Package ‘dearseq’

May 1, 2024

Type Package
Title Differential Expression Analysis for RNA-seq data through a robust variance component test
Version 1.16.0
Date 2023-06-16
Depends R (>= 3.6.0)
Imports CompQuadForm, dplyr, ggplot2, KernSmooth, magrittr, matrixStats, methods, patchwork, parallel, pbapply, reshape2, rlang, scattermore, stats, statmod, survey, tibble, viridisLite
Suggests Biobase, BiocManager, BiocSet, edgeR, DESeq2, GEOquery, GSA, knitr, limma, readxl, rmarkdown, S4Vectors, SummarizedExperiment, testthat, covr
License GPL-2 | file LICENSE
biocViews BiomedicalInformatics, CellBiology, DifferentialExpression, DNAsq, GeneExpression, Genetics, GeneSetEnrichment, ImmunoOncology, KEGG, Regression, RNASeq, Sequencing, SystemsBiology, TimeCourse, Transcription, Transcriptomics
BugReports https://github.com/borishejblum/dearseq/issues
Encoding UTF-8
RoxygenNote 7.2.3
VignetteBuilder knitr
Contents

git_url https://git.bioconductor.org/packages/dearseq

git_branch RELEASE_3_19

git_last_commit 7ebc2c8

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-04-30

Author Denis Agniel [aut],
    Boris P. Hejblum [aut, cre] (<https://orcid.org/0000-0003-0646-452X>),
    Marine Gauthier [aut],
    Mélanie Huchon [ctb]

Maintainer Boris P. Hejblum <boris.hejblum@u-bordeaux.fr>

Contents

dearseq-package .................................................. 3
%^% .................................................. 4
baduel_5gs .................................................. 4
dear_seq .................................................. 6
dgsa_seq .................................................. 11
PBT_gmt .................................................. 15
permPvals .................................................. 16
plot.dearseq .................................................. 17
plot_hist_pvals .................................................. 18
plot_ord_pvals .................................................. 19
plot_weights .................................................. 20
sp_weights .................................................. 21
spaghettiPlot1GS ........................................... 23
summary.dearseq .................................................. 24
vc_score .................................................. 25
vc_score_h .................................................. 26
vc_score_h_perm .................................................. 28
vc_score_perm .................................................. 31
vc_test_asym .................................................. 33
vc_test_perm .................................................. 35
voom_weights .................................................. 37

Index 39
Description


Details

Analysis of RNA-seq data with variance component score test accounting for data heteroscedasticity through precision weights. Performs gene-wise analysis as well as gene set analysis, including for complex experimental designs such as longitudinal data.

| Package:  | dearseq |
| Type:     | Package |
| Version:  | 1.13.3  |
| Date:     | 2023-06-16 |
| License:  | GPL-2   |

The two main functions of the dearseq package are dear_seq and dgsa_seq.

Author(s)

Maintainer: Boris P. Hejblum <boris.hejblum@u-bordeaux.fr> (ORCID)
Authors:
  • Denis Agniel <denis.agniel@gmail.com>
  • Marine Gauthier <marine.gauthier@u-bordeaux.fr>

Other contributors:
  • Mélanie Huchon <melanie.huchon@u-bordeaux.fr> [contributor]

References


See Also

Useful links:

- Report bugs at https://github.com/borishejblum/dearseq/issues

---

`%%^`  

\textit{Power for covariances matrices}

---

**Description**

Compute the power of a positive definite symmetric matrix.

**Usage**

\[ x \ %^{\%} \ n \]

**Arguments**

- `x`: A positive definite symmetric matrix.
- `n`: A real number.

**Value**

A matrix of the same dimensions as `x`.

---

**baduel_5gs**  

\textit{Small portion of RNA-seq data from plant physiology study.}

---

**Description**

A subsample of the RNA-seq data from Baduel \textit{et al.} studying Arabidopsis Arenosa physiology.

**Usage**

\[ \text{data(baduel\_5gs)} \]
Format

3 objects

• design: a design matrix for the 48 measured samples, containing the following variables:
  – SampleName corresponding column names from expr_norm_corr
  – Intercept an intercept variable
  – Population a factor identifying the plant population
  – Age_weeks numeric age of the plant at sampling time (in weeks)
  – Replicate a purely technical variable as replicates are not from the same individual over weeks. Should not be used in analysis.
  – Vernalized a logical variable indicating whether the plant had undergone vernalization (exposition to cold and short day photoperiods)
  – Vernalized a binary variable indicating whether the plant belonged to the KA population
  – AgeWeeks_Population interaction variable between the AgeWeeks and Population variables
  – AgeWeeks_Vernalized interaction variable between the AgeWeeks and Vernalized variables
  – Vernalized_Population interaction variable between the Vernalized and Population variables
  – AgeWeeks_Vernalized_Population interaction variable between the AgeWeeks, Vernalized and Population variables

• baduel_gmt: a gmt object containing 5 gene sets of interest (see GSA.read.gmt), which is simply a list with the 3 following components:
  – genesets: a list of \( n \) gene identifiers vectors composing each gene set (each gene set is represented as the vector of the gene identifiers composing it)
  – geneset.names: a vector of length \( n \) containing the gene set names (i.e. gene sets identifiers)
  – geneset.descriptions: a vector of length \( n \) containing gene set descriptions (e.g. textual information on their biological function)

• expr_norm_corr: a numeric matrix containing the normalized batch corrected expression for the 2454 genes included in either of the 5 gene sets of interests

Source


References


Examples

```r
if(interactive()){
  data('baduel_5gs')

  set.seed(54321)
  KA_vs_TBG <- dgsa_seq(exprmat=log2(expr_norm_corr+1),
   covariates=apply(as.matrix(design[, c('Intercept', 'Vernalized', 'AgeWeeks', 'Vernalized_Population', 'AgeWeeks_Population')]), 2, as.numeric),
   variables2test = as.matrix(design[, c('PopulationKA', 'AgeWeeks_Population')]),
   genesets=baduel_gmt$genesets[c(3,5)],
   which_test = 'permutation', which_weights = 'loclin',
   n_perm=1000, preprocessed = TRUE)

  set.seed(54321)
  Cold <- dgsa_seq(exprmat=log2(expr_norm_corr+1),
   covariates=apply(as.matrix(design[, c('Intercept', 'AgeWeeks', 'PopulationKA', 'AgeWeeks_Population')]), 2, as.numeric),
   variables2test=as.matrix(design[, c('Vernalized', 'Vernalized_Population')]),
   genesets=baduel_gmt$genesets[c(3,5)],
   which_test = 'permutation', which_weights = 'loclin',
   n_perm=1000, preprocessed = TRUE)
}
```

---

**Dear_seq**

* Differential expression analysis of RNA-seq data through a variance component test

