Package ‘randRotation’

May 7, 2024

Title Random Rotation Methods for High Dimensional Data with Batch Structure

Version 1.16.0

Description A collection of methods for performing random rotations on high-dimensional, normally distributed data (e.g. microarray or RNA-seq data) with batch structure. The random rotation approach allows exact testing of dependent test statistics with linear models following arbitrary batch effect correction methods.

License GPL-3

biocViews Software, Sequencing, BatchEffect, BiomedicalInformatics, RNASeq, Preprocessing, Microarray, DifferentialExpression, GeneExpression, Genetics, MicroRNAArray, Normalization, StatisticalMethod

Encoding UTF-8

LazyData false

RdMacros Rdpack

Imports methods, graphics, utils, stats, Rdpack (>= 0.7)

RoxygenNote 7.2.3

Suggests knitr, BiocParallel, lme4, nlme, rmarkdown, BiocStyle, testthat (>= 2.1.0), limma, sva

VignetteBuilder knitr

URL https://github.com/phettegger/randRotation

BugReports https://github.com/phettegger/randRotation/issues

git_url https://git.bioconductor.org/packages/randRotation
git_branch RELEASE_3_19
git_last_commit ece5d10
git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-06

Author Peter Hettegger [aut, cre] (<https://orcid.org/0000-0001-8557-588X>)

Maintainer Peter Hettegger <p.hettegger@gmail.com>
Contents

randRotation-package ........................................ 2
.fdr.qu ..................................................... 3
contrastModel ................................................ 4
dim,initRandrot-method ...................................... 5
dimnames,initRandrot-method ................................ 5
initBatchRandrot-class ...................................... 6
initRandrot ................................................... 6
initRandrot-class ........................................... 10
initRandrotW .................................................. 11
pFdr .......................................................... 12
qunif .......................................................... 15
randorth ....................................................... 16
randpermut .................................................... 17
randrot ........................................................ 18
randRotation-defunct ....................................... 20
rotateStat ..................................................... 20
rotateStat-class ............................................. 22
show,initRandrot-method ................................... 23
weights,initRandrot-method ................................. 24
X_decomp ...................................................... 24

Index 26

randRotation-package  randRotation: Random Rotation Methods for High Dimensional Data with Batch Structure

Description

A collection of methods for performing random rotations on high-dimensional, normally distributed data (e.g. microarray or RNA-seq data) with batch structure. The random rotation approach allows exact testing of dependent test statistics with linear models following arbitrary batch effect correction methods.

Details

Please refer to the package vignette for further details on usage and for a "quick start". rotateStat is the central function of the package. Methods are described in (Hettegger et al. 2021).

Author(s)

Maintainer: Peter Hettegger <p.hettegger@gmail.com> (ORCID)
.fdr.qu

References

See Also
Useful links:
- [https://github.com/phettegger/randRotation](https://github.com/phettegger/randRotation)
- Report bugs at [https://github.com/phettegger/randRotation/issues](https://github.com/phettegger/randRotation/issues)

---

.fdr.qu  
*Internal functions for p-value and FDR estimation*

Description
Internal functions for p-value and FDR estimation

Usage
```
.fdr.qu(
  s0,  # numeric vector of original (non-rotated) statistics.
  stats,  # numeric matrix of rotated statistics.
  beta = 0.05,  # numeric between 0 and 1. See (Yekutieli and Benjamini 1999).
  na.rm = FALSE,  # logical. Should missing values be removed ?
  ref.vector = sort(s0, decreasing = TRUE, na.last = TRUE)
)
```
```
.fdr.q(
  s0,  # numeric vector of original (non-rotated) statistics.
  stats,  # numeric matrix of rotated statistics.
  beta = 0.05,  # numeric between 0 and 1. See (Yekutieli and Benjamini 1999).
  na.rm = FALSE,  # logical. Should missing values be removed ?
  ref.vector = sort(s0, decreasing = TRUE, na.last = TRUE)
)
```
```
.pFdr(s0, stats, method, pooled, na.rm, beta)
```

Arguments
- `s0`  
  numeric vector of original (non-rotated) statistics.
- `stats`  
  numeric matrix of rotated statistics.
- `beta`  
  numeric between 0 and 1. See (Yekutieli and Benjamini 1999).
- `na.rm`  
  logical. Should missing values be removed ?
contrastModel

**ref.vector**  
Reference vector defining at which grid points of $s\theta$ and $(\text{stats})$ the FDRs are approximated. All other points are approximated by spline interpolation. NAs are removed from ref.vector

**method**  
A p-value or FDR adjustment method, see `pFdr`.

**pooled**  
logical. TRUE if marginal distributions are exchangeable for all features so that rotated stats can be pooled, see `pFdr`.

**Value**

numeric vector of (adjusted) p-value or FDR estimations for $s\theta$.

