Package ‘variancePartition’

October 14, 2021

Type Package

Title Quantify and interpret divers of variation in multilevel gene expression experiments

Version 1.22.0

Date 2020-10-26

Author Gabriel E. Hoffman

Maintainer Gabriel E. Hoffman <gabriel.hoffman@mssm.edu>

Description Quantify and interpret multiple sources of biological and technical variation in gene expression experiments. Uses a linear mixed model to quantify variation in gene expression attributable to individual, tissue, time point, or technical variables. Includes dream differential expression analysis for repeated measures.

VignetteBuilder knitr

License GPL (>= 2)

BugReports https://github.com/GabrielHoffman/variancePartition/issues

Suggests BiocStyle, knitr, pander, rmarkdown, edgeR, dendextend, tximport, tximportData, ballgown, DESeq2, RUnit, BiocGenerics, r2glmm, readr

biocViews RNASeq, GeneExpression, GeneSetEnrichment, DifferentialExpression, BatchEffect, QualityControl, Regression, Epigenetics, FunctionalGenomics, Transcriptomics, Normalization, Preprocessing, Microarray, ImmunoOncology, Software

Depends R (>= 3.6.0), ggplot2, limma, BiocParallel, scales, Biobase, methods

Imports MASS, pbkrtest (>= 0.4-4), lmerTest, iterators, splines, foreach, doParallel, colorRamps, gplots, progress, reshape2, lme4 (>= 1.1-10), grDevices, graphics, utils, stats

RoxygenNote 7.1.1

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

git_branch RELEASE_3_13

git_last_commit 25d1f1e

git_last_commit_date 2021-05-19

Date/Publication 2021-10-14

R topics documented:

`as.data.frame,varPartResults-method` *Convert to data.frame*

Description

Convert varPartResults to data.frame

Usage

```
## S4 method for signature 'varPartResults'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	varPartResults
`row.names`	pass thru to generic
`optional`	pass thru to generic
`...`	other arguments.

Value

data.frame

Examples

```
# load library
# library(variancePartition)

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
varPart <- fitExtractVarPartModel( geneExpr[1:5,], form, info )

# convert to matrix
as.data.frame(varPart)
```

as.matrix,varPartResults-method

Convert to matrix

Description

Convert varPartResults to matrix

Usage

```
## S4 method for signature 'varPartResults'
as.matrix(x, ...)
```

Arguments

x	varPartResults
...	other arguments.

Value

matrix

Examples

```
# load library
# library(variancePartition)

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
varPart <- fitExtractVarPartModel( geneExpr[1:5,], form, info )

# convert to matrix
as.matrix(varPart)
```

`calcVarPart` *Compute variance statistics*

Description

For linear model, variance fractions are computed based on the sum of squares explained by each component. For the linear mixed mode, the variance fractions are computed by variance component estimates for random effects and sum of squares for fixed effects.

For a generalized linear model, the variance fraction also includes the contribution of the link function so that fractions are reported on the linear (i.e. link) scale rather than the observed (i.e. response) scale. For linear regression with an identity link, fractions are the same on both scales. But for logit or probit links, the fractions are not well defined on the observed scale due to the transformation imposed by the link function.

The variance implied by the link function is the variance of the corresponding distribution: logit -> logistic distribution -> variance is pi^2/3 probit -> standard normal distribution -> variance is 1

Reviewed by Nakagawa and Schielzeth. 2012. A general and simple method for obtaining R2 from generalized linear mixed-effects models. https://doi.org/10.1111/j.2041-210x.2012.00261.x

Proposed by McKelvey and Zavoina. A statistical model for the analysis of ordinal level dependent variables. The Journal of Mathematical Sociology 4(1) 103-120 https://doi.org/10.1080/0022250X.1975.9989847

Also see DeMaris. Explained Variance in Logistic Regression: A Monte Carlo Study of Proposed Measures. Sociological Methods & Research 2002 https://doi.org/10.1177/0049124102031001002

We note that Nagelkerke's pseudo R^2 evaluates the variance explained by the full model. Instead, a variance partitioning approach evaluates the variance explained by each term in the model, so that the sum of each systematic plus random term sums to 1 (Hoffman and Schadt, 2016, Nakagawa and Schielzeth, 2012).

Usage

```
calcVarPart(fit, showWarnings = TRUE, ...)