**Description**

Wrapper function for gene-by-gene association testing of RNA-seq data

**Usage**

```r
dear_seq(
  exprmat = NULL,
  object = NULL,
  covariates = NULL,
  variables2test,
  sample_group = NULL,
  weights_var2test_condi = (which_test != "permutation"),
  cov_variables2test_eff = NULL,
  which_test = c("permutation", "asymptotic"),
  which_weights = c("loclin", "voom", "none"),
)```
n_perm = 1000,  
progressbar = TRUE,  
parallel_comp = TRUE,  
nb_cores = parallel:::detectCores(logical = FALSE) - 1,  
preprocessed = FALSE,  
gene_based_weights = FALSE,  
bw = "nrd",  
kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",  
"tricube", "cosine", "optcosine"),  
transform = TRUE,  
padjust_methods = c("BH", "BY", "holm", "hochberg", "hommel", "bonferroni"),  
lowess_span = 0.5,  
R = NULL,  
adaptive = TRUE,  
max_adaptive = 64000,  
homogen_traj = FALSE,  
na.rm_dearseq = TRUE
)

Arguments

exprmat  a numeric matrix of size $G \times n$ containing the raw RNA-seq counts or preprocessed expressions from $n$ samples for $G$ genes. Default is NULL, in which case object must not be NULL.

object an object that can be either a SummarizedExperiment, an ExpressionSet, a DESeqDataSet, or a DGEList. Default is NULL, in which case exprmat must not be NULL.

covariates
  • If exprmat is specified as a matrix: then covariates must be a numeric matrix of size $n \times p$ containing the model covariates for $n$ samples (design matrix). Usually, its first column is the intercept (full of 1s).
  • If object is specified: then covariates must be a character vector of length $p$ containing the colnames of the design matrix given in object.

If covariates is NULL (the default), then it is just the intercept.

variables2test  
  • If exprmat is specified as a matrix: a numeric design matrix of size $n \times K$ containing the $K$ variables to be tested.
  • If object is specified: then variables2test must be a character vector of length $K$ containing the colnames of the design matrix given in object.

sample_group  a vector of length $n$ indicating whether the samples should be grouped (e.g. paired samples or longitudinal data). Coerced to be a factor. Default is NULL in which case no grouping is performed.

weights_var2test_condi  a logical flag indicating whether heteroscedasticity weights computation should be conditional on both the variables to be tested variables2test and on the covariates, or on covariates alone. Default is TRUE for the asymptotic test (in which case conditional means are estimated conditionally on both variables2test and covariates), and FALSE for the permutation test (in which case conditional means are estimated conditionally on only the covariates).
cov_variables2test_eff

A matrix of size $K \times K$ containing the covariance matrix of the $K$ random effects. Only used if `homogen_traj` is FALSE. Default assume diagonal correlation matrix, i.e. independence of random effects.

which_test

A character string indicating which method to use to approximate the variance component score test, either 'permutation' or 'asymptotic'. Default is 'permutation'.

which_weights

A character string indicating which method to use to estimate the mean-variance relationship weights. Possibilities are 'loclin', 'voom' or 'none' (in which case no weighting is performed). Default is 'loclin'. See `sp_weights` and `voom_weights` for details.

n_perm

The number of perturbations. Default is 1000

progressbar

Logical indicating whether a progressbar should be displayed when computing permutations (only in interactive mode).

parallel_comp

A logical flag indicating whether parallel computation should be enabled. Only Linux and MacOS are supported, this is ignored on Windows. Default is TRUE.

nb_cores

An integer indicating the number of cores to be used when `parallel_comp` is TRUE. Default is `parallel::detectCores(logical=FALSE) - 1`.

preprocessed

A logical flag indicating whether the expression data have already been preprocessed (e.g. log2 transformed). Default is FALSE, in which case `y` is assumed to contain raw counts and is normalized into log(counts) per million.

gene_based_weights

A logical flag used for 'loclin' weights, indicating whether to estimate weights at the gene-level, or rather at the observation-level. Default is FALSE, which is what it should be for gene-wise analysis.

bw

A character string indicating the smoothing bandwidth selection method to use. See `bandwidth` for details. Possible values are 'ucv', 'SJ', 'bcv', 'nrd' or 'nrd0'.

kernel

A character string indicating which kernel should be used. Possibilities are 'gaussian', 'epanechnikov', 'rectangular', 'triangular', 'biweight', 'tricube', 'cosine', 'optcosine'. Default is 'gaussian' (NB: 'tricube' kernel corresponds to the loess method).

transform

A logical flag used for 'loclin' weights, indicating whether values should be transformed to uniform for the purpose of local linear smoothing. This may be helpful if tail observations are sparse and the specified bandwidth gives suboptimal performance there. Default is TRUE.

padjust_methods

Multiple testing correction method used if `genesets` is a list. Default is 'BH', i.e. Benjamini-Hochberg procedure for controlling the FDR. Other possibilities are: 'holm', 'hochberg', 'hommel', 'bonferroni' or 'BY' (for Benjamini-Yekutieli procedure).

lowess_span

Smoother span for the lowess function, between 0 and 1. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. Only used if `which_weights` is 'voom'. Default is 0.5.
dear_seq

R

library.size (optional, important to provide if preprocessed = TRUE). Default is NULL

adaptive

a logical flag indicating whether adaptive permutation should be performed. Default is TRUE

max_adaptive

The maximum number of permutations considered. Default is 64000

homogen_traj

a logical flag indicating whether trajectories should be considered homogeneous. Default is FALSE in which case trajectories are not only tested for trend, but also for heterogeneity.

na.rm_dearseq

logical: should missing values in y (including NA and NaN) be omitted from the calculations? Default is TRUE.

Value

A list with the following elements:

- which_test: a character string carrying forward the value of the 'which_test' argument indicating which test was perform (either 'asymptotic' or 'permutation').
- preprocessed: a logical flag carrying forward the value of the 'preprocessed' argument indicating whether the expression data were already preprocessed, or were provided as raw counts and transformed into log-counts per million.
- n_perm: an integer carrying forward the value of the 'n_perm' argument indicating the number of perturbations performed (NA if asymptotic test was performed).
- genesets: carrying forward the value of the 'genesets' argument defining the gene sets of interest (NULL for gene-wise testing).
- pval: computed p-values. A data.frame with one raw for each each gene set, or for each gene if genesets argument is NULL, and with 2 columns: the first one 'rawPval' contains the raw p-values, the second one contains the FDR adjusted p-values (according to the 'padjust_methods' argument) and is named 'adjPval'.

References


See Also

sp_weights vc_test_perm vc_test_asym p.adjust

Examples

#Monte-Carlo estimation of the proportion of DE genes over `nsims`
#simulations under the null

#number of runs
nsims <- 2 #100
res <- numeric(nsims)
for(i in 1:nsims){

}
n <- 1000 # number of genes
nr <- 5 # number of measurements per subject (grouped data)
ni <- 50 # number of subjects
r <- nr * ni # number of measurements
t <- matrix(rep(1:nr), ni, ncol = 1, nrow = r) # the variable to be tested
sigma <- 0.5
b0 <- 1
b1 <- 0

# under the null:

# create the matrix of gene expression
y.tilde <- b0 + b1 * t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma * abs(y.tilde)))))
    ncol = n, nrow = r) + matrix(rep(y.tilde, n), ncol = n, nrow = r)

# no covariates
x <- matrix(1, ncol = 1, nrow = r)

# run test
# asymptotic test with preprocessed grouped data
res_genes <- dear_seq(exprmat = y, covariates = x, variables2test = t,
                     sample_group = rep(1:ni, each = nr),
                     which_test = 'asymptotic',
                     which_weights = 'none', preprocessed = TRUE)

# proportion of raw p-values > 0.05
mean(res_genes$pvals[, 'rawPval'] > 0.05)

# quantiles of raw p-values
quantile(res_genes$pvals[, 'rawPval'])

# proportion of raw p-values < 0.05 i.e. proportion of DE genes
res[i] <- mean(res_genes$pvals[, 'rawPval'] < 0.05)
message(i)
}

# results
mean(res)

if(interactive()){
b0 <- 1
# under the null:
b1 <- 0

# create the matrix of gene expression
y.tilde <- b0 + b1 * t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma * abs(y.tilde))))
    ncol = n, nrow = r) + matrix(rep(y.tilde, n), ncol = n, nrow = r)

# run test
# asymptotic test with preprocessed grouped data
res_genes <- dear_seq(exprmat = y, covariates = x, variables2test = t,
dgsa_seq

sample_group=rep(1:ni, each=nr),
which_weights='none', preprocessed=TRUE)

#results
summary(res_genes$pvals)
}

dgsa_seq  

Time-course Gene Set Analysis

Description

Wrapper function for performing gene set analysis of (potentially longitudinal) RNA-seq data

Usage

dgsa_seq(
  exprmat = NULL,
  object = NULL,
  covariates = NULL,
  variables2test,
  weights_var2test_condi = (which_test != "permutation"),
  genesets,
  sample_group = NULL,
  cov_variables2test_eff = NULL,
  which_test = c("permutation", "asymptotic"),
  which_weights = c("loclin", "voom", "none"),
  n_perm = 1000,
  progressbar = TRUE,
  parallel_comp = TRUE,
  nb_cores = parallel::detectCores(logical = FALSE) - 1,
  preprocessed = FALSE,
  gene_based_weights = TRUE,
  bw = "nrd",
  kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",
              "tricube", "cosine", "optcosine"),
  transform = TRUE,
  padjust_methods = c("BH", "BY", "holm", "hochberg", "hommel", "bonferroni"),
  lowess_span = 0.5,
  R = NULL,
  adaptive = TRUE,
  max_adaptive = 64000,
  homogen_traj = FALSE,
  na.rm_gsa_seq = TRUE,
  verbose = TRUE
)
Arguments