**References**


**contrastModel**  
`Create transformed model matrix for contrast rotation`

**Description**

This function takes a model matrix $X$ and a contrast matrix $C$ and creates a transformed model matrix corresponding to a transformed set of coefficients.

**Usage**

`contrastModel(X, C, coef.h = seq_len(ncol(C)))`

**Arguments**

- **X**  
  (numeric) model matrix with dimensions samples x coefficients.

- **C**  
  (numeric) contrast matrix with dimensions coefficients x contrasts. The contrast matrix must have full column rank.

- **coef.h**  
  column numbers of contrasts (in $C$) which should be set as coef.h in the transformed model, see `initRandrot`. All columns are set as coef.h by default.

**Details**

The last $n$ coefficients of the transformed model matrix correspond to the $n$ contrasts. By default, all contrasts are set as coef.h. See package vignette for examples of data rotations with contrasts.

**Value**

A transformed model matrix with coef.h set as attribute.
Author(s)

Peter Hettegger

Examples

```r
group <- c("A", "A", "B", "B")
X <- model.matrix(~0+group)
C <- cbind(contrast1 = c(1, -1))
X2 <- contrastModel(X, C)
```

---

**dim,initRandrot-method**

*Dimensions of an Object*

**Description**

Retrieve the dimensions of an object.

**Usage**

```r
## S4 method for signature 'initRandrot'
dim(x)
```

```r
## S4 method for signature 'initBatchRandrot'
dim(x)
```

**Arguments**

- `x` An object of class `initRandrot-class` or `initBatchRandrot-class`.

**Value**

Vector of length two with number of features and number of samples. See also `initRandrot`.

---

**dimnames,initRandrot-method**

*Dimnames of an Object*

**Description**

Retrieve the dimnames of an object.
Usage

```r
## S4 method for signature 'initRandrot'
dimnames(x)
```

```r
## S4 method for signature 'initBatchRandrot'
dimnames(x)
```

Arguments

- `x` An object of class `initRandrot-class` or `initBatchRandrot-class`.

Value

A list with names of features and samples, see `initRandrot`.

---

`initBatchRandrot-class`

*Initialised random rotation batch object*

Description

This class contains `initRandrot` or `initRandrotW` class objects for each batch. See also descriptions in `initRandrot` and `initRandrot-class`.

Components

- `batch.obj` List of `initRandrot` or `initRandrotW` class objects for each batch.
- `split.by` List of sample indices for each batch.

Author(s)

Peter Hettegger

---

`initRandrot`

*Initialisation of a random rotation Object*

Description

Initialization of a linear model for subsequent generation of randomly rotated data (`randrot`) associated with the null hypothesis $H_0: \beta_{\text{coef.h}} = 0$. Basics of rotation tests are found in (Hettegger et al. 2021) and (Langsrud 2005).
Usage

initRandrot(Y = NULL, X = NULL, coef.h = NULL, weights = NULL, cormat = NULL)

initBatchRandrot(
  Y = NULL,
  X = NULL,
  coef.h = NULL,
  batch = NULL,
  weights = NULL,
  cormat = NULL
)

## S4 method for signature 'list'
initBatchRandrot(
  Y = NULL,
  X = Y$design,
  coef.h = NULL,
  batch = NULL,
  weights = Y$weights,
  cormat = NULL
)

## S4 method for signature 'list'
initRandrot(
  Y = NULL,
  X = Y$design,
  coef.h = NULL,
  weights = Y$weights,
  cormat = NULL
)

Arguments

Y
  a data matrix with features x samples dimensions or a list with elements E, design and weights (see Details). Missing values (NA) are allowed but e.g. lead to NAs for all samples of the respective features in the rotated dataset and should thus be avoided. We highly recommend avoiding missing values by e.g. replacing them by imputation or removing features containing NAs.

X
  the design matrix of the experiment with samples x coefficients dimensions. For initBatchRandrot, specify the design matrix without the batch variable. A warning is generated if X[, coef.d] does not have full rank, see Details.

coef.h
  single integer or vector of integers specifying the "hypothesis coefficients" (H₀ coefficients). coef.h should correspond to the last columns in X (see Details). If available, attr(X, "coef.h") is used, see contrastModel. By default, all coefficients are set as H₀ coefficients. If coef.h is set -1, no coefficient is set as H₀ coefficient.

weights
  numerical matrix of finite positive weights > 0 (as in weighted least squares regression. Dimensions must be equal to dimensions of Y.
cormat  the sample correlation matrix with samples x samples dimensions. Must be a real symmetric positive-definite square matrix. See Details for usage in initBatchRandrot.

batch  Batch covariate of the same length as ncol(Y).

Details

This function performs basic initial checks and preparatory calculations for random rotation data generation. Nomenclature of variables is mainly as in (Langsrud 2005) and (Hettegger et al. 2021). See also package vignette for application examples.