## S4 method for signature 'lm'
calcVarPart(fit, showWarnings = TRUE, ...)

## S4 method for signature 'lmerMod'
calcVarPart(fit, showWarnings = TRUE, ...)

## S4 method for signature 'glm'
calcVarPart(fit, showWarnings = TRUE, ...)
```

Arguments

`fit`	model fit from lm() or lmer()
`showWarnings`	show warnings about model fit (default TRUE)
`...`	additional arguments (not currently used)

Details

Compute fraction of variation attributable to each variable in regression model. Also interpretable as the intra-class correlation after correcting for all other variables in the model.

Value

fraction of variance explained / ICC for each variable in the linear model

Examples

```
library(lme4)
data(varPartData)

# Linear mixed model
fit <- lmer( geneExpr[1,] ~ (1|Tissue) + Age, info)
calcVarPart( fit )

# Linear model
# Note that the two models produce slightly different results
# This is expected: they are different statistical estimates
# of the same underlying value
fit <- lm( geneExpr[1,] ~ Tissue + Age, info)
calcVarPart( fit )
```

`canCorPairs` *canCorPairs*

Description

Assess correlation between all pairs of variables in a formula

Usage

```
canCorPairs(formula, data, showWarnings = TRUE)
```

Arguments

formula	standard additive linear model formula (doesn't support random effects currently, so just change the syntax)
data	data.frame with the data for the variables in the formula
showWarnings	default to true

Details

Canonical Correlation Analysis (CCA) is similar to correlation between two vectors, except that CCA can accommodate matricies as well. For a pair of variables, canCorPairs assesses the degree to which they co-vary and contain the same information. Variables in the formula can be a continuous variable or a discrete variable expanded to a matrix (which is done in the backend of a regression model). For a pair of variables, canCorPairs uses CCA to compute the correlation between these variables and returns the pairwise correlation matrix.

Statistically, let rho be the array of correlation values returned by the standard R function cancor to compute CCA. canCorPairs returns sqrt(mean(rho^2)), which is the fraction of the maximum possible correlation. When comparing a two vectors, or a vector and a matrix, this gives the save value as the absolute correlation. When comparing two sets of categorical variables (i.e. expanded to two matricies), this is equivalent to Cramer's V statistic.

Note that CCA returns correlation values between 0 and 1.

Value

Matrix of correlation values between all pairs of variables.

Examples

```
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# specify formula
form <- ~ Individual + Tissue + Batch + Age + Height

# Compute Canonical Correlation Analysis (CCA)
# between all pairs of variables
# returns absolute correlation value
C = canCorPairs( form, info)

# Plot correlation matrix
plotCorrMatrix( C )
```

`classifyTestsF` *Multiple Testing Genewise Across Contrasts*

Description

For each gene, classify a series of related t-statistics as up, down or not significant.

Usage

```
classifyTestsF(object, ...)
```

Arguments

`object`	numeric matrix of t-statistics or an 'MArrayLM2' object from which the t-statistics may be extracted.
`...`	additional arguments

Details

Works like limma::classifyTestsF, except object can have a list of covariance matrices object$cov.coefficients.list, instead of just one in object$cov.coefficients

See Also

limma::classifyTestsF

`classifyTestsF,MArrayLM2-method` *Multiple Testing Genewise Across Contrasts*

Description

For each gene, classify a series of related t-statistics as up, down or not significant.

Usage

```
## S4 method for signature 'MArrayLM2'
classifyTestsF(
  object,
  cor.matrix = NULL,
  df = Inf,
  p.value = 0.01,
  fstat.only = FALSE
)
```

Arguments

`object`	numeric matrix of t-statistics or an 'MArrayLM2' object from which the t-statistics may be extracted.
`cor.matrix`	covariance matrix of each row of t-statistics. Defaults to the identity matrix.
`df`	numeric vector giving the degrees of freedom for the t-statistics. May have length 1 or length equal to the number of rows of tstat.
`p.value`	numeric value between 0 and 1 giving the desired size of the test
`fstat.only`	logical, if 'TRUE' then return the overall F-statistic as for 'FStat' instead of classifying the test results

Details

Works like limma::classifyTestsF, except object can have a list of covariance matrices object$cov.coefficients.list, instead of just one in object$cov.coefficients

See Also

limma::classifyTestsF

---

colinearityScore *Collinearity score*

Description

Collinearity score for a regression model indicating if variables are too highly correlated to give meaningful results

Usage

```
colinearityScore(fit)
```

Arguments

`fit`	regression model fit from lm() or lmer()

Value

Returns the collinearity score between 0 and 1, where a score > 0.999 means the degree of collinearity is too high. This function reports the correlation matrix between coefficient estimates for fixed effects. The collinearity score is the maximum absolute correlation value of this matrix. Note that the values are the correlation between the parameter estimates, and not between the variables themselves.

Examples

```
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)
form <- ~ Age + (1|Individual) + (1|Tissue)

res <- fitVarPartModel( geneExpr[1:10,], form, info )