exprmat a numeric matrix of size $G \times n$ containing the raw RNA-seq counts or preprocessed expressions from $n$ samples for $G$ genes. Default is NULL, in which case object must not be NULL.

object an object that can be either a `SummarizedExperiment`, an `ExpressionSet`, a `DESeqDataSet`, or a `DGEList`. Default is NULL, in which case exprmat must not be NULL.

covariates
- If exprmat is specified as a matrix: then covariates must be a numeric matrix of size $n \times p$ containing the model covariates for $n$ samples (design matrix). Usually, its first column is the intercept (full of 1s).
- If object is specified: then covariates must be a character vector of length $p$ containing the colnames of the design matrix given in object.

variables2test
- If exprmat is specified as a matrix: a numeric design matrix of size $n \times K$ containing the $K$ variables to be tested.
- If object is specified: then variables2test must be a character vector of length $K$ containing the colnames of the design matrix given in object.

weights_var2test_condi a logical flag indicating whether heteroscedasticity weights computation should be conditional on both the variable(s) to be tested $\phi$ and on covariate(s) $x$, or on $x$ alone. Default is TRUE for the asymptotic test (in which case conditional means are estimated conditionally on both variables2test and covariates), and FALSE for the permutation test (in which case conditional means are estimated conditionally on only the covariates).

genesets Can be either:
- a vector
- a list
- a BiocSet object

Can be a vector of index or subscripts that defines which rows of $y$ constitute the investigated gene set (when only 1 gene set is being tested). Can also be a list of index (or rownames of $y$) when several gene sets are tested at once, such as the first element of a `gmt` object. Finally, can also be a `BiocSet` object. If NULL, then gene-wise p-values are returned.

sample_group a vector of length $n$ indicating whether the samples should be grouped (e.g. paired samples or longitudinal data). Coerced to be a factor. Default is NULL in which case no grouping is performed.

cov_variables2test_eff a matrix of size $K \times K$ containing the covariance matrix of the $K$ random effects. Only used if homogen_traj is FALSE. Default assume diagonal correlation matrix, i.e. independence of random effects.

which_test a character string indicating which method to use to approximate the variance component score test, either 'permutation' or 'asymptotic'. Default is 'permutation'.

...
which_weights  a character string indicating which method to use to estimate the mean-variance relationship weights. Possibilities are 'loclin', 'voom' or 'none' (in which case no weighting is performed). Default is 'loclin'. See sp_weights and voom_weights for details.

n_perm  the number of perturbations. Default is 1000.

progressbar  logical indicating whether a progressBar should be displayed when computing permutations (only in interactive mode).

parallel_comp  a logical flag indicating whether parallel computation should be enabled. Only Linux and MacOS are supported, this is ignored on Windows. Default is TRUE.

nb_cores  an integer indicating the number of cores to be used when parallel_comp is TRUE. Default is parallel::detectCores(logical=FALSE) - 1.

preprocessed  a logical flag indicating whether the expression data have already been preprocessed (e.g. log2 transformed). Default is FALSE, in which case y is assumed to contain raw counts and is normalized into log(counts) per million.

gene_based_weights  a logical flag used for 'loclin' weights, indicating whether to estimate weights at the gene-level, or rather at the observation-level. Default is TRUE, and weights are then estimated at the gene-level.

bw  a character string indicating the smoothing bandwidth selection method to use. See bandwidth for details. Possible values are 'ucv', 'SJ', 'bcv', 'nrd' or 'nrd0'

kernel  a character string indicating which kernel should be used. Possibilities are 'gaussian', 'epanechnikov', 'rectangular', 'triangular', 'biweight', 'tricube', 'cosine', 'optcosine'. Default is 'gaussian' (NB: 'tricube' kernel corresponds to the loess method).

transform  a logical flag used for 'loclin' weights, indicating whether values should be transformed to uniform for the purpose of local linear smoothing. This may be helpful if tail observations are sparse and the specified bandwidth gives suboptimal performance there. Default is TRUE.

padjust_methods  multiple testing correction method used if genesets is a list. Default is 'BH', i.e. Benjamini-Hochberg procedure for controlling the FDR. Other possibilities are: 'holm', 'hochberg', 'hommel', 'bonferroni' or 'BY' (for Benjamini-Yekutieli procedure).

lowess_span  smoother span for the lowess function, between 0 and 1. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. Only used if which_weights is 'voom'. Default is 0.5.

R  library size (optional, important to provide if preprocessed = TRUE). Default is NULL

adaptive  a logical flag indicating whether adaptive permutation should be performed. Default is TRUE

max_adaptive  The maximum number of permutations considered. Default is 64000
homogen_traj  a logical flag indicating whether trajectories should be considered homogeneous. Default is FALSE in which case trajectories are not only tested for trend, but also for heterogeneity.

na.rm_gscaleq  logical: should missing values in y (including NA and NaN) be omitted from the calculations? Default is TRUE.

verbose  logical: should informative messages be printed during the computation? Default is TRUE.

Value

A list with the following elements:

- which_test: a character string carrying forward the value of the 'which_test' argument indicating which test was perform (either 'asymptotic' or 'permutation').
- preprocessed: a logical flag carrying forward the value of the 'preprocessed' argument indicating whether the expression data were already preprocessed, or were provided as raw counts and transformed into log-counts per million.
- n_perm: an integer carrying forward the value of the 'n_perm' argument indicating the number of perturbations performed (NA if asymptotic test was performed).
- genesets: carrying forward the value of the 'genesets' argument defining the gene sets of interest (NULL for gene-wise testing).
- pval: computed p-values. A data.frame with one raw for each each gene set, or for each gene if genesets argument is NULL, and with 2 columns: the first one 'rawPval' contains the raw p-values, the second one contains the FDR adjusted p-values (according to the 'padjust_methods' argument) and is named 'adjPval'.

References


See Also

sp_weights vc_test_perm vc_test_asym p.adjust

Examples

nsims <- 2 #100
res_quan <- list()
for(i in 1:2){
  n <- 200000
  nr <- 3
  r <- nr*20 #4*nr#100*nr
  t <- matrix(rep(1:nr), r/nr, ncol=1, nrow=r)
  sigma <- 0.4
  b0 <- 1
# under the null:
b1 <- 0

y.tilde <- b0 + b1*t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
    matrix(rep(y.tilde, n), ncol=n, nrow=r))
x <- matrix(1, ncol=1, nrow=r)

# run test
res <- dgsa_seq(exprmat = y, covariates = x, variables2test = t,
    genesets=lapply(0:9, function(x){x*10+(1:10)}),
    cov_variables2test_eff = matrix(1),
    sample_group = rep(1:(r/nr), each=nr),
    which_test='asymptotic',
    which_weights='none', preprocessed=TRUE)
res_genes <- dgsa_seq(exprmat = y, covariates = x,
    variables2test = cbind(t),#, rnorm(r), #t^2
    genesets = NULL,
    cov_variables2test_eff = diag(1),
    sample_group = rep(1:(r/nr), each=nr),
    which_test = 'asymptotic',
    which_weights = 'none', preprocessed = TRUE)
length(res_genes$pvals[, 'rawPval'])
quantile(res_genes$pvals[, 'rawPval'])
res_quant[i] <- res_genes$pvals[, 'rawPval']

# round(rowMeans(vapply(res_quant, FUN = quantile, FUN.VALUE = rep(1.1, 5)), 3)
# plot(density(unlist(res_quant)))
# mean(unlist(res_quant)<0.05)
if(interactive()){
    res_genes <- dgsa_seq(exprmat = y, covariates = x, variables2test = t,
        genesets = NULL,
        cov_variables2test_eff = matrix(1),
        sample_group = rep(1:(r/nr), each=nr),
        which_test = 'permutation',
        which_weights = 'none', preprocessed = TRUE,
        n_perm = 1000, parallel_comp = FALSE)
    mean(res_genes$pvals$rawPval < 0.05)
    summary(res_genes$pvals$adjPval)
}

PBT_gmt

PBT gene sets related to kidney transplant

Description

9 Pathogenesis Based Transcripts (PBT) gene sets specifically related to kidney transplant
Usage

data(PBT_gmt)

Format

a gmt object containing 9 gene sets specific to kidney transplant (see GSA.read.gmt), which is simply a list with the 3 following components:

- genesets: a list of \( n \) gene identifiers vectors composing each gene set (each gene set is represented as the vector of the gene identifiers composing it)
- geneset.names: a vector of length \( n \) containing the gene set names (i.e. gene sets identifiers)
- geneset.descriptions: a vector of length \( n \) containing gene set descriptions (e.g. textual information on their biological function)

Source

http://atagc.med.ualberta.ca/Research/GeneLists

References


Examples

data('PBT_gmt')
PBT_gmt

permPvals

Exact permutation p-values

Description

Calculates exact p-values for permutation tests when permutations are randomly drawn with replacement. This implementation is based on (slightly adapted) the implementation by Belinda Phipson and Gordon Smyth from the R package statmod

Usage

permPvals(nPermSupObs, nPermEff, totalPossibleNPerm)
Arguments

nPermSupObs  number of permutations that yielded test statistics at least as extreme as the observed data. Can be a vector or an array of values.
nPermEff    number of permutations effectively computed.
totalPossibleNPerm  total number of permutations possible.

Value

a vector (or an array, similar to nperm_supobs) of exact p-values

Author(s)

Belinda Phipson and Gordon Smyth (adapted by Boris Hejblum)

References


See Also

statmod::permp

Examples

permPvals(10, 100, 1000)

---

**plot.dearseq**  
*Plot method for dearseq objects*

Description

Plot method for dearseq objects

Usage