Y can also be a list with elements E, design and weights. Y$E is thereby used as Y, Y$design is used as X and Y$weights is used as weights. By this, the functions are compatible with results from e.g. voom (limma package), see Examples.

coe.h specifies the model coefficients associated with the null hypothesis ("hypothesis coefficients"). All other model coefficients are considered as "determined coefficients" coef.d (Langsrud 2005). The design matrix is rearranged so that coef.h correspond to the last columns of the design matrix and coef.d correspond to the first columns of the design matrix. This is necessary for adequate transformation of the combined null-hypothesis $H_0 : \beta_{\text{coef.h}} = 0$ by QR decomposition. If X[,coef.d] does not have full rank, a warning is generated and coef.d is set to coef.d <- seq_len(qr(X[,coef.d])$rank).

Weights must be finite positive numerics greater zero. This is necessary for model (QR) decomposition and for back transformation of the rotated data into the original variance structure, see also randrot. Weights as estimated e.g. by voom (Law et al. 2014) are suitable and can be used without further processing. Note that due to the whitening transformation (i.e. by using the arguments weights and/or cormat) the rank of the transformed (whitened) design matrix X could change (become smaller), which could become dangerous for the fitting procedures. If you get errors using weights and/or cormat, try the routine without using weights and/or cormat to exclude this source of errors.

The following section provides a brief summary how rotations are calculated. A more general introduction is given in (Langsrud 2005). For reasons of readability, we omit writing `%*%` for matrix multiplication and write `*` for transposed matrix. The rotation is done by multiplying the features x samples data matrix Y with the transpose of the restricted random rotation matrix Rt

\[ Rt = Xd Xd^* + [Xh Xe] R [Xh Xe]^* \]

with R being a (reduced) random rotation matrix andXd, Xh and Xe being columns of the full QR decomposition of the design matrix X. [Xd Xh Xe] = qr.Q(qr(X), complete = TRUE), where Xd correspond to columns coef.d, Xh to columns coef.h and Xe to the remaining columns.

If weights and/or cormat are specified, each feature Y[i,] and the design matrix X are whitening transformed before rotation. The whitening matrix T is defined as T = solve(C) w, where solve(C) is the inverse Cholesky decomposition of the correlation matrix (cormat = CC*) and w is a diagonal matrix of the square roots of the sample weights for the according feature (w = diag(sqrt(weights[i,]))).

The rotated data for one feature y.r[i,] is thus calculated as

\[ y.r[i,] = ( \text{solve}(T) R t(y[i,])^* )^* \text{ and } [Xd Xh Xe] = qr.Q(qr(TX), complete = TRUE) \]

For weights = NULL and cormat = NULL, T is the identity matrix.
Note that a separate QR decomposition is calculated for each feature if weights are specified. The restricted random orthogonal matrix \( R_t \) is calculated with the same reduced random orthogonal matrix \( R \) for all features.

When using \texttt{initBatchRandrot}, \texttt{initRandrot} is called for each batch separately. When using \texttt{initBatchRandrot} with \texttt{cormat}, \texttt{cormat} needs to be a list of correlation matrices with one matrix for each batch. Note that this implicitly assumes a block design of the sample correlation matrix, where sample correlation coefficients between batches are zero. For a more general sample correlation matrix, allowing non-zero sample correlation coefficients between batches, see package vignette. Batches are split according to \texttt{split(seq_along(batch), batch)}.

**Value**

An initialised \texttt{initRandrot}, \texttt{initRandrotW} or \texttt{initBatchRandrot} object.

**Author(s)**

Peter Hettegger

**References**


**See Also**

\texttt{randrot}, \texttt{rotateStat}

**Examples**

# For further examples see '?rotateStat' and package vignette.

# Example 1: Compatibility with limma::voom

## Not run:
v <- voom(counts, design)
ir <- initRandrot(v)
## End(Not run)

# Example 2:

#set.seed(0)
# Dataframe of phenotype data (sample information)
# We simulate 2 sample classes processed in 3 batches
pdata <- data.frame(batch = rep(1:3, c(10,10,10)),
                    phenotype = rep(c("Control", "Cancer"), c(5,5)))
features <- 100

# Matrix with random gene expression data
data <- matrix(rnorm(features * nrow(pdata)), features)
rownames(data) <- paste("feature", 1:nrow(data))
mod1 <- model.matrix(~phenotype, pdata)

# Initialisation of the random rotation class
init1 <- initBatchRandrot(Y = data, X = mod1, coef.h = 2, batch = pdata$batch)
init1
# See '?rotateStat'

## initRandrot-class

### Initialised random rotation class

#### Description

List-based S4 class containing all information necessary to generate randomly rotated data with the `randrot` method. `initRandrot` and `initRandrotW` objects are created with the `initRandrot` method.