# evaluate the collinearity score on the first model fit
# this reports the correlation matrix between coefficients estimates
# for fixed effects
# the collinearity score is the maximum absolute correlation value
# If the collinearity score > .999 then the variance partition
# estimates may be problematic
# In that case, a least one variable should be omitted
colinearityScore(res[[1]])
```

dream *Differential expression with linear mixed model*

Description

Fit linear mixed model for differential expression and preform hypothesis test on fixed effects as specified in the contrast matrix L

Usage

```
dream(
  exprObj,
  formula,
  data,
  L,
  ddf = c("Satterthwaite", "Kenward-Roger"),
  useWeights = TRUE,
  weightsMatrix = NULL,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  suppressWarnings = FALSE,
  quiet = FALSE,
  BPPARAM = bpparam(),
  computeResiduals = TRUE,
  REML = TRUE,
  ...
)
```

Arguments

`exprObj`	matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)` Formulas with only fixed effects also work, and lmFit() followed by contrasts.fit() are run.
`data`	data.frame with columns corresponding to formula
`L`	contrast matrix specifying a linear combination of fixed effects to test
`ddf`	Specifiy "Satterthwaite" or "Kenward-Roger" method to estimate effective degress of freedom for hypothesis testing in the linear mixed model. Note that Kenward-Roger is more accurate, but is *much* slower. Satterthwaite is a good enough approximation for most datasets.

`useWeights`	if TRUE, analysis uses heteroskedastic error estimates from `voom()`. Value is ignored unless exprObj is an `EList()` from `voom()` or `weightsMatrix` is specified
`weightsMatrix`	matrix the same dimension as exprObj with observation-level weights from `voom()`. Used only if useWeights is TRUE
`control`	control settings for `lmer()`
`suppressWarnings`	if TRUE, do not stop because of warnings or errors in model fit
`quiet`	suppress message, default FALSE
`BPPARAM`	parameters for parallel evaluation
`computeResiduals`	if TRUE, compute residuals and extract with `residuals(fit)`. Setting to FALSE saves memory
`REML`	use restricted maximum likelihood to fit linear mixed model. default is TRUE. See Details.
`...`	Additional arguments for `lmer()` or `lm()`

Details

A linear (mixed) model is fit for each gene in exprObj, using formula to specify variables in the regression. If categorical variables are modeled as random effects (as is recommended), then a linear mixed model us used. For example if formula is `~ a + b + (1|c)`, then the model is

`fit <-lmer( exprObj[j,] ~ a + b + (1|c),data=data)`

`useWeights=TRUE` causes `weightsMatrix[j,]` to be included as weights in the regression model.

Note: Fitting the model for 20,000 genes can be computationally intensive. To accelerate computation, models can be fit in parallel using `BiocParallel` to run code in parallel. Parallel processing must be enabled before calling this function. See below.

The regression model is fit for each gene separately. Samples with missing values in either gene expression or metadata are omitted by the underlying call to lmer.

Hypothesis tests and degrees of freedom are producted by `lmerTest` and `pbkrtest` pacakges

While `REML=TRUE` is required by `lmerTest` when ddf='Kenward-Roger', ddf='Satterthwaite' can be used with `REML` as `TRUE` or `FALSE`. Since the Kenward-Roger method gave the best power with an accurate control of false positive rate in our simulations, and since the Satterthwaite method with REML=TRUE gives p-values that are slightly closer to the Kenward-Roger p-values, `REML=TRUE` is the default. See Vignette "3) Theory and practice of random effects and REML"

Value

MArrayLM2 object (just like MArrayLM from limma), and the directly estimated p-value (without eBayes)

Examples

```
# load library
# library(variancePartition)
library(BiocParallel)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

form <- ~ Batch + (1|Individual) + (1|Tissue)

# Fit linear mixed model for each gene
# run on just 10 genes for time
fit = dream( geneExpr[1:10,], form, info)

# view top genes
topTable( fit )

# get contrast matrix testing if the coefficient for Batch2 is
# different from coefficient for Batch3
# The variable of interest must be a fixed effect
L = getContrast( geneExpr, form, info, c("Batch2", "Batch3"))

# plot contrasts
plotContrasts( L )

# Fit linear mixed model for each gene
# run on just 10 genes for time
# Note that that dream() is not compatible with eBayes()
fit2 = dream( geneExpr[1:10,], form, info, L)

# view top genes
topTable( fit2 )

# Parallel processing using multiple cores with reduced memory usage
param = SnowParam(4, "SOCK", progressbar=TRUE)
fit3 = dream( geneExpr[1:10,], form, info, L, BPPARAM = param)

# Fit fixed effect model for each gene
# Use lmFit in the backend
# Need to run eBayes afterward
form <- ~ Batch
fit4 = dream( geneExpr[1:10,], form, info)
fit4 = eBayes( fit4 )

# view top genes
topTable( fit4 )
```

```
# Compute residuals using dream
fit5 = dream( geneExpr[1:10,], form, info, L, BPPARAM = param, computeResiduals=TRUE)

