Package 'randRotation'

January 23, 2021

Title Random Rotation Methods for High Dimensional Data with Batch Structure

Version 1.2.1

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 7.1.1

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_12

git_last_commit a4d468b

git_last_commit_date 2020-11-15

Date/Publication 2021-01-22

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

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

R topics documented:

`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 (also non-linear) batch effect correction methods.

Details

Please refer to the package vignette for further details on usage and for a "quick start". `rotateStat` is basically the central function of the package.

Author(s)

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

See Also

Useful links:

- https://github.com/phettegger/randRotation
- Report bugs at https://github.com/phettegger/randRotation/issues

`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

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

`df_estimate` *Estimation of degrees of freedom (df) for an arbitrary mapping function*

Description

This function has been deprecated and will be defunct in the next release ! This function estimates the local degrees of freedom (df) of mapped data for an arbitrary mapping function. The estimation is done for a set of selected features.

Usage

```
df_estimate(
  data,
  features = sample(nrow(data), 10),
  mapping,
  ...,
  delta = sqrt(.Machine$double.eps)
)
```

Arguments

`data`	A numerical data matrix.
`features`	Features for which the df should be estimated (default `sample(nrow(data),10)`).
`mapping`	A mapping function that takes a matrix `features` x `samples` dimensions as first argument and returns a matrix of mapped data with the same dimensions. Any further arguments can be passed to `mapping` through `...`.
`...`	Additional arguments passed to `mapping`.
`delta`	A numeric delta for the finite differences (default `sqrt(.Machine$double.eps)`).

Details

The df are estimated as the rank of the local Jacobian matrix. It is thus the rank of the local linear approximation of the mapping function, where linearisation is performed around `data`. The Jacobian matrix `J` for a certain feature `j` is calculated with finite differences:

`data2 <-data`

`data2[j,i] <-data2[j,i] + delta`

`J[,i] = (mapping(data2,...) -mapping(data,...))/delta`

In the current implementation, the rank of `J` is calculated as sum of the singular values of `J`. So for each feature, a SVD decomposition of the `ncol(data)` x `ncol(data)` matrix `J` is calculated.

This function should be considered experimental due to the common numerical issues associated with finite differences and numerical calculation of matrix ranks. So always check results for plausibility.

An estimation of df is generated for each feature specified in `features`.

Value

A named numeric vector of estimated df for each feature. Names correspond to `features`.

Examples

```
#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)

# The limma::removeBatchEffect function is a commonly used function for batch effect correction:
mapping <- function(Y, batch, mod) {
  limma::removeBatchEffect(x = Y, batch = batch, design = mod)
}

#The following 2 lines were commented out, as df_estimate() is deprecated.
#dfs <- df_estimate(edata, features = 1, mapping = mapping, batch = pdata$batch, mod = mod1)
#dfs
```

---

dim,initRandrot-method

Dimensions of an Object

Description

Retrieve the dimensions of an object.

Usage

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

## 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

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

## 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 $H_0 : \beta_{coef.h} = 0$. Basics of rotation tests are found in (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, see (Langsrud 2005). Nomenclature of variables is mainly as in (Langsrud 2005). 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.

coef.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 $H_0 : \beta_{coef.h} = 0$ 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 and Xd, 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 decompostion 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,] = ( solve(T) Rt T (y[i,])* )*` 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 `Rt` is calculated with the same reduced random orthogonal matrix `R` for all features.

When using `initBatchRandrot`, `initRandrot` is called for each batch separately. When using `initBatchRandrot` with `cormat`, `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 `split(seq_along(batch),batch)`.

Value

An initialised `initRandrot`, `initRandrotW` or `initBatchRandrot` object.

Author(s)

Peter Hettegger

References

Langsrud O (2005). "Rotation tests." *Statistics and Computing*, **15**(1), 53–60. ISSN 09603174, doi: 10.1007/s1122200547895, https://doi.org/10.1007/s11222-005-4789-5.

Law CW, Chen Y, Shi W, Smyth GK (2014). "Voom: Precision weights unlock linear model analysis tools for RNA-seq read counts." *Genome Biology*, **15**(2), 1–17. ISSN 1474760X, doi: 10.1186/gb2014152r29, https://doi.org/10.1186/gb-2014-15-2-r29.

See Also

`randrot`, `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
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)
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`, altough some components are changed or added.

Components

The following components are included as list elements:

`X` Original (non-transformed) design matrix.

