of sample labels to index of the $\binom{N}{2}$-length vector |
| `reverse` | $\binom{N}{2} \times 2$ matrix mapping the index to pairs of sample labels |

### Details

This function permutes the distances (Hamming, $l_p^p$, etc.) within blocks. Permutations respect the fact that we are actually permuting the underlying labels. Arguments forward and reverse should be precomputed using buildForward and buildReverse.

Dependencies: buildForward, buildReverse

### Value

A matrix with same dimensions as dists containing the block-permuted pairwise distances

---

## distDataLargeP *Asymptotic p-value of Exchangeability Using Distance Data*

---

### Description

Generates an asymptotic p-value.

### Usage

```
distDataLargeP(dist_list)
```

### Arguments

| | |
|---|---|
| `dist_list` | The list (length $B$) of pairwise distance data. Each element in list should be either a distance matrix or a table recording pairwise distances. |

### Details

Generates a weighted convolution of chi-squares distribution of $V$ statistic by storing the provided list of distance data as an $\binom{N}{2} \times B$ array, and then using large-$P$ theory to generate the asymptotic null distribution against which the p-value of observed $V$ statistic is computed.

Each element of dist_list should be a $N \times N$ distance matrix.

Dependencies: buildReverse, getChi2Weights, weightedChi2P

### Value

The asymptotic p-value obtained from the weighted convolution of chi-squares distribution.

---

## `distDataPermute` *p-value Computation for Test of Exchangeability Using Distance Data*

### Description

Generates a block permutation p-value.

### Usage

```
distDataPermute(dist_list, nruns)
```

### Arguments

| | |
|---|---|
| `dist_list` | The list (length $B$) of pairwise distance data. Each element in list should be either a distance matrix or a table recording pairwise distances. |
| `nruns` | The resampling number (use at least 1000) |

### Details

Generates a block permutation distribution of $V$ statistic by storing the provided list of distance data as an $\binom{N}{2} \times B$ array, and then permuting the underlying indices of each individual to generate resampled $\binom{N}{2} \times B$ arrays. The observed $V$ statistic is also computed from the distance data.

Each element of dist_list should be a $N \times N$ distance matrix.

Dependencies: buildForward, buildReverse, cachePermute

### Value

The p-value obtained from comparing the empirical tail cdf of the observed $V$ statistic computed from distance data.

---

## `distDataPValue` *A Non-parametric Test for Exchangeability and Homogeneity (Distance List Version)*

---

### Description

Computes the p-value of a multivariate dataset, which informs the user if the sample is exchangeable at a given significance level, while simultaneously accounting for feature dependencies. See Aw, Spence and Song (2021) for details.

### Usage

```
distDataPValue(dist_list, largeP = FALSE, nruns = 1000)
```

### Arguments

| | |
|---|---|
| `dist_list` | The list of distances |
| `largeP` | Boolean indicating whether to use large $P$ asymptotics. Default is FALSE. |
| `nruns` | Resampling number for exact test. Default is 1000. |

### Details

This version takes in a list of distance matrices recording pairwise distances between individuals across $B$ independent features.

Dependencies: distDataLargeP and distDataPermute from auxiliary.R

### Value

The p-value to be used to test the null hypothesis of exchangeability

---

## `getBinVStat` *V Statistic for Binary Matrices*

---

### Description

Computes $V$ statistic for a binary matrix $\mathbf{X}$, as defined in Aw, Spence and Song (2021+).

### Usage

```
getBinVStat(X)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ binary matrix |

### Details

Dependencies: getHammingDistance

### Value

$V(\mathbf{X})$, the variance of the pairwise Hamming distance between samples

### Examples

```
X <- matrix(nrow = 5, ncol = 10, rbinom(50, 1, 0.5))
getBinVStat(X)
```

---

## `getBlockCov` *Covariance Computations Between Pairs of Distances (Block Dependencies Case)*

### Description

Computes covariance matrix entries and associated alpha, beta and gamma quantities defined in Aw, Spence and Song (2021), for partitionable features that are grouped into blocks. Uses precomputation to compute the unique entries of the asymptotic covariance matrix of the pairwise Hamming distances in $O(N^2)$ time.

### Usage

```
getBlockCov(X, block_boundaries, block_labels, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts. |
| `block_labels` | Length $P$ vector recording the block label of each feature. |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

This is used in the large $P$ asymptotics of the permutation test.

Dependencies: buildReverse, getHammingDistance, getLpDistance

### Value

The three distinct entries of covariance matrix, $(\alpha, \beta, \gamma)$

---

## `getChi2Weights` *Get Chi Square Weights*

---

### Description

Computes weights for the asymptotic random variable from the $\alpha, \beta$ and $\gamma$ computed of data array $\mathbf{X}$.