# Return residuals
residuals(fit5)
```

---

eBayes,MArrayLM2-method *eBayes for MArrayLM2*

Description

eBayes for MArrayLM2. It simply returns the orginal input with no change

Usage

```
## S4 method for signature 'MArrayLM2'
eBayes(
  fit,
  proportion = 0.01,
  stdev.coef.lim = c(0.1, 4),
  trend = FALSE,
  robust = FALSE,
  winsor.tail.p = c(0.05, 0.1)
)
```

Arguments

`fit`	fit
`proportion`	proportion
`stdev.coef.lim`	stdev.coef.lim
`trend`	trend
`robust`	robust
`winsor.tail.p`	winsor.tail.p

Details

Note that empirical Bayes is problematic for linear mixed models. This function returns its original input with no change. It is included here just for compatibility.

Value

results of eBayes

ESS *Effective sample size*

Description

Compute effective sample size based on correlation structure in linear mixed model

Usage

```
ESS(fit, method = "full")

## S4 method for signature 'lmerMod'
ESS(fit, method = "full")
```

Arguments

`fit`	model fit from lmer()
`method`	"full" uses the full correlation structure of the model. The "approximate" method makes the simplifying assumption that the study has a mean of m samples in each of k groups, and computes m based on the study design. When the study design is evenly balanced (i.e. the assumption is met), this gives the same results as the "full" method.

Details

Effective sample size calculations are based on:

Liu, G., and Liang, K. Y. (1997). Sample size calculations for studies with correlated observations. Biometrics, 53(3), 937-47.

"full" method: if

$$V_x = var(Y; x)$$

is the variance-covariance matrix of Y, the response, based on the covariate x, then the effective sample size corresponding to this covariate is

$$\Sigma_{i,j}(V_x^{-1})_{i,j}$$

. In R notation, this is: `sum(solve(V_x))`. In practice, this can be evaluted as sum(w), where R

"approximate" method: Letting m be the mean number of samples per group,

$$k$$

be the number of groups, and

$$\rho$$

be the intraclass correlation, the effective sample size is

$$mk/(1 + \rho(m - 1))$$

Note that these values are equal when there are exactly m samples in each group. If m is only an average then this an approximation.

Value

effective sample size for each random effect in the model

Examples

```
library(lme4)
data(varPartData)

# Linear mixed model
fit <- lmer( geneExpr[1,] ~ (1|Individual) + (1|Tissue) + Age, info)

# Effective sample size
ESS( fit )
```

`extractVarPart` *Extract variance statistics*

Description

Extract variance statistics from list of models fit with `lm()` or `lmer()`

Usage

```
extractVarPart(modelList, showWarnings = TRUE, ...)
```

Arguments

`modelList`	list of `lmer()` model fits
`showWarnings`	show warnings about model fit (default TRUE)
`...`	other arguments

Value

`data.frame` of fraction of variance explained by each variable, after correcting for all others.

Examples

```
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)
```

```
# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expresson
# If categoritical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
plotVarPart( sortCols( varPart ) )

# Advanced:
# Fit model and extract variance in two separate steps
# Step 1: fit model for each gene, store model fit for each gene in a list
results <- fitVarPartModel( geneExpr, form, info )

# Step 2: extract variance fractions
varPart <- extractVarPart( results )
```

---

fitExtractVarPartModel

Fit linear (mixed) model, report variance fractions

Description

Fit linear (mixed) model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables. Report fraction of variance attributable to each variable

Usage

```
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'matrix'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'data.frame'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'EList'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
```

```
)