`initRandrotW` is organised as its base class `initRandrot`, although some components are changed or added.

#### Components

The following components are included as list elements:

- **X**: Original (non-transformed) design matrix.
- **Xhe, Xhe.Y, Yd**: Pre-multiplied matrix products needed for generation of rotated data (`randrot`).
- **coef.h, coef.d**: Indices of $H_0$ coefficients (coef.h or "hypothesis coefficients") and indices of all other coefficients (coef.d or "determined coefficients").
- **cormat**: Correlation matrix, see `initRandrot`.
- **tcholC**: Cholesky decomposition of cormat: `cormat = crossprod(tcholC)`.
- **rank**: Rank of the qr decomposition of (transformed/whitened) X

The following components are changed or added in `initRandrotW`-class as compared to `initRandrot-class`:

- **decomp.list**: List containing Xd, Xhe and rank of the transformed/whitened design matrix for each feature, see also `X_decomp`.

- **w**: Numeric matrix with dimensions features x samples containing component wise square root of the weight matrix, see `initRandrot`.

#### Author(s)

Peter Hettegger
initRandrotW

Description

Internal function

Usage

initRandrotW(Y, X, coef.h, coef.d, weights, cormat, cholCinv, tcholC)

Arguments

Y a data matrix with features x samples dimensions or a list with elements E, design and weights (see Details). Missing values (NA) are allowed but e.g. lead to NAs for all samples of the respective features in the rotated dataset and should thus be avoided. We highly recommend avoiding missing values by e.g. replacing them by imputation or removing features containing NAs.

X the design matrix of the experiment with samples x coefficients dimensions. For initBatchRandrot, specify the design matrix without the batch variable. A warning is generated if X[, coef.d] does not have full rank, see Details.

coefficients single integer or vector of integers specifying the "hypothesis coefficients" (H0 coefficients). coef.h should correspond to the last columns in X (see Details). If available, attr(X, "coef.h") is used, see contrastModel. By default, all coefficients are set as H0 coefficients. If coef.h is set -1, no coefficient is set as H0 coefficient.

coefficients Determined coefficients. These are all other coefficients that are not hypothesis coefficients (see also initRandrot).

weights numerical matrix of finite positive weights > 0 (as in weighted least squares regression. Dimensions must be equal to dimensions of Y.

cormat the sample correlation matrix with samples x samples dimensions. Must be a real symmetric positive-definite square matrix. See Details for usage in initBatchRandrot.

cholCinv Inverse of the Cholesky factorisation of cormat.

tcholC Transposed of the Cholesky factorisation of cormat.

Value

An initialised initRandrotW object.

Author(s)

Peter Hettegger
# Examples

```r
# For further examples see '?rotateStat' and package vignette.

#set.seed(0)

dataframe of phenotype data (sample information)
# We simulate 2 sample classes processed in 3 batches
pdata <- data.frame(phenotype = rep(c("Control", "Cancer"), c(5, 5)))
features <- 100

dataframe with random gene expression data
edata <- matrix(rnorm(features * nrow(pdata)), features)
rownames(data) <- paste("feature", 1:nrow(data))

mod1 <- model.matrix(~phenotype, pdata)

### Simulate weights
weights <- matrix(rbeta(features * nrow(pdata), 2, 2)+0.1, features)

# Initialisation of the random rotation class
init1 <- initRandrot(Y = edata, X = mod1, coef.h = 2,
                      weights = weights)
init1
```

## Description

This function calculates either (1) resampling based p-values with subsequent p-value adjustment using `stats::p.adjust` or (2) resampling based false-discovery-rates (FDRs) for rotated statistics from a `rotateStat` object.

### Usage

```r
pFdr(obj, method = "none", pooled = TRUE, na.rm = FALSE, beta = 0.05)
```

### Arguments

- **obj**: A `rotateStat` object as returned by `rotateStat`.
- **method**: Can be either "none" (default), "fdr.q", "fdr.qu" or any term that can be passed as method argument to `stats::p.adjust`, see Details. If method = "none", resampling based p-values without further adjustment are calculated.
- **pooled**: logical. TRUE (default) if marginal distributions are exchangeable for all features so that rotated stats can be pooled, see Details.
na.rm logical. NA values are ignored if set TRUE. NA values should be avoided and could e.g. be removed by imputation in original data or by removing features that contain NA values. Few NA values do not have a large effect, but many NA values can lead to wrong estimations of p-values and FDRs. We highly recommend avoiding NA values.

beta numeric between 0 and 1. Corresponds to beta in (Yekutieli and Benjamini 1999).