### Usage

```
getChi2Weights(alpha, beta, gamma, N)
```

### Arguments

| | |
|---|---|
| `alpha` | covariance matrix entry computed from getCov |
| `beta` | covariance matrix entry computed from getCov |
| `gamma` | covariance matrix entry computed from getCov |
| `N` | The sample size, i.e., nrow(X) where X is the original dataset |

### Details

This is used in the large $P$ asymptotics of the permutation test.

Dependencies: None

### Value

The weights $(w_1, w_2)$

---

## `getCov` *Covariance Computations Between Pairs of Distances (Independent Case)*

---

### Description

Computes covariance matrix entries and associated alpha, beta and gamma quantities defined in Aw, Spence and Song (2021), assuming the $P$ features of the dataset $\mathbf{X}$ are independent.

### Usage

```
getCov(X, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix |
| `p` | The power $p$ of $l_p^p$, i.e., $||x||_p^p = (x_1^p + ...x_n^p)$ |

### Details

This is used in the large $P$ asymptotics of the permutation test.

Dependencies: buildReverse, getLpDistance

### Value

The three distinct entries of covariance matrix, $(\alpha, \beta, \gamma)$

---

## `getHammingDistance` *A Hamming Distance Vector Calculator*

### Description

Computes all pairwise Hamming distances for a binary matrix $\mathbf{X}$.

### Usage

```
getHammingDistance(X)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ binary matrix |

### Details

Dependencies: hamming_bitwise from fast_dist_calc.cpp

### Value

A length $\binom{N}{2}$ vector of pairwise Hamming distances

### Examples

```
X <- matrix(nrow = 5, ncol = 10, rbinom(50, 1, 0.5))
getHammingDistance(X)
```

---

## `getLpDistance` *A l_p^p Distance Vector Calculator*

---

### Description

Computes all pairwise $l_p^p$ distances for a real matrix $\mathbf{X}$, for a specified choice of Minkowski norm exponent $p$.

### Usage

```
getLpDistance(X, p)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ real matrix |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

Dependencies: lp_distance from fast_dist_calc.cpp

### Value

A length $\binom{N}{2}$ vector of pairwise $l_p^p$ distances

### Examples

```
X <- matrix(nrow = 5, ncol = 10, rnorm(50))
getLpDistance(X, p = 2)
```

---

## `getPValue` *A Non-parametric Test for Exchangeability and Homogeneity*

---

### Description

Computes the p-value of a multivariate dataset $\mathbf{X}$, which informs the user if the sample is exchangeable at a given significance level, while simultaneously accounting for feature dependencies. See Aw, Spence and Song (2021) for details.

### Usage

```
getPValue(
  X,
  block_boundaries = NULL,
  block_labels = NULL,
  largeP = FALSE,
  largeN = FALSE,
  nruns = 5000,
  p = 2
)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform test of exchangeability |
| `block_boundaries` | Vector denoting the positions where a new block of non-independent features starts. Default is NULL. |
| `block_labels` | Length $P$ vector recording the block label of each feature. Default is NULL. |
| `largeP` | Boolean indicating whether to use large $P$ asymptotics. Default is FALSE. |
| `largeN` | Boolean indicating whether to use large $N$ asymptotics. Default is FALSE. |
| `nruns` | Resampling number for exact test. Default is 5000. |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$. Default is 2. |

### Details

Automatically detects if dataset is binary, and runs the Hamming distance version of test if so. Otherwise, computes the squared Euclidean distance between samples and evaluates whether the variance of Euclidean distances, $V$, is atypically large under the null hypothesis of exchangeability. Note the user may tweak the choice of power $p$ if they prefer an $l_p^p$ distance other than Euclidean.

Under the hood, the variance statistic, $V$, is computed efficiently. Moreover, the user can specify their choice of block permutations, large $P$ asymptotics, or large $P$ and large $N$ asymptotics. The latter two return reasonably accurate p-values for moderately large dimensionalities.

User recommendations: When the number of independent blocks $B$ or number of independent features $P$ is at least 50, it is safe to use large $P$ asymptotics. If $P$ or $B$ is small, however, stick with permutations.