## S4 method for signature 'ExpressionSet'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'sparseMatrix'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)
```

Arguments

`exprObj`	matrix of expression data (g genes x n samples), or `ExpressionSet`, or `EList` returned by `voom()` from the `limma` package
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)`
`data`	`data.frame` with columns corresponding to formula
`REML`	use restricted maximum likelihood to fit linear mixed model. default is FALSE. See Details.
`useWeights`	if TRUE, analysis uses heteroskedastic error estimates from `voom()`. Value is ignored unless exprObj is an `EList()` from `voom()` or `weightsMatrix` is specified
`weightsMatrix`	matrix the same dimension as exprObj with observation-level weights from `voom()`. Used only if `useWeights` is TRUE

`showWarnings`	show warnings about model fit (default TRUE)
`control`	control settings for `lmer()`
`quiet`	suppress message, default FALSE
`BPPARAM`	parameters for parallel evaluation
`...`	Additional arguments for `lmer()` or `lm()`

Details

A linear (mixed) model is fit for each gene in exprObj, using formula to specify variables in the regression. If categorical variables are modeled as random effects (as is recommended), then a linear mixed model us used. For example if formula is ~ `a + b + (1|c)`, then the model is

fit <- lmer(exprObj[j,] ~ a + b + (1|c), data=data)

If there are no random effects, so formula is ~ a + b + c, a 'standard' linear model is used:

`fit <-lm( exprObj[j,] ~ a + b + c,data=data)`

In both cases, `useWeights=TRUE` causes `weightsMatrix[j,]` to be included as weights in the regression model.

Note: Fitting the model for 20,000 genes can be computationally intensive. To accelerate computation, models can be fit in parallel using `BiocParallel` to run in parallel. Parallel processing must be enabled before calling this function. See below.

The regression model is fit for each gene separately. Samples with missing values in either gene expression or metadata are omitted by the underlying call to `lm/lmer`.

`REML=FALSE` uses maximum likelihood to estimate variance fractions. This approach produced unbiased estimates, while `REML=TRUE` can show substantial bias. See Vignette "3) Theory and practice of random effects and REML"

Value

list() of where each entry is a model fit produced by `lmer()` or `lm()`

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)
```

```
# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
plotVarPart( sortCols( varPart ) )

# Note: fitExtractVarPartModel also accepts ExpressionSet
data(sample.ExpressionSet, package="Biobase")

# ExpressionSet example
form <- ~ (1|sex) + (1|type) + score
info2 <- pData(sample.ExpressionSet)
varPart2 <- fitExtractVarPartModel( sample.ExpressionSet, form, info2 )
```

`fitVarPartModel` *Fit linear (mixed) model*

Description

Fit linear (mixed) model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables.

Usage

```
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)
```

```
## S4 method for signature 'matrix'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'data.frame'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'EList'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)
```

```
## S4 method for signature 'ExpressionSet'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'sparseMatrix'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)
```

Arguments

`exprObj`	matrix of expression data (g genes x n samples), or `ExpressionSet`, or `EList` returned by `voom()` from the `limma` package
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)`
`data`	`data.frame` with columns corresponding to formula
`REML`	use restricted maximum likelihood to fit linear mixed model. default is FALSE. See Details.
`useWeights`	if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an `EList()` from `voom()` or `weightsMatrix` is specified
`weightsMatrix`	matrix the same dimension as exprObj with observation-level weights from `voom()`. Used only if useWeights is TRUE

`showWarnings`	show warnings about model fit (default TRUE)
`fxn`	apply function to model fit for each gene. Defaults to identify function so it returns the model fit itself
`control`	control settings for `lmer()`
`quiet`	suppress message, default FALSE
`BPPARAM`	parameters for parallel evaluation
`...`	Additional arguments for `lmer()` or `lm()`

Details

A linear (mixed) model is fit for each gene in exprObj, using formula to specify variables in the regression. If categorical variables are modeled as random effects (as is recommended), then a linear mixed model us used. For example if formula is `~ a + b + (1|c)`, then the model is

`fit <-lmer( exprObj[j,] ~ a + b + (1|c),data=data)`

If there are no random effects, so formula is `~ a + b + c`, a 'standard' linear model is used:

`fit <-lm( exprObj[j,] ~ a + b + c,data=data)`

In both cases, `useWeights=TRUE` causes `weightsMatrix[j,]` to be included as weights in the regression model.

Note: Fitting the model for 20,000 genes can be computationally intensive. To accelerate computation, models can be fit in parallel using `BiocParallel` to run in parallel. Parallel processing must be enabled before calling this function. See below.

The regression model is fit for each gene separately. Samples with missing values in either gene expression or metadata are omitted by the underlying call to lm/lmer.

Since this function returns a list of each model fit, using this function is slower and uses more memory than `fitExtractVarPartModel()`.

`REML=FALSE` uses maximum likelihood to estimate variance fractions. This approach produced unbiased estimates, while `REML=TRUE` can show substantial bias. See Vignette "3) Theory and practice of random effects and REML"

Value

`list()` of where each entry is a model fit produced by `lmer()` or `lm()`

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)
```

```
# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
# also sort columns
plotVarPart( sortCols( varPart ) )

# Advanced:
# Fit model and extract variance in two separate steps
# Step 1: fit model for each gene, store model fit for each gene in a list
results <- fitVarPartModel( geneExpr, form, info )

# Step 2: extract variance fractions
varPart <- extractVarPart( results )

# Note: fitVarPartModel also accepts ExpressionSet
data(sample.ExpressionSet, package="Biobase")

# ExpressionSet example
form <- ~ (1|sex) + (1|type) + score
info2 <- pData(sample.ExpressionSet)
results2 <- fitVarPartModel( sample.ExpressionSet, form, info2 )
```

`getContrast` *Extract contrast matrix for linear mixed model*

Description

Extract contrast matrix, L, testing a single variable. Contrasts involving more than one variable can be constructed by modifying L directly

Usage