Details

Larger values of obj$s0 are considered more significant when compared to the empirical distribution. E.g. for calculation of resampling based p-values (with pooled = FALSE) we in principle use \( p.val <- (\text{rowSums(obj$stats} >= \text{obj$s0})+1)/(\text{ncol(obj$stats)}+1) \) according to (Phipson and Smyth 2010).

method = "fdr.q" and method = "fdr.qu" are resampling based fdr estimates and can only be used with pooled = TRUE. method = "fdr.q" is the FDR local estimator and method = "fdr.qu" is the FDR upper limit, see (Reiner et al. 2003; Yekutieli and Benjamini 1999). For all other method arguments resampling based p-values are calculated and passed to stats::p.adjust for p-value adjustment. So these methods provide resampling based p-values with (non-resampling based) p-value adjustment. method = "fdr.q" and method = "fdr.qu" were adapted from package fdrame (Benjamini et al. 2019; Reiner et al. 2003).

When pooled = TRUE, marginal distributions of the test statistics are considered exchangeable for all features. The resampling based p-values of each feature are then calculated from all rotated statistics (all features, all rotations). For these cases, if the number of features is reasonably large, usually only few resamples (argument R in rotateStat) are required. We want to emphasize that in order for the marginal distributions to be exchangeable, the statistics must be a pivotal quantity (i.e. it must be scale independent). Pivotal quantities are e.g. t values. Using e.g. linear models with coef as statistics is questionable if the different features are measured on different scales. The resampled coefficients then have different variances and pooled = TRUE is not applicable. We thus highly recommend using pivotal quantities as statistics in rotateStat if possible.

When pooled = FALSE the resampling based p-values are calculated for each feature separately. This is required if one expects the resampling based statistics to be distributed differently for individual features. For most common applications this should not be the case and the marginal distribution are exchangeable for all features, hence pooled = TRUE by default.

If method = "fdr.q" or method = "fdr.qu" and weights were specified when initialising the random rotation object (see parameter initialised.obj in rotateStat), a warning is displayed. The correlation structure (dependence structure) of linear model coefficients between different features is not generally preserved if different weights are used for different features. Methods fdr.q and fdr.qu rely on preserved correlation structure of dependent statistics and thus should not be used if statistics based on model coefficients (e.g. t statistics of model coefficients) are used in combination with different weights.

P-values and FDRs are calculated for each column of obj$s0 separately.

Value

A numeric matrix of corrected p-values or FDRs with dimension dim(obj$s0).
Author(s)

Peter Hettegger

References


See Also

rotateStat

Examples

# See also '?rotateStat':

#set.seed(0)

# Dataframe of phenotype data (sample information)
# We simulate 2 sample classes processed in 3 batches
pdata <- data.frame(batch = rep(1:3, c(10,10,10)),
                    phenotype = rep(c("Control", "Cancer"), c(5,5)))
features <- 100

# Matrix with random gene expression data
edata <- matrix(rnorm(features * nrow(pdata)), features)
rownames(edata) <- paste("feature", 1:nrow(edata))

mod1 <- model.matrix(~phenotype, pdata)

# Initialisation of the random rotation class
init1 <- initBatchRandrot(Y = edata, X = mod1, coef.h = 2, batch = pdata$batch)

# Definition of the batch effect correction procedure with subsequent calculation
# of two-sided test statistics
statistic <- function(., batch, mod, coef){
    capture.output(suppressMessages(
        Y.tmp <- sva::ComBat(., batch = batch, mod)
    )))
    fit1 <- lm.fit(mod, t(Y.tmp))
    abs(coef(fit1)[coef,])
}

# We calculate test statistics for the second coefficient
res1 <- rotateStat(initialised.obj = init1,
    R = 10,
    statistic = statistic,
    batch = pdata$batch, mod = mod1, coef = 2)

hist(pFdr(res1))

---

**qqunif**

Quantile-Quantile plot of data sample against uniform theoretical quantiles

**Description**

 qqunif produces a QQ plot of the values in ps against the theoretical quantiles of the uniform distribution.

**Usage**

 qqunif(  
 ps,  
 log = "xy",  
 pch = 20,  
 xlab = "theoretical quantiles",  
 ylab = "sample quantiles",  
 plot.it = TRUE,  
 ...  
 )

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>numeric vector of values (e.g. p-values). Values must be between 0 and 1. Values like NA, NaN, Inf etc. produce an error.</td>
</tr>
<tr>
<td>log</td>
<td>character indicating whether axis should be plotted in log scale. Either &quot;&quot;, &quot;x&quot;, &quot;y&quot; or &quot;xy&quot;.</td>
</tr>
<tr>
<td>pch</td>
<td>Point symbol, see par.</td>
</tr>
</tbody>
</table>
randorth

xlab     Label for the x axis.
ylab     Label for the y axis.
plot.it  logical whether the result should be plotted.
...      Graphical parameters forwarded to qqplot

Details

This function can e.g. be used for comparing p-values against the uniform distribution. The log
scale of the x and y axes allow a closer look at low p-values.
This function is a modified version of the examples in the qqnorm documentation page.