```r
## S3 method for class 'dearseq'
plot(x, signif_threshold = 0.05, ...)
```
**Arguments**

- `x` an object of class `dear_seq`
- `signif_threshold` a value between 0 and 1 specifying the nominal significance threshold. Default is 0.05.
- `...` further arguments

**Value**

a `ggplot` object

**Author(s)**

Boris Hejblum

---

**plot_hist_pvals**  
*Plotting raw p-values histogram*

**Description**

Display the histogram of raw p-values for diagnostic plots

**Usage**

```
plot_hist_pvals(pvals, binwidth = 0.02)
```

**Arguments**

- `pvals` a vector of raw p-values
- `binwidth` a value specifying the width of the histogram bins. Default is 0.02.

**Value**

a `ggplot` object

**Author(s)**

Boris Hejblum
plot_ord_pvals

Examples

```r
#generate fake data
n <- 1000 #number of genes
nr=5 #number of measurements per subject (grouped data)
ni=50 #number of subjects
r <- nr*ni #number of measurements
t <- matrix(rep(1:nr), ni, ncol=1, nrow=r) # the variable to be tested
sigma <- 0.5
x <- matrix(1, ncol=1, nrow=r) #no covariates only intercept
y.tilde <- rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
     matrix(rep(y.tilde, n), ncol=n, nrow=r))

#Run dear_seq()
res_genes <- dear_seq(exprmat=y, covariates=x, variables2test=t,
sample_group=rep(1:ni, each=nr),
which_test = "asymptotic",
which_weights='none', preprocessed=TRUE)

#Plot
plot_hist_pvals(res_genes$pvals$rawPval)
```

plot_ord_pvals

### Plot of gene-wise p-values

Description

This function prints the sorted exact p-values along with the Benjamini-Hochberg limit and the 5

Usage

```r
plot_ord_pvals(pvals, signif_threshold = 0.05)
```

Arguments

- **pvals**: a vector of length n containing the raw p-values for each gene
- **signif_threshold**: a value between 0 and 1 specifying the nominal significance threshold. Default is 0.05.

Value

A plot of sorted gene-wise p-values
Examples

```r
#generate fake data
n <- 1000 #number of genes
nr=5 #number of measurements per subject (grouped data)
ni=50 #number of subjects
r <- nr*ni #number of measurements
t <- matrix(rep(1:nr), ni, ncol=1, nrow=r) # the variable to be tested
sigma <- 0.5
x <- matrix(1, ncol=1, nrow=r) #no covariates only intercept
y.tilde <- rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
     matrix(rep(y.tilde, n), ncol=n, nrow=r))

#Run dear_seq()
res_genes <- dear_seq(exprmat=y, covariates=x, variables2test=t,
sample_group=rep(1:ni, each=nr),
which_test = "asymptotic",
which_weights='none', preprocessed=TRUE)

#Plot
plot_ord_pvals(res_genes$pvals$rawPval)
```

---

**plot_weights**

*Plotting mean-variance fit for precision weights estimation*

**Description**

Display the variability with respect to the level of expression and the associated smoothed estimation of precision weights accounting for heteroscedasticity.

**Usage**