```
getContrast(exprObj, formula, data, coefficient)
```

Arguments

`exprObj`	matrix of expression data (g genes x n samples), or `ExpressionSet`, or `EList` returned by `voom()` from the `limma` package
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)` Formulas with only fixed effects also work
`data`	data.frame with columns corresponding to formula
`coefficient`	the coefficient to use in the hypothesis test

Value

Contrast matrix testing one variable

Examples

```
# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# get contrast matrix testing if the coefficient for Batch2 is zero
# The variable of interest must be a fixed effect
form <- ~ Batch + (1|Individual) + (1|Tissue)
L = getContrast( geneExpr, form, info, "Batch3")

# get contrast matrix testing if Batch3 - Batch2 = 0
form <- ~ Batch + (1|Individual) + (1|Tissue)
L = getContrast( geneExpr, form, info, c("Batch3", "Batch2"))

# To test against Batch1 use the formula:
#  ~ 0 + Batch + (1|Individual) + (1|Tissue)
# to estimate Batch1 directly instead of using it as the baseline
```

`get_prediction` *Compute predicted value of formula for linear (mixed) model*

Description

Compute predicted value of formula for linear (mixed) model for with `lm` or `lmer`

Usage

```
get_prediction(fit, formula)

## S4 method for signature 'lmerMod'
get_prediction(fit, formula)
```

```
## S4 method for signature 'lm'
get_prediction(fit, formula)
```

Arguments

`fit`	model fit with `lm` or `lmer`
`formula`	formula of fixed and random effects to predict

Details

Similar motivation as `lme4:::predict.merMod()`, but that function cannot use just a subset of the fixed effects: it either uses none or all. Note that the intercept is included in the formula by default. To exclude it from the prediction use `~ 0 + ...` syntax

Value

Predicted values from formula using parameter estimates from fit linear (mixed) model

Examples

```
library(lme4)

# Linear model
fit <- lm(Reaction ~ Days, sleepstudy)

# prediction of intercept
get_prediction( fit, ~ 1)

# prediction of Days without intercept
get_prediction( fit, ~ 0 + Days)

# Linear mixed model

# fit model
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)

# predict Days, but exclude intercept
get_prediction( fm1, ~ 0 + Days)

# predict Days and (Days | Subject) random effect, but exclude intercept
get_prediction( fm1, ~ 0 + Days +  (Days | Subject))
```

`ggColorHue` *Default colors for ggplot*

Description

Return an array of n colors the same as the default used by ggplot2

Usage

```
ggColorHue(n)
```

Arguments

`n`	number of colors

Value

array of colors of length n

Examples

```
ggColorHue(4)
```

`MArrayLM2-class` *Class MArrayLM2*

Description

Class `MArrayLM2`

`plotCompareP` *Compare p-values from two analyses*

Description

Plot -log10 p-values from two analyses and color based on donor component from variancePartition analysis

Usage

```
plotCompareP(
  p1,
  p2,
  vpDonor,
  dupcorvalue,
  fraction = 0.2,
  xlabel = bquote(duplicateCorrelation ~ (-log[10] ~ p)),
  ylabel = bquote(dream ~ (-log[10] ~ p))
)
```

Arguments

`p1`	p-value from first analysis
`p2`	p-value from second analysis
`vpDonor`	donor component for each gene from variancePartition analysis
`dupcorvalue`	scalar donor component from duplicateCorrelation
`fraction`	fraction of highest/lowest values to use for best fit lines
`xlabel`	for x-axis
`ylabel`	label for y-axis

Value

ggplot2 plot

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Perform very simple analysis for demonstration

# Analysis 1
form <- ~ Batch
fit = dream( geneExpr, form, info)
fit = eBayes( fit )
res = topTable( fit, number=Inf, coef="Batch3" )

# Analysis 2
form <- ~ Batch + (1|Tissue)
```

```
fit2 = dream( geneExpr, form, info)
res2 = topTable( fit2, number=Inf, coef="Batch3" )

# Compare p-values
plotCompareP( res$P.Value, res2$P.Value, runif(nrow(res)), .3 )
```

`plotContrasts` *Plot representation of contrast matrix*

Description

Plot contrast matrix to clarify interpretation of hypothesis tests with linear contrasts

Usage

```
plotContrasts(L)
```

Arguments

`L`	contrast matrix

Value

ggplot2 object

Examples

```
# load library
# library(variancePartition)

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# get contrast matrix testing if the coefficient for Batch2 is zero
form <- ~ Batch + (1|Individual) + (1|Tissue)
L1 = getContrast( geneExpr, form, info, "Batch3")

# get contrast matrix testing if the coefficient for Batch2 is different from Batch3
form <- ~ Batch + (1|Individual) + (1|Tissue)
L2 = getContrast( geneExpr, form, info, c("Batch2", "Batch3"))

# combine contrasts into single matrix
L_combined = cbind(L1, L2)

# plot contrasts
plotContrasts( L_combined )
```

plotCorrMatrix *plotCorrMatrix*

Description

Plot correlation matrix

Usage

```
plotCorrMatrix(
  C,
  dendrogram = "both",
  sort = TRUE,
  margins = c(13, 13),
  key.xlab = "correlation",
  ...
)
```