Value

A list of x and y coordinates, as in qqplot.

Examples

qqunif(runif(100))

randorth(n, type = c("orthonormal", "unitary"), I.matrix = FALSE)

Arguments

n   numeric of length 1 defining the dimensions of the n x n square matrix.
type Either "orthonormal" or "unitary" defining whether a real orthonormal ma-
trix or a complex unitary matrix should be returned.
I.matrix If TRUE, the identity matrix is returned.

Details

A random orthogonal matrix R is generated in order that t(R) (for "orthonormal") or Conj(t(R))
(for "unitary") equals the inverse matrix of R.
This function was adapted from the pracma package (pracma::randortho).
The random orthogonal matrices are distributed with Haar measure over O(n), where O(n) is the
set of orthogonal matrices of order n. The random orthogonal matrices are basically distributed
"uniformly" in the space of random orthogonal matrices of dimension n x n. See also the Examples
and (Stewart 1980; Mezzadri 2007).
Value

A random orthogonal matrix of dimension \( n \times n \).

Author(s)

Peter Hettegger

References


Examples

```r
# The following example shows the orthogonality of the random orthogonal matrix:
R1 <- randorth(4)
zapsmall(t(R1) %*% R1)

R1 <- randorth(4, "unitary")
zapsmall(Conj(t(R1)) %*% R1)

# The following example shows the distribution of 2-dimensional random orthogonal vectors
# on the unit circle.
tmp1 <- vapply(1:400, function(i)randorth(2)[,1], numeric(2))
plot(t(tmp1), xlab = "x", ylab = "y")
```

---

**randpermut**

*Generate random permutation matrix for n samples*

**Description**

Generate a random permutation matrix for \( n \) samples.

**Usage**

`randpermut(n)`

**Arguments**

`n` Number of samples
Details

This method generates an orthogonal matrix with only one entry in each row and column being 1, all other entries being 0.

Value

A random permutation matrix of dimension $n \times n$

Author(s)

Peter Hettegger

Examples

```r
tmp1 <- randpermut(5)
t(tmp1) %*% t(tmp1)
```

---

### randrot

**Random rotation of initialised object**

Description

Perform random data rotation of a previously initialised object (see `initRandrot`) associated with the null hypothesis $H_0: \beta_{coef.h} = 0$.

Usage

```r
randrot(object, ...)

## S4 method for signature 'initRandrot'
randrot(object, ...)

## S4 method for signature 'initRandrotW'
randrot(object, ...)

## S4 method for signature 'initBatchRandrot'
randrot(object, ...)
```

Arguments

- **object**: An initialised object of class `initRandrot-class` or `initBatchRandrot-class`.
- **...**: Further arguments passed to `randorth`
Details

This function generates a randomly rotated dataset from an initialised randrot object (see initRandrot). See also package vignette for application examples. Only the numerical matrix of rotated data is returned, no design matrix, weights or other info is return for efficiency purposes. Please consider that, if you e.g. use weights or if you use rotateStat, you may need to forward the design matrix X, weights etc. to subsequent analyses. See the example in rotateStat.

Details on the calculation of a rotated dataset are given in initRandrot, (Langsrud 2005) and (Hettegger et al. 2021).

Value

numeric matrix of rotated data under the specified combined null hypothesis.

Author(s)

Peter Hettegger

References


Examples

# For further examples see 'rotateStat' and package vignette.

#set.seed(0)

# Dataframe of phenotype data (sample information)
# We simulate 2 sample classes processed in 3 batches
pdata <- data.frame(batch = rep(1:3, c(10,10,10)),
phenotype = rep(c("Control", "Cancer"), c(5,5)))
features <- 100

# Matrix with random gene expression data
edata <- matrix(rnorm(features * nrow(pdata)), features)
rownames(edata) <- paste("feature", 1:nrow(edata))
mod1 <- model.matrix(~phenotype, pdata)

# Initialisation of the random rotation class
init1 <- initBatchRandrot(Y = edata, X = mod1, coef.h = 2,
batch = pdata$batch)

### Fit model to original data

### Random rotation

### Apply model

### Summary

### Additional analysis

### Conclusion
fit.orig <- lm.fit(mod1, t(edata))
head(t(coef(fit.orig)))

### Fit model to rotated data

edata.rot <- randrot(init1)
fit.rot <- lm.fit(mod1, t(edata.rot))
head(t(coef(fit.rot)))

# Note that the coefficients stay equal if we regress only on the
# non-hypothesis coefficients

mod0 <- model.matrix(~1, pdata)
fit.orig0 <- lm.fit(mod0, t(edata))
fit.rot0 <- lm.fit(mod0, t(edata.rot))
head(t(coef(fit.orig0)))
head(t(coef(fit.rot0)))

---

**randRotation-defunct**  Defunct functions in package `randRotation`

**Description**

These functions are defunct and no longer available.