Dependencies: All functions in auxiliary.R

### Value

The p-value to be used to test the null hypothesis of exchangeability

### Examples

```
# Example 1 (get p-value of small matrix with independent features using exact test)
suppressWarnings(require(doParallel))
# registerDoParallel(cores = 2)
```

```
X1 <- matrix(nrow = 5, ncol = 10, rbinom(50, 1, 0.5)) # binary matrix, small
getPValue(X1) # perform exact test with 5000 permutations

# should be larger than 0.05

# Example 2 (get p-value of high-dim matrix with independent features using asymptotic test)
X2 <- matrix(nrow = 10, ncol = 1000, rnorm(1e4)) # real matrix, large enough
getPValue(X2, p = 2, largeP = TRUE) # very fast

# should be larger than 0.05
# getPValue(X2, p = 2) # slower, do not run (Output: 0.5764)

# Example 3 (get p-value of high-dim matrix with partitionable features using exact test)

X3 <- matrix(nrow = 10, ncol = 1000, rbinom(1e4, 1, 0.5))
getPValue(X3, block_labels = rep(c(1,2,3,4,5), 200))

# Warning message: # there are features that have zero variation (i.e., all 0s or 1s)
# In getPValue(X3, block_labels = rep(c(1, 2, 3, 4, 5), 200)) :
# There exist columns with all ones or all zeros for binary X.

# Example 4 (get p-value of high-dim matrix with partitionable features using asymptotic test)

## This elaborate example generates binarized versions of time series data.

# Helper function to binarize a marker
# by converting z-scores to {0,1} based on
# standard normal quantiles
binarizeMarker <- function(x, freq, ploidy) {
 if (ploidy == 1) {
   return((x > qnorm(1-freq)) + 0)
 } else if (ploidy == 2) {
   if (x <= qnorm((1-freq)^2)) {
     return(0)
   } else if (x <= qnorm(1-freq^2)) {
     return(1)
   } else return(2)
 } else {
   cat("Specify valid ploidy number, 1 or 2")
 }
}

getAutoRegArray <- function(B, N, maf_l = 0.38, maf_u = 0.5, rho = 0.5, ploid = 1) {
# get minor allele frequencies by sampling from uniform
mafs <- runif(B, min = maf_l, max = maf_u)
# get AR array
ar_array <- t(replicate(N, arima.sim(n = B, list(ar=rho))))
# theoretical column variance
column_var <- 1/(1-rho^2)
# rescale so that variance per marker is 1
ar_array <- ar_array / sqrt(column_var)
# rescale each column of AR array
```

```
for (b in 1:B) {
  ar_array[,b] <- sapply(ar_array[,b],
                         binarizeMarker,
                         freq = mafs[b],
                         ploidy = ploid)
}
return(ar_array)
}

## Function to generate the data array with desired number of samples
getExHaplotypes <- function(N) {
  array <- do.call("cbind",
                   lapply(1:50, function(x) {getAutoRegArray(N, B = 20)}))
  return(array)
}

##  Generate data and run test
X4 <- getExHaplotypes(10)
getPValue(X4, block_boundaries = seq(from = 1, to = 1000, by = 25), largeP = TRUE)

# stopImplicitCluster()
```

---

## `getRealVStat` *V Statistic for Real Matrices*

### Description

Computes $V$ statistic for a real matrix $\mathbf{X}$, where $V(\mathbf{X})$ = scaled variance of $l_p^p$ distances between the row samples of $\mathbf{X}$.

### Usage

```
getRealVStat(X, p)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ real matrix |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$s |

### Details

Dependencies: getLpDistance

### Value

$V(\mathbf{X})$, the variance of the pairwise $l_p^p$ distance between samples

### Examples

```
X <- matrix(nrow = 5, ncol = 10, rnorm(50))
getRealVStat(X, p = 2)
```

---

## `hamming_bitwise` *Fast Bitwise Hamming Distance Vector Computation*

### Description

Takes in a binary matrix X, whose transpose t(X) has N rows, and computes a vector recording all N choose 2 pairwise Hamming distances of t(X), ordered lexicographically.

### Usage

```
hamming_bitwise(X)
```

### Arguments

| | |
|---|---|
| `X` | binary matrix (IntegerMatrix class ) |

### Value

vector of Hamming distances (NumericVector class)

### Examples

```
# t(X) = [[1,0], [0,1], [1,1]] --> output = [2,1,1]
```

---

## `indGaussian` *Approximate p-value for Test of Exchangeability (Assuming Large N and P)*

### Description

Computes the large $(N, P)$ asymptotic p-value for dataset $\mathbf{X}$, assuming its $P$ features are independent

### Usage

```
indGaussian(X, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform test of exchangeability |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

## Details

This is the large $N$ and large $P$ asymptotics of the permutation test.

Dependencies: getBinVStat, getRealVStat, getCov, getChi2Weights

## Value

The asymptotic p-value

---

`indLargeP` *Approximate p-value for Test of Exchangeability (Assuming Large P)*

## Description

Computes the large $P$ asymptotic p-value for dataset $\mathbf{X}$, assuming its $P$ features are independent.

## Usage

```
indLargeP(X, p = 2)
```

## Arguments

| | |
|---|---|
| `X` | The binary or real matrix on which to perform test of exchangeability |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

## Details

This is the large $P$ asymptotics of the permutation test.