```r
plot_weights(x)
```

**Arguments**

- **x**
  - a list (such as outputed by the functions `sp_weights` or `voom_weights`) containing the following components:
    - weights: a matrix $n \times G$ containing the estimated precision weights
    - plot_utilities: a list containing the following elements:
      - reverse_trans: a function encoding the reverse function used for smoothing the observations before computing the weights
      - method: the weight computation method (either "voom" or "loclin")
      - smth: the vector of the smoothed values computed
      - gene_based: a logical indicating whether the computed weights are based on average at the gene level or on individual observations
      - mu: the transformed observed counts or averages
      - v: the observed variability estimates
sp_weights

Value

a ggplot object

Author(s)

Boris Hejblum

Examples

G <- 10000
n <- 12
p <- 2
y <- sapply(1:n, FUN = function(x){rnbinom(n = G, size = 0.07, mu = 200)})
x <- sapply(1:p, FUN = function(x){rnorm(n = n, mean = n, sd = 1)})

if(interactive()){
  w <- sp_weights(y, x, use_phi=FALSE, na.rm = TRUE, gene_based=TRUE)
  plot_weights(w)
  vw <- voom_weights(y, x)
  plot_weights(vw)
}

sp_weights Non parametric local heteroscedasticity weights

Description

Computes precision weights that account for heteroscedasticity in RNA-seq count data based on non-parametric local linear regression estimates.

Usage

sp_weights(
  y,
  x,
  phi = NULL,
  use phi = TRUE,
  preprocessed = FALSE,
  gene_based = FALSE,
  bw = c("nrd", "ucv", "SJ", "nrd0", "bcv"),
  kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",
             "tricube", "cosine", "optcosine"),
  transform = TRUE,
  verbose = TRUE,
  na.rm = FALSE
)
**Arguments**

- **y**: a numeric matrix of size $G \times n$ containing the raw RNA-seq counts or preprocessed expression from $n$ samples for $G$ genes.
- **x**: a numeric matrix of size $n \times p$ containing the model covariate(s) from $n$ samples (design matrix).
- **phi**: a numeric design matrix of size $n \times K$ containing the $K$ variable(s) of interest (e.g., bases of time).
- **use_phi**: a logical flag indicating whether conditional means should be conditioned on phi and on covariate(s) $x$, or on $x$ alone. Default is TRUE in which case conditional means are estimated conditionally on both $x$ and $phi$.
- **preprocessed**: a logical flag indicating whether the expression data have already been preprocessed (e.g., log2 transformed). Default is FALSE, in which case $y$ is assumed to contain raw counts and is normalized into log(counts) per million.
- **gene_based**: a logical flag indicating whether to estimate weights at the gene-level. Default is FALSE, when weights will be estimated at the observation-level.
- **bw**: a character string indicating the smoothing bandwidth selection method to use. See `bandwidth` for details. Possible values are 'ucv', 'SJ', 'bcv', 'nrd' or 'nrd0'. Default is 'nrd'.
- **kernel**: a character string indicating which kernel should be used. Possibilities are 'gaussian', 'epanechnikov', 'rectangular', 'triangular', 'biweight', 'tricube', 'cosine', 'optcosine'. Default is 'gaussian' (NB: 'tricube' kernel corresponds to the loess method).
- **transform**: a logical flag indicating whether values should be transformed to uniform for the purpose of local linear smoothing. This may be helpful if tail observations are sparse and the specified bandwidth gives suboptimal performance there. Default is TRUE.
- **verbose**: a logical flag indicating whether informative messages are printed during the computation. Default is TRUE.
- **na.rm**: logical: should missing values (including NA and NaN) be omitted from the calculations? Default is FALSE.

**Value**

A list containing the following components:

- **weights**: a matrix $n \times G$ containing the computed precision weights
- **plot_utilities**: a list containing the following elements:
  - **reverse_trans**: a function encoding the reverse function used for smoothing the observations before computing the weights
  - **method**: the weight computation method ("loclin")
  - **smth**: the vector of the smoothed values computed
  - **gene_based**: a logical indicating whether the computed weights are based on average at the gene level or on individual observations
  - **mu**: the transformed observed counts or averages
  - **v**: the observed variability estimates
**Description**

Spaghetti plot for Specific Gene Set

**Usage**

```r
spaghettiPlot1GS(gs_index, gmt, expr_mat, design, var_time, var_indiv, sampleIdColname, var_group = NULL, var_subgroup = NULL, plotChoice = c("Medians", "Individual"), loess_span = 0.75)
```
Summary method for dearseq objects

Arguments

- **gs_index**: index of the specific gene set in `gmt`.
- **gmt**: a list of elements: `geneset`, `geneset.name` and `geneset.description` (see `GSA.read.gmt`).
- **expr_mat**: a data.frame with numerics of size `G x n` containing the raw RNA-seq counts from `n` samples for `G` genes.
- **design**: a data.frame or DFrame containing the information of each sample (SampleID).
- **var_time**: the time or visit variable contained in `design`.
- **var_indiv**: the patient variable contained in `design`.
- **sampleIdColname**: a character string indicating the name of the sample ID variable in `design` to be matched with the colnames of `expr_mat`.
- **var_group**: a group variable in `design` data to divide into two facets. Default is `NULL`.
- **var_subgroup**: a subgroup variable in `design` data to add 2 curves on plot for each subgroup. Default is `NULL`.
- **plotChoice**: to choose which type of plot (either "Medians", "Individual" or both). Default is `c("Medians", "Individual")`.
- **loess_span**: smoothing span. Default is `0.75`.

Value

a ggplot2 plot object

Examples

data(baduel_5gs)
design$Indiv <- design$Population:design$Replicate
design$Vern <- ifelse(design$Vernalized, "Vernalized", "Non-vernalized")

library(ggplot2)
spaghetti1GS(gs_index = 3, gmt = baduel_gmt, expr_mat = log2(expr_norm_corr+1),
  design = design, var_time = AgeWeeks, var_indiv = Indiv,
  sampleIdColname = "sample", var_group=Vern, var_subgroup=Population,
  plotChoice = "Medians", loess_span= 1.5) +
xlab("Age (weeks)") + guides(color = "none")

---

**summary.dearseq**

Summary method for dearseq objects

**Description**

Summary method for dearseq objects
vc_score

Usage

```r
## S3 method for class 'dearseq'
summary(object, signif_threshold = 0.05, ...)