**Details**

Defunct functions are: df_estimate

---

**rotateStat**  Generate data rotations and calculate statistics on it

**Description**

This function generates rotations of data and calculates the provided statistic on each rotation and the non-rotated (original) data. This is the central function of the package.

**Usage**

rotateStat(  
    initialised.obj,  
    R = 10,  
    statistic,  
    ...,  
    parallel = FALSE,  
    BPPARAM = BiocParallel::bpparam()  
)
Arguments

initialised.obj
An initialised random rotation object as returned by \texttt{initRandrot} and \texttt{initBatchRandrot}.

R
The number of resamples/rotations. Single numeric larger than 1.

statistic
A function which takes a data matrix (same dimensions as \(Y\) - see also \texttt{initRandrot}) as first argument and returns a statistic of interest. Any further arguments are passed to it with the \ldots argument. We highly recommend using pivotal quantities as statistic if possible (see also Details in \texttt{pFdr}). Note that \texttt{pFdr} considers larger values of statistics as more significant, so one-tailed tests may require reversal of the sign and two-tailed tests may require taking absolute values, see Examples. The results of statistic for each resample are finally combined with \texttt{as.matrix} and \texttt{cbind}, so ensure that statistic returns either a vector or a matrix. Results with multiple columns are possible and handled adequately in subsequent functions (e.g. \texttt{pFdr}). Note that statistic must not necessarily be of the same length as \(nrow(Y)\), but can also be e.g. a summary statistic of genes (like in gene set testing).

\ldots
Further named arguments for statistic which are passed unchanged each time it is called. Avoid partial matching to arguments of \texttt{rotateStat}. See also the Examples.

parallel
logical if parallel computation should be performed, see details for use of parallel computing.

BPPARAM
An optional \texttt{BiocParallelParam} instance, see documentation of \texttt{BiocParallel} package of Bioconductor.

Details

The function takes an initialised randrot object (\texttt{initRandrot}) and a function that calculates a statistic on the data. The statistic function thereby takes the a matrix \(Y\) as first argument. Any further arguments are passed to it by \ldots.

Together with \texttt{pFdr}, this function implements the workflow described in (Hettegger et al. 2021).

Be aware that only data is rotated (see also \texttt{randrot}), so any additional information including weights, \(X\) etc. need to be provided to statistic. See also package vignette and Examples.

Parallel processing is implemented with the BiocParallel package of Bioconductor. The default argument \texttt{BiocParallel::bpparam()} for BPPARAM returns the registered default backend. See package documentation for further information and usage options. If parallel = \texttt{TRUE} the function calls in statistic need to be called explicitly with package name and "::". So e.g. calling \texttt{lmFit} from the limma package is done with \texttt{limma::lmFit(...)}, see also the examples in the package vignette.

Value

An object of class \texttt{rotateStat}.

Author(s)

Peter Hettegger
References


Examples

```r
#set.seed(0)

# Dataframe of phenotype data (sample information)
# We simulate 2 sample classes processed in 3 batches
pdata <- data.frame(batch = rep(1:3, c(10,10,10)),
                    phenotype = rep(c("Control", "Cancer"), c(5,5)))
features <- 100

# Matrix with random gene expression data
edata <- matrix(rnorm(features * nrow(pdata)), features)
rownames(edata) <- paste("feature", 1:nrow(edata))

# Initialisation of the random rotation class
init1 <- initBatchRandrot(Y = edata, X = mod1, coef.h = 2, batch = pdata$batch)
init1

# Definition of the batch effect correction procedure with subsequent calculation
# of two-sided test statistics
statistic <- function(., batch, mod, coef){
  # The "capture.output" and "suppressMessages" simply suppress any output
  capture.output(suppressMessages(
    Y.tmp <- sva::ComBat(., batch = batch, mod)
  ))
  fit1 <- lm.fit(mod, t(Y.tmp))
  abs(coef(fit1)[coef,])
}

# We calculate test statistics for the second coefficient
res1 <- rotateStat(initialised.obj = init1,
                    R = 10,
                    statistic = statistic,
                    batch = pdata$batch, mod = mod1, coef = 2)
hist(pFdr(res1))
```

---

**rotateStat-class**  
*Rotated object containing rotated and non-rotated statistics*
**show.initRandrot-method**

**Description**

This list based class contains calculated statistics for the original data (s0) and rotated data (stats). See also `rotateStat`.

**Components**

- **s0** Calculated statistics for original (non-rotated) data as returned by the statistic function (`rotateStat`).
- **stats** List of length `ncol.s` containing statistics on rotated data for each column returned by the statistic function.
- **ncol.s** Number of columns returned by the statistic function.
- **R** Number of resamples/rotations.