Arguments

`C`	correlation matrix: R or R^2 matrix
`dendrogram`	character string indicating whether to draw 'both' or none'
`sort`	sort rows and columns based on clustering
`margins`	spacing of plot
`key.xlab`	label of color gradient
`...`	additional arguments to heatmap.2

Details

Plots image of correlation matrix using customized call to heatmap.2

Value

Image of correlation matrix

Examples

```
# simulate simple matrix of 10 variables
mat = matrix(rnorm(1000), ncol=10)

# compute correlation matrix
C = cor(mat)

# plot correlations
plotCorrMatrix( C )

# plot squared correlations
```

```
plotCorrMatrix( C^2, dendrogram="none" )
```

`plotCorrStructure` *plotCorrStructure*

Description

Plot correlation structure of a gene based on random effects

Usage

```
plotCorrStructure(
  fit,
  varNames = names(coef(fit)),
  reorder = TRUE,
  pal = colorRampPalette(c("white", "red", "darkred")),
  hclust.method = "complete"
)
```

Arguments

`fit`	linear mixed model fit of a gene produced by lmer() or fitVarPartModel()
`varNames`	variables in the metadata for which the correlation structure should be shown. Variables must be random effects
`reorder`	how to reorder the rows/columns of the correlation matrix. reorder=FALSE gives no reorder. reorder=TRUE reorders based on hclust. reorder can also be an array of indices to reorder the samples manually
`pal`	color palette
`hclust.method`	clustering methods for hclust

Value

Image of correlation structure between each pair of experiments for a single gene

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
data(varPartData)
```

```
# specify formula
form <- ~ Age + (1|Individual) + (1|Tissue)

# fit and return linear mixed models for each gene
fitList <- fitVarPartModel( geneExpr[1:10,], form, info )

# Focus on the first gene
fit = fitList[[1]]

# plot correlation sturcture based on Individual, reordering samples with hclust
plotCorrStructure( fit, "Individual" )

# don't reorder
plotCorrStructure( fit, "Individual", reorder=FALSE )

# plot correlation sturcture based on Tissue, reordering samples with hclust
plotCorrStructure( fit, "Tissue" )

# don't reorder
plotCorrStructure( fit, "Tissue", FALSE )

# plot correlation structure based on all random effects
# reorder manually by Tissue and Individual
idx = order(info$Tissue, info$Individual)
plotCorrStructure( fit, reorder=idx )

# plot correlation structure based on all random effects
# reorder manually by Individual, then Tissue
idx = order(info$Individual, info$Tissue)
plotCorrStructure( fit, reorder=idx )
```

plotPercentBars *Bar plot of variance fractions*

Description

Bar plot of variance fractions for a subset of genes

Usage

```
plotPercentBars(varPart, col = c(ggColorHue(ncol(varPart) - 1), "grey85"))
```

Arguments

varPart	object returned by extractVarPart() or fitExtractVarPartModel()
col	color of bars for each variable

Value

Returns ggplot2 barplot

Examples

```
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# Bar plot for a subset of genes showing variance fractions
plotPercentBars( varPart[1:5,] )

# Move the legend to the top
plotPercentBars( varPart[1:5,] ) + theme(legend.position="top")
```

---

plotStratify *plotStratify*

Description

Plot gene expression stratified by another variable

Usage

```
plotStratify(
  formula,
  data,
  xlab,
  ylab,
  main,
  sortBy,
  colorBy,
  sort = TRUE,
  text = NULL,
  text.y = 1,
  text.size = 5,
  pts.cex = 1,
  ylim = NULL,
  legend = TRUE,
  x.labels = FALSE
)
```

Arguments

`formula`	specify variables shown in the x- and y-axes. Y-axis should be continuous variable, x-axis should be discrete.
`data`	data.frame storing continuous and discrete variables specified in formula
`xlab`	label x-asis. Defaults to value of xval
`ylab`	label y-asis. Defaults to value of yval
`main`	main label
`sortBy`	name of column in geneExpr to sort samples by. Defaults to xval
`colorBy`	name of column in geneExpr to color box plots. Defaults to xval
`sort`	if TRUE, sort boxplots by median value, else use default ordering
`text`	plot text on the top left of the plot
`text.y`	indicate position of the text on the y-axis as a fraction of the y-axis range
`text.size`	size of text
`pts.cex`	size of points
`ylim`	specify range of y-axis
`legend`	show legend
`x.labels`	show x axis labels

Value

ggplot2 object

Examples