## S3 method for class 'summary.dearseq'
print(x, ...)
```

Arguments

- `object`: an object of class `dear_seq`
- `signif_threshold`: a value between 0 and 1 specifying the nominal significance threshold. Default is 0.05.
- `...`: further arguments
- `x`: an object of class `'summary.dearseq'`.

Value

- a list

Author(s)

- Boris Hejblum

---

vc_score  

*Computes variance component score test statistics*

Description

This function computes the variance component score test statistics

Usage

```r
vc_score(y, x, indiv, phi, w, Sigma_xi = diag(ncol(phi)), na_rm = FALSE)
```

Arguments

- `y`: a numeric matrix of dim g x n containing the raw RNA-seq counts for g genes from n samples.
- `x`: a numeric design matrix of dim n x p containing the p covariates to be adjusted for.
- `indiv`: a vector of length n containing the information for attributing each sample to one of the studied individuals. Coerced to be a factor.
- `phi`: a numeric design matrix of size n x K containing the K variables to be tested
- `w`: a vector of length n containing the weights for the n samples.
Sigma_xi

a matrix of size $K \times K$ containing the covariance matrix of the $K$ random effects on $\phi$.

na_rm

logical: should missing values (including NA and NaN) be omitted from the calculations? Default is FALSE.

Value

A list with the following elements:

- score: approximation of the set observed score
- q: observation-level contributions to the score
- q_ext: pseudo-observations used to compute the covariance, taking into account the contributions of OLS estimates
- gene_scores_unscaled: a vector of the approximations of the individual gene scores

Examples

```r
set.seed(123)

## generate some fake data
n <- 100
r <- 12
t <- matrix(rep(1:3), r/3, ncol=1, nrow=r)
sigma <- 0.4
b0 <- 1

# under the null:
b1 <- 0
# under the alternative:
b1 <- 0.7
y.tilde <- b0 + b1*t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
       matrix(rep(y.tilde, n), ncol=n, nrow=r))
x <- matrix(1, ncol=1, nrow=r)

# run test
scoreTest <- vc_score(y, x, phi=t, w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
                       Sigma_xi=matrix(1), indiv=rep(1:(r/3), each=3))

scoreTest$score
```

---

**vc_score_h**

Computes variance component score test statistic for homogeneous trajectories

**Description**

This function computes the variance component score test statistics for homogeneous trajectories.
Usage

vc_score_h(y, x, indiv, phi, w, Sigma_xi = diag(ncol(phi)), na_rm = FALSE)

Arguments

y          a numeric matrix of dim g x n containing the raw or normalized RNA-seq counts
           for g genes from n samples.
x          a numeric design matrix of dim n x p containing the p covariates to be adjusted
           for.
indiv      a vector of length n containing the information for attributing each sample to
           one of the studied individuals. Coerced to be a factor.
phi        a numeric design matrix of size n x K containing the K longitudinal variables to
           be tested (typically a vector of time points or functions of time).
w          a vector of length n containing the weights for the n samples, corresponding to
           the inverse of the diagonal of the estimated covariance matrix of y.
Sigma_xi   a matrix of size K x K containing the covariance matrix of the K random effects
           corresponding to phi.
na_rm       logical: should missing values (including NA and NaN) be omitted from the cal-
           culations? Default is FALSE.

Value

A list with the following elements:

- score: approximation of the set observed score
- q: observation-level contributions to the score
- q_ext: pseudo-observations used to compute covariance taking into account the contributions
  of OLS estimates
- gene_scores: approximation of the individual gene scores

Examples

set.seed(123)

##generate some fake data
ng <- 100
nindiv <- 30
nt <- 5
nsample <- nindiv*nt
tim <- matrix(rep(1:nt), nindiv, ncol=1, nrow=nsample)
tim2 <- tim^2
sigma <- 5
b0 <- 10

#under the null:
beta1 <- rnorm(n=ng, 0, sd=0)
# under the (heterogen) alternative:
beta1 <- rnorm(n=ng, 0, sd=0.1)
# under the (homogen) alternative:
beta1 <- rnorm(n=ng, 0.06, sd=0)

y.tilde <- b0 + rnorm(ng, sd = sigma)
y <- t(matrix(rep(y.tilde, nsample), ncol=ng, nrow=nsample, byrow=TRUE) * 
    matrix(rep(beta1, each=nsample), ncol=ng, nrow=nsample, byrow=FALSE) * 
    matrix(rep(tim, ng), ncol=ng, nrow=nsample, byrow=FALSE) + 
    matrix(rep(beta1, each=nsample), ncol=ng, nrow=nsample, byrow=FALSE) * 
    matrix(rnorm(ng*nsample, sd = sigma), ncol=ng, nrow=nsample, byrow=FALSE)
)
myindiv <- rep(1:nindiv, each=nt)
x <- cbind(1, myindiv/2==floor(myindiv/2))
myw <- matrix(rnorm(nsample*ng, sd=0.1), ncol=nsample, nrow=ng)

# run test
score_homogen <- vc_score_h(y, x, phi=tim, indiv=myindiv,
    w=myw, Sigma_xi=cov(tim))
score_homogen$score

score_heterogen <- vc_score(y, x, phi=tim, indiv=myindiv,
    w=myw, Sigma_xi=cov(tim))
score_heterogen$score

scoreTest_homogen <- vc_test_asym(y, x, phi=tim, indiv=rep(1:nindiv, each=nt),
    w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
    Sigma_xi=cov(tim),
    homogen_traj = TRUE)
scoreTest_homogen$set_pval

scoreTest_heterogen <- vc_test_asym(y, x, phi=tim, indiv=rep(1:nindiv, each=nt),
    w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
    Sigma_xi=cov(tim),
    homogen_traj = FALSE)
scoreTest_heterogen$set_pval

---

vc_score_h_perm

Computes variance component score test statistics for homogeneous trajectory and its permuted distribution

**Description**

This function computes the variance component score test statistics for homogeneous trajectories along with its permuted values for estimating its distribution under the null hypothesis.
vc_score_h_perm

Usage
vc_score_h_perm(
  y,
  x,
  indiv,
  phi,
  w,
  Sigma_xi = diag(ncol(phi)),
  na_rm = FALSE,
  n_perm = 1000,
  progressbar = TRUE,
  parallel_comp = TRUE,
  nb_cores = parallel::detectCores(logical = FALSE) - 1
)

Arguments

y  a numeric matrix of dim g x n containing the raw or normalized RNA-seq counts for g genes from n samples.

x  a numeric design matrix of dim n x p containing the p covariates to be adjusted for.

indiv a vector of length n containing the information for attributing each sample to one of the studied individuals. Coerced to be a factor.

phi a numeric design matrix of size n x K containing the K longitudinal variables to be tested (typically a vector of time points or functions of time).

w  a vector of length n containing the weights for the n samples, corresponding to the inverse of the diagonal of the estimated covariance matrix of y.

Sigma_xi a matrix of size K x K containing the covariance matrix of the K random effects corresponding to phi.

na_rm logical: should missing values (including NA and NaN) be omitted from the calculations? Default is FALSE.

n_perm the number of permutation to perform. Default is 1000.

progressbar logical indicating whether a progressBar should be displayed when computing permutations (only in interactive mode).

parallel_comp a logical flag indicating whether parallel computation should be enabled. Only Linux and MacOS are supported, this is ignored on Windows. Default is TRUE.

nb_cores an integer indicating the number of cores to be used when parallel_comp is TRUE. Default is parallel::detectCores(logical=FALSE) - 1.

Value
A list with the following elements:

- score: an approximation of the observed set score
- scores_perm: a vector containing the permuted set scores
- gene_scores_unscaled: approximation of the individual gene scores
- gene_scores_unscaled_perm: a list of approximation of the permuted individual gene scores
Examples

```r
set.seed(123)

# generate some fake data
ng <- 100
nindiv <- 30
nt <- 5
nsample <- nindiv*nt
tim <- matrix(rep(1:nt), nindiv, ncol=1, nrow=nsample)
tim2 <- tim^2
sigma <- 5
b0 <- 10