Dependencies: getBinVStat, getRealVStat, getChi2Weights, weightedChi2P, getCov

## Value

The asymptotic p-value

---

## `lp_distance` *Fast l_p^p Distance Vector Computation*

---

### Description

Takes in a double matrix X, whose transpose t(X) has N rows, and computes a vector recording all $\binom{N}{2}$ pairwise $l_p^p$ distances of t(X), ordered lexicographically.

### Usage

```
lp_distance(X, p)
```

### Arguments

| | |
|---|---|
| `X` | double matrix (arma::mat class) |
| `p` | numeric Minkowski power (double class) |

### Value

vector of $l_p^p$ distances (arma::vec class)

### Examples

```
# X = [[0.5,0.5],[0,1],[0.3,0.7]] --> lPVec = [x,y,z]
# with x = (0.5^p + 0.5^p)
```

---

## `naiveBlockPermute1` *Resampling V Statistic (Version 1)*

---

### Description

Generates a new array $\mathbf{X}'$ under the permutation null and then returns the $V$ statistic computed for $\mathbf{X}'$.

### Usage

```
naiveBlockPermute1(X, block_labels, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ binary or real matrix |
| `block_labels` | A vector of length $P$, whose $p$th component indicates the block membership of feature $p$ |
| `p` | The power $p$ of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

This is Version 1, which takes in the block labels. It is suitable in the most general setting, where the features are grouped by labels. Given original $\mathbf{X}$ and a list denoting labels of each feature, independently permutes the rows within each block of $\mathbf{X}$ and returns resulting $V$. If block labels are not specified, then features are assumed independent, which is to say that block_labels is set to 1:ncol($\mathbf{X}$).

Dependencies: getBinVStat, getRealVStat

### Value

$V(\mathbf{X}')$, where $\mathbf{X}'$ is a resampled by permutation of entries blockwise

### Examples

```
X <- matrix(nrow = 5, ncol = 10, rnorm(50)) # real matrix example
naiveBlockPermute1(X, block_labels = c(1,1,2,2,3,3,4,4,5,5), p = 2) # use Euclidean distance

X <- matrix(nrow = 5, ncol = 10, rbinom(50, 1, 0.5)) # binary matrix example
naiveBlockPermute1(X, block_labels = c(1,1,2,2,3,3,4,4,5,5))
```

---

## `naiveBlockPermute2` *Resampling V Statistic (Version 2)*

### Description

Generates a new array $\mathbf{X}'$ under the permutation null and then returns the $V$ statistic computed for $\mathbf{X}'$.

### Usage

```
naiveBlockPermute2(X, block_boundaries, p = 2)
```

### Arguments

| | |
|---|---|
| `X` | The $N \times P$ binary or real matrix |
| `block_boundaries` | A vector of length at most P, whose entries indicate positions at which to demarcate blocks |
| `p` | The power p of $l_p^p$, i.e., $\|\|x\|\|_p^p = (x_1^p + ...x_n^p)$ |

### Details

This is Version 2, which takes in the block boundaries. It is suitable for use when the features are already arranged such that the block memberships are determined by index delimiters. Given original $\mathbf{X}$ and a list denoting labels of each feature, independently permutes the rows within each block of $\mathbf{X}$ and returns resulting $V$. If block labels are not specified, then features are assumed independent, which is to say that block_labels is set to 1:ncol($\mathbf{X}$).

Dependencies: getBinVStat, getRealVStat

## Value

$V(\mathbf{X}')$, where $\mathbf{X}'$ is a resampled by permutation of entries blockwise

## Examples

```
X <- matrix(nrow = 5, ncol = 10, rnorm(50)) # real matrix example
naiveBlockPermute2(X, block_boundaries = c(4,7,9), p = 2) # use Euclidean distance

X <- matrix(nrow = 5, ncol = 10, rbinom(50, 1, 0.5)) # binary matrix example
naiveBlockPermute2(X, block_boundaries = c(4,7,9))
```

---

# `weightedChi2P` *Tail Probability for Chi Square Convolution Random Variable*

## Description

Computes $P(X > val)$ where $X = w_1 Y + w_2 Z$, where $Y$ is chi square distributed with $d_1$ degrees of freedom, $Z$ is chi square distributed with $d_2$ degrees of freedom, and $w_1$ and $w_2$ are weights with $w_2$ assumed positive. The probability is computed using numerical integration of the densities of the two chi square distributions. (Method: trapezoidal rule)

## Usage

```
weightedChi2P(val, w1, w2, d1, d2)
```

## Arguments

| | |
|---|---|
| `val` | observed statistic |
| `w1` | weight of first chi square rv |
| `w2` | weight of second chi square rv, assumed positive |
| `d1` | degrees of freedom of first chi square rv |
| `d2` | degrees of freedom of second chi square rv |

## Details

This is used in the large $P$ asymptotics of the permutation test.

Dependencies: None

## Value

1 - CDF = P(X > val)

# Index