`Xhe`, `Xhe.Y.w`, `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

pFdr *Calculate resampling based p-values and FDRs*

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

```
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$s0 are considered more significant when compared to the empirical distribution. E.g. for calculation of resampling based p-values (with `pooled = FALSE`) we in principle use `p.val <-(rowSums(obj$stats >= obj$s0)+1)/(ncol(obj$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 calculcated 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 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

Benjamini Y, Kenigsberg E, Reiner A, Yekutieli D (2019). *fdrame: FDR adjustments of Microarray Experiments (FDR-AME)*. R package version 1.56.0.

Phipson B, Smyth GK (2010). "Permutation P-values should never be zero: Calculating exact P-values when permutations are randomly drawn." *Statistical Applications in Genetics and Molecular Biology*, **9**(1). ISSN 15446115, doi: 10.2202/15446115.1585, 1603.05766, https://doi.org/10.2202/1544-6115.1585.

Reiner A, Yekutieli D, Benjamini Y (2003). "Identifying differentially expressed genes using false discovery rate controlling procedures." *Bioinformatics*, **19**(3), 368–375. ISSN 13674803, doi: 10.1093/bioinformatics/btf877, https://doi.org/10.1093/bioinformatics/btf877.

Yekutieli D, Benjamini Y (1999). "Resampling-based false discovery rate controlling multiple test procedures for correlated test statistics." *Journal of Statistical Planning and Inference*, **82**(1-2), 171–196. ISSN 03783758, doi: 10.1016/S03783758(99)000415, https://doi.org/10.1016/S0378-3758(99)00041-5.

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)
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))
```

`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",
```

```
  ...
)
```

Arguments

ps	numeric vector of values (e.g. p-values). Values must be between 0 and 1. Values like NA, NaN, Inf etc. produce an error.
log	character indicating whether axis should be plotted in log scale. Either "", "x", "y" or "xy".
pch	Point symbol, see par.
xlab	Label for the x axis.
ylab	Label for the y axis.
...	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 *Random orthogonal matrix*

Description

Generation of a random orthogonal n x n matrix.

Usage

```
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 matrix 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 x n.

Author(s)

Peter Hettegger

References

Mezzadri F (2007). "How to generate random matrices from the classical compact groups." *Notices of the American Mathematical Society*, **54**(5), 592–604. ISSN 1088-9477, 0609050, http://arxiv.org/abs/0609050.

Stewart GW (1980). "The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators." *SIAM Journal on Numerical Analysis*. ISSN 0036-1429, doi: 10.1137/0717034, https://doi.org/10.1137/0717034.

Examples

```
# 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 methods 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 x n

Author(s)

Peter Hettegger

Examples

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

```
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 datset are given in `initRandrot` and (Langsrud 2005).

Value

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

Author(s)

Peter Hettegger

References

Langsrud O (2005). "Rotation tests." *Statistics and Computing*, **15**(1), 53–60. ISSN 09603174, doi: 10.1007/s1122200547895, https://doi.org/10.1007/s11222-005-4789-5.

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)
init1

### Fit model to original data

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-deprecated` *Deprecated functions in package 'randRotation'*

Description

These functions are provided for compatibility with older versions of 'randRotation' only, and will be defunct at the next release.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- df_estimate: There is no replacement function for 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 initRandrot and 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 initRandrot) as first argument and returns a statistic of interest. Any further arguments are passed to it by .... We highly recommend using pivotal quantities as statistic if possible (see also Details in pFdr). Note that 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 as.matrix and 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. pFdr).
...	Further named arguments for statistic which are passed unchanged each time it is called. Avoid partial matching to arguments of rotateStat. See also the Examples.
parallel	logical if parallel computation should be performed, see details for use of parallel computing.
BPPARAM	An optional BiocParallelParam instance, see documentation of BiocParallel package of Bioconductor.

Details

The function takes an initialised randrot object (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 ....

Be aware that only data is rotated (see also 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 BiocParallel::bpparam() for BPPARAM returns the registered default backend. See package documentation for further information and usage options. If parallel = TRUE the function calls in statistic need to be called explicitly with package name and "::". So e.g. calling lmFit from the limma package is done with limma::lmFit(...), see also the examples in the package vignette.

Value

An object of class rotateStat.

Author(s)

Peter Hettegger

Examples

```
#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)
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*

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

```
## 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-H0 coefficients.

Details

The design matrix X is QR decomposed into X = Xq Xr. By performing a full QR decomposition, Xq is automatically extended to a full basis. Xq is further split into Xd and Xhe, where Xd corresponds to columns coef.d (non-H0 or non-Null-Hypothesis columns) and Xhe correspond to all other columns (H0 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 (Langsrud 2005) for further details.

Value

A list object containing matrices Xd, Xhe and rank of the qr decomposition.

Author(s)

Peter Hettegger

References

Langsrud O (2005). "Rotation tests." *Statistics and Computing*, **15**(1), 53–60. ISSN 09603174, doi: 10.1007/s1122200547895, https://doi.org/10.1007/s11222-005-4789-5.

Examples

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

Index