# under the null:
beta1 <- rnorm(n=ng, 0, sd=0)
# under the (heterogen) alternative:
beta1 <- rnorm(n=ng, 0, sd=0.1)
# under the (homogen) alternative:
beta1 <- rnorm(n=ng, 0.06, sd=0)

y.tilde <- b0 + rnorm(ng, sd = sigma)
y <- t(matrix(rep(y.tilde, nsample), ncol=ng, nrow=nsample, byrow=TRUE) +
    matrix(rep(beta1, each=nsample), ncol=ng, nrow=nsample, byrow=FALSE) *
    matrix(rep(tim, ng), ncol=ng, nrow=nsample, byrow=FALSE) +
    matrix(rep(tim2, ng), ncol=ng, nrow=nsample, byrow=FALSE) +
    matrix(rnorm(ng*nsample, sd = sigma), ncol=ng, nrow=nsample,
        byrow=FALSE))

myindiv <- rep(1:nindiv, each=nt)
x <- cbind(1, myindiv/2==floor(myindiv/2))
myw <- matrix(rnorm(nsample*ng, sd=0.1), ncol=nsample, nrow=ng)

# run test
# We only use few permutations (10) to keep example running time low
# Otherwise one can use n_perm = 1000
score_homogen <- vc_score_h_perm(y, x, phi=tim, indiv=myindiv,
    w=myw, Sigma_xi=cov(tim), n_perm = 10,
    parallel_comp = FALSE)
score_homogen$score

score_heterogen <- vc_score_perm(y, x, phi=tim, indiv=myindiv,
    w=myw, Sigma_xi=cov(tim), n_perm = 10,
    parallel_comp = FALSE)
score_heterogen$score

scoreTest_homogen <- vc_test_asym(y, x, phi=tim, indiv=rep(1:nindiv, each=nt),
    w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
    Sigma_xi=cov(tim), homogen_traj = TRUE)
scoreTest_homogen$set_pval

scoreTest_heterogen <- vc_test_asym(y, x, phi=tim, indiv=rep(1:nindiv,
```
vc_score_perm

Each

computation

involves

the

PCA

of

the

residuals

matrix

w.

Sigma_xi = cov(tim), homogen_traj = FALSE)

scoreTest_heterogen$set_pval

vc_score_perm

Computes variance component score test statistics and its permuted
distribution

Description

This function computes the variance component score test statistics along with its permuted values
for estimating its distribution under the null hypothesis.

Usage

vc_score_perm(
  y,
  x,
  indiv,
  phi,
  w,
  Sigma_xi = diag(ncol(phi)),
  na_rm = FALSE,
  n_perm = 1000,
  progressbar = TRUE,
  parallel_comp = TRUE,
  nb_cores = parallel::detectCores(logical = FALSE) - 1
)

Arguments

y

- a numeric matrix of dim g x n containing the raw RNA-seq counts for g genes
  from n samples

x

- a numeric design matrix of dim n x p containing the p covariates to be adjusted
  for

indiv

- a vector of length n containing the information for attributing each sample to
  one of the studied individuals. Coerced to be a factor

phi

- a numeric design matrix of size n x K containing the K variables to be tested.

w

- a vector of length n containing the weights for the n samples.

Sigma_xi

- a matrix of size K x K containing the covariance matrix of the K random effects
  on phi

na_rm

- logical: should missing values (including NA and NaN) be omitted from the cal-
  culations? Default is FALSE.

n_perm

- the number of permutation to perform. Default is 1000.
progressbar logical indicating whether a progressBar should be displayed when computing permutations (only in interactive mode).

parallel_comp a logical flag indicating whether parallel computation should be enabled. Only Linux and MacOS are supported, this is ignored on Windows. Default is TRUE.

nb_cores an integer indicating the number of cores to be used when parallel_comp is TRUE. Default is parallel::detectCores(logical=FALSE) - 1.

Value

A list with the following elements:

- score: an approximation of the observed set score
- scores_perm: a vector containing the permuted set scores
- gene_scores_unscaled: approximation of the individual gene scores
- gene_scores_unscaled_perm: a list of approximations of the permuted individual gene scores

Examples

set.seed(123)

##generate some fake data
n <- 100
r <- 12
t <- matrix(rep(1:3), r/3, ncol=1, nrow=r)
sigma <- 0.4
b0 <- 1
b1 <- 0

#under the null:
b1 <- 0
#under the alternative:
b1 <- 0.7

y.tilde <- b0 + b1*t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
      matrix(rep(y.tilde, n), ncol=n, nrow=r))
x <- matrix(1, ncol=1, nrow=r)

#run test
scoreTest <- vc_score_perm(y, x, phi=t, w=matrix(1, ncol=ncol(y),
    nrow=nrow(y)),
    Sigma_xi=matrix(1), indiv=rep(1:(r/3), each=3),
    parallel_comp = FALSE)

scoreTest$score
vc_test_asym

Asymptotic variance component test statistic and p-value

Description
This function computes an approximation of the variance component test based on the asymptotic
distribution of a mixture of $\chi^2$'s using the saddlepoint method from pchisqsum, as per Chen &
Lumley 2019 CSDA.

Usage
vc_test_asym(
  y,
  x,
  indiv = rep(1, nrow(x)),
  phi,
  w,
  Sigma_xi = diag(ncol(phi)),
  genewise_pvals = FALSE,
  homogen_traj = FALSE,
  na.rm = FALSE
)

Arguments
y  a numeric matrix of dim g x n containing the raw or normalized RNA-seq counts
   for g genes from n samples.
x  a numeric design matrix of dim n x p containing the p covariates to be adjusted
   for
indiv a vector of length n containing the information for attributing each sample to
       one of the studied individuals. Coerced to be a factor.
phi a numeric design matrix of size n x K containing the K longitudinal variables to
     be tested (typically a vector of time points or functions of time)
w  a vector of length n containing the weights for the n samples, corresponding to
   the inverse of the diagonal of the estimated covariance matrix of y.
Sigma_xi a matrix of size K x K containing the covariance matrix of the K random effects
       corresponding to phi.
genewise_pvals a logical flag indicating whether gene-wise p-values should be returned. Default is
                  FALSE in which case gene set p-value is computed and returned instead.
homogen_traj a logical flag indicating whether trajectories should be considered homogeneous.
              Default is FALSE in which case trajectories are not only tested for trend, but also
              for heterogeneity.
na.rm logical: should missing values (including NA and NaN) be omitted from the cal-
          culations? Default is FALSE.
Value

A list with the following elements when the set p-value is computed:

- `set_score_obs`: the approximation of the observed set score
- `set_pval`: the associated set p-value

or a list with the following elements when gene-wise p-values are computed:

- `gene_scores_obs`: vector of approximating the observed gene-wise scores
- `gene_pvals`: vector of associated gene-wise p-values

References

Chen T & Lumley T (2019), Numerical evaluation of methods approximating the distribution of a large quadratic form in normal variables, Computational Statistics & Data Analysis, 139:75-81.

See Also

`pchisqsum`

Examples

```r
set.seed(123)

# generate some fake data
n <- 100
r <- 12
t <- matrix(rep(1:(r/4)), 4, ncol=1, nrow=r)
sigma <- 0.4
b0 <- 1

# under the null:
b1 <- 0
# under the alternative:
# b1 <- 0.5
y.tilde <- b0 + b1*t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde)))), ncol=n, nrow=r) +
   matrix(rep(y.tilde, n), ncol=n, nrow=r))
x <- matrix(1, ncol=1, nrow=r)