**Author(s)**

Peter Hettegger

---

**show,initRandrot-method**

*Show an Object*

**Description**

Display the object by printing structured summary information.

**Usage**

```r
## S4 method for signature 'initRandrot'
show(object)
## S4 method for signature 'initBatchRandrot'
show(object)
## S4 method for signature 'rotateStat'
show(object)
```

**Arguments**

- **object** An object of class `initRandrot-class, initRandrotW-class` or `initBatchRandrot-class`.

**Details**

The show method always displays the original design matrix (X), not the transformed (whitened) versions.

**Value**

`show` returns an invisible `NULL`.
weights,initRandrot-method

Extract model weights

Description

weights is a generic function which extracts fitting weights from objects returned by modeling functions. NOTE: This man page is for the weights S4 generic function defined in the randRotation package.

Usage

## S4 method for signature 'initRandrot'
weights(object, ...)

## S4 method for signature 'initBatchRandrot'
weights(object, ...)

Arguments

object
An object of class initRandrot-class, initRandrotW-class or initBatchRandrot-class.

...  
Kept for compatibility with the default method, see ?stats::weights. For objects defined in package randRotation, this argument is currently not needed.

Value

Weights extracted from the object object. NULL if no weights were specified. See ?stats::weights for the value returned by the default method.

Examples

weights
showMethods("weights")
selectMethod("weights", "ANY") # the default method

X_decomp

Decomposition of the design matrix for random rotation generation

Description

Full QR decomposition of the design matrix X. No argument checks are performed, see Details.

Usage

X_decomp(X = NULL, coef.d = seq_len(ncol(X) - 1))
**Arguments**

- `X` Design matrix as generated by `model.matrix`.
- `coef.d` Non-\(H_0\) coefficients.

**Details**

The design matrix \(X\) is QR decomposed into \(X = X_q X_r\). By performing a full QR decomposition, \(X_q\) is automatically extended to a full basis. \(X_q\) is further split into \(X_d\) and \(X_{he}\), where \(X_d\) corresponds to columns `coef.d` (non-\(H_0\) or non-Null-Hypothesis columns) and \(X_{he}\) correspond to all other columns (\(H_0\) and error columns), see `initRandrot`. No argument checks are performed for reasons of performance as this function is called frequently by `initRandrot` when weights are used. See (Hettegger et al. 2021) and (Langsrud 2005) for further details.

**Value**

A `list` object containing matrices \(X_d\), \(X_{he}\) and rank of the qr decomposition.

**Author(s)**

Peter Hettegger

**References**


**Examples**

```r
design <- cbind(1, rep(0:1, 5))
X_decomp(design)
```
Index

* internal
  .fdr.qu, 3
  initRandrotW, 11
  .fdr.$q(.fdr.qu), 3
  .fdr.qu, 3
  .pFdr(.fdr.qu), 3

BiocParallel::bpparam(), 21
BiocParallelParam, 21

contrastModel, 4, 7, 11

df_estimate (randRotation-defunct), 20
dim, initBatchRandrot-method
  (dim, initRandrot-method), 5
dim, initRandrot-method, 5
dimnames, initBatchRandrot-method
  (dimnames, initRandrot-method), 5
dimnames, initRandrot-method, 5

initBatchRandrot, 9, 21
initBatchRandrot (initRandrot), 6
initBatchRandrot, list-method
  (initRandrot), 6
initBatchRandrot-class, 6
initRandrot, 4–6, 6, 9–11, 18, 19, 21, 25
initRandrot, list-method (initRandrot), 6
initRandrot-class, 10
initRandrotW, 9, 11, 11
initRandrotW-class (initRandrot-class), 10

list, 25

model.matrix, 25

NA, 7, 11

par, 15

.pFdr, 4, 12, 21

pracma::randortho, 16

qqnorm, 16
qqplot, 16
qqunif, 15

random, 16, 18
randperm, 17
randrot, 6, 8–10, 18, 21
randrot, initBatchRandrot-method
  (randrot), 18
randrot, initRandrot-method (randrot), 18
randrot, initRandrotW-method (randrot), 18
randRotation, 24
randRotation (randRotation-package), 2
randRotation-defunct, 20
randRotation-package, 2
rotateStat, 2, 9, 12–14, 19, 20, 21, 23
rotateStat-class, 22

show, initBatchRandrot-method
  (show, initRandrot-method), 23
show, initRandrot-method, 23
show, rotateStat-method
  (show, initRandrot-method), 23
stats::p.adjust, 12, 13

weights, 24
weights, initBatchRandrot-method
  (weights, initRandrot-method), 24
weights, initRandrot-method, 24

X_decomp, 10, 24