# run test
asymTestRes <- vc_test_asym(y, x, phi=cbind(t, t^2),
                           w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
                           Sigma_xi=diag(2), indiv=1:r, genewise_pvals=TRUE)
plot(density(asymTestRes$gene_pvals))
quantile(asymTestRes$gene_pvals)
```
vc_test_perm

Permutation-based variance component test statistic and p-value

Description

This function computes an approximation of the Variance Component test for a mixture of \( \chi^2 \)s using permutations. This is preferable to the asymptotic approximation for small sample sizes. We rely on exact p-values following Phipson and Smyth, 2010 (see References).

Usage

```r
vc_test_perm(
  y,
  x,
  indiv = rep(1, nrow(x)),
  phi,
  w,
  Sigma_xi = diag(ncol(phi)),
  n_perm = 1000,
  progressbar = TRUE,
  parallel_comp = TRUE,
  nb_cores = parallel::detectCores(logical = FALSE) - 1,
  genewise_pvals = FALSE,
  adaptive = TRUE,
  max_adaptive = 64000,
  homogen_traj = FALSE,
  na.rm = FALSE
)
```

Arguments

- `y`: a numeric matrix of dim \( G \times n \) containing the raw RNA-seq counts for \( G \) genes from \( n \) samples.
- `x`: a numeric design matrix of dim \( n \times p \) containing the \( p \) covariates to be adjusted for.
- `indiv`: a vector of length \( n \) containing the information for attributing each sample to one of the studied individuals. Coerced to be a factor.
- `phi`: a numeric design matrix of size \( n \times K \) containing the \( K \) variables to be tested.
- `w`: a vector of length \( n \) containing the weights for the \( n \) samples.
- `Sigma_xi`: a matrix of size \( K \times K \) containing the covariance matrix of the \( K \) random effects.
- `n_perm`: the number of perturbations. Default is 1000.
- `progressbar`: logical indicating whether a progress bar should be displayed when computing permutations (only in interactive mode).
- `parallel_comp`: a logical flag indicating whether parallel computation should be enabled. Only Linux and MacOS are supported, this is ignored on Windows. Default is TRUE.
nb_cores an integer indicating the number of cores to be used when parallel_comp is TRUE. Default is parallel::detectCores(logical=FALSE) - 1.

genewise_pvals a logical flag indicating whether gene-wise p-values should be returned. Default is FALSE in which case gene-set p-value is computed and returned instead.

adaptive a logical flag indicating whether adaptive permutation should be performed. Default is TRUE

max_adaptive The maximum number of permutations considered. Default is 64000

homogen_traj a logical flag indicating whether trajectories should be considered homogeneous. Default is FALSE in which case trajectories are not only tested for trend, but also for heterogeneity.

na.rm logical: should missing values (including NA and NaN) be omitted from the calculations? Default is FALSE.

Value

A list with the following elements when the set p-value is computed:

- set_score_obs: the approximation of the observed set score
- set_pval: the associated set p-value

or a list with the following elements when gene-wise p-values are computed:

- gene_scores_obs: vector of approximating the observed gene-wise scores
- gene_pvals: vector of associated gene-wise p-values
- ds_fdr: vector of associated gene-wise discrete false discovery rates

References


Examples

set.seed(123)

```r
##generate some fake data
n <- 100
t <- matrix(rep(1:3), 4, ncol=1, nrow=r)
sigma <- 0.4
b0 <- 1

#under the null:
b1 <- 0
#under the alternative:
b1 <- 0.5
```
y.tilde <- b0 + b1*t + rnorm(r, sd = sigma)
y <- t(matrix(rnorm(n*r, sd = sqrt(sigma*abs(y.tilde))), ncol=n, nrow=r) +
    matrix(rep(y.tilde, n), ncol=n, nrow=r))
x <- matrix(1, ncol=1, nrow=r)

#run test
permTestRes <- vc_test_perm(y, x, phi=t,
    w=matrix(1, ncol=ncol(y), nrow=nrow(y)),
    indiv=rep(1:4, each=3), n_perm=50, #1000,
    parallel_comp = FALSE)
permTestRes$set_pval

---

**Description**

Implementation of the procedure described in Law *et al.* for estimating precision weights from RNA-seq data.

**Usage**

```r
voom_weights(y, x, preprocessed = FALSE, lowess_span = 0.5, R = NULL)
```

**Arguments**

- **y**: a matrix of size $G \times n$ containing the raw RNA-seq counts or preprocessed expressions from $n$ samples for $G$ genes.
- **x**: a matrix of size $n \times p$ containing the model covariates from $n$ samples (design matrix).
- **preprocessed**: a logical flag indicating whether the expression data have already been preprocessed (e.g. log2 transformed). Default is FALSE, in which case y is assumed to contain raw counts and is normalized into log(counts) per million.
- **lowess_span**: smoother span for the lowess function, between 0 and 1. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. Default is 0.5.
- **R**: library.size (optional, important to provide if preprocessed = TRUE). Default is NULL.

**Value**

a vector of length $n$ containing the computed precision weights

**Author(s)**

Boris Hejblum
References


See Also

lowess approxfun voom

Examples

set.seed(123)

G <- 10000
n <- 12
p <- 2

y <- sapply(1:n, FUN=function(x){rnbinom(n=G, size=0.07, mu=200)})
x <- sapply(1:p, FUN=function(x){rnorm(n=n, mean=n, sd=1)})

my_w <- voom_weights(y, x)
plot_weights(my_w)
if (requireNamespace('limma', quietly = TRUE)) {
  w_voom <- limma::voom(counts=y, design=x, plot=TRUE)
  # slightly faster, same results
  all.equal(my_w$weights, w_voom$weights)
}

if(interactive()){
  # microbenchmark::microbenchmark(limma::voom(counts=t(y), design=x,
  # plot=FALSE), voom_weights(x, y),
  # times=30)
}
Index

* datasets
  - baduel_5gs, 4
  - PBT_gmt, 15

* internal
  - vc_score, 25
  - vc_score_h, 26
  - vc_score_h_perm, 28
  - vc_score_perm, 31
  - gchar, 4
  - approxfun, 38
  - baduel (baduel_5gs), 4
  - baduel_5gs, 4
  - baduel_gmt (baduel_5gs), 4
  - bandwidth, 8, 13, 22, 23
  - BiocSet, 12
  - dear_seq, 3, 6
  - dearseq (dearseq-package), 3
  - dearseq-package, 3
  - density, 23
  - DESeqDataSet, 7, 12
  - design (baduel_5gs), 4
  - DGEList, 7, 12
  - dgsa_seq, 3, 11
  - expr_norm_corr (baduel_5gs), 4
  - ExpressionSet, 7, 12

  - ggplot, 18, 21
  - gmt, 12
  - GSA.read_gmt, 5, 16, 24

  - lowess, 38

  - p.adjust, 9, 14
  - PBT (PBT_gmt), 15
  - PBT_gmt, 15
  - pchisqsum, 33, 34

  - permPvals, 16
  - plot.dearseq, 17
  - plot_hist_pvals, 18
  - plot_ord_pvals, 19
  - plot_weights, 20
  - print.summary.dearseq
    - (summary.dearseq), 24
  - sp_weights, 8, 9, 13, 14, 20, 21
  - spaghettiPlot1GS, 23
  - SummarizedExperiment, 7, 12
  - summary.dearseq, 24

  - vc_score, 25
  - vc_score_h, 26
  - vc_score_h_perm, 28
  - vc_score_perm, 31
  - vc_test_asym, 9, 14, 33
  - vc_test_perm, 9, 14, 35
  - voom, 38

  voom_weights, 8, 13, 20, 37