```
# Note: This is a newer, more convient interface to plotStratifyBy()

# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# Create data.frame with expression and Tissue information for each sample
GE = data.frame( Expression = geneExpr[1,], Tissue = info$Tissue)

# Plot expression stratified by Tissue
plotStratify( Expression ~ Tissue, GE )
```

```
# Omit legend and color boxes grey
plotStratify( Expression ~ Tissue, GE, colorBy = NULL)

# Specify colors
col = c( B="green", A="red", C="yellow")
plotStratify( Expression ~ Tissue, GE, colorBy=col, sort=FALSE)
```

`plotStratifyBy` *plotStratifyBy*

Description

Plot gene expression stratified by another variable

Usage

```
plotStratifyBy(
  geneExpr,
  xval,
  yval,
  xlab = xval,
  ylab = yval,
  main = NULL,
  sortBy = xval,
  colorBy = xval,
  sort = TRUE,
  text = NULL,
  text.y = 1,
  text.size = 5,
  pts.cex = 1,
  ylim = NULL,
  legend = TRUE,
  x.labels = FALSE
)
```

Arguments

`geneExpr`	data.frame of gene expression values and another variable for each sample. If there are multiple columns, the user can specify which one to use
`xval`	name of column in geneExpr to be used along x-axis to stratify gene expression
`yval`	name of column in geneExpr indicating gene expression
`xlab`	label x-asis. Defaults to value of xval
`ylab`	label y-asis. Defaults to value of yval
`main`	main label
`sortBy`	name of column in geneExpr to sort samples by. Defaults to xval

`colorBy`	name of column in geneExpr to color box plots. Defaults to xval
`sort`	if TRUE, sort boxplots by median value, else use default ordering
`text`	plot text on the top left of the plot
`text.y`	indicate position of the text on the y-axis as a fraction of the y-axis range
`text.size`	size of text
`pts.cex`	size of points
`ylim`	specify range of y-axis
`legend`	show legend
`x.labels`	show x axis labels

Value

ggplot2 object

Examples

```
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# Create data.frame with expression and Tissue information for each sample
GE = data.frame( Expression = geneExpr[1,], Tissue = info$Tissue)

# Plot expression stratified by Tissue
plotStratifyBy( GE, "Tissue", "Expression")

# Omit legend and color boxes grey
plotStratifyBy( GE, "Tissue", "Expression", colorBy = NULL)

# Specify colors
col = c( B="green", A="red", C="yellow")
plotStratifyBy( GE, "Tissue", "Expression", colorBy=col, sort=FALSE)
```

`plotVarPart` *Violin plot of variance fractions*

Description

Violin plot of variance fraction for each gene and each variable

Usage

```
plotVarPart(
  obj,
  col = c(ggColorHue(ncol(obj) - 1), "grey85"),
  label.angle = 20,
  main = "",
  ylab = "",
  convertToPercent = TRUE,
  ...
)

## S4 method for signature 'matrix'
plotVarPart(
  obj,
  col = c(ggColorHue(ncol(obj) - 1), "grey85"),
  label.angle = 20,
  main = "",
  ylab = "",
  convertToPercent = TRUE,
  ...
)

## S4 method for signature 'data.frame'
plotVarPart(
  obj,
  col = c(ggColorHue(ncol(obj) - 1), "grey85"),
  label.angle = 20,
  main = "",
  ylab = "",
  convertToPercent = TRUE,
  ...
)

## S4 method for signature 'varPartResults'
plotVarPart(
  obj,
  col = c(ggColorHue(ncol(obj) - 1), "grey85"),
  label.angle = 20,
  main = "",
  ylab = "",
  convertToPercent = TRUE,
  ...
)
```

Arguments

`obj`	`varParFrac` object returned by `fitExtractVarPart` or `extractVarPart`
`col`	vector of colors

`label.angle`	angle of labels on x-axis
`main`	title of plot
`ylab`	text on y-axis
`convertToPercent`	multiply fractions by 100 to convert to percent values
`...`	additional arguments

Value

Makes violin plots of variance components model. This function uses the graphics interface from ggplot2. Warnings produced by this function usually ggplot2 warning that the window is too small.

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
plotVarPart( sortCols( varPart ) )
```

`residuals,MArrayLM-method`

residuals for MArrayLM

Description

residuals for MArrayLM

Usage

```
## S4 method for signature 'MArrayLM'
residuals(object, ...)
```

Arguments

`object`	MArrayLM object from dream
`...`	other arguments, currently ignored

Value

results of residuals

`residuals,MArrayLM2-method`
residuals for MArrayLM2

Description

residuals for MArrayLM2

Usage

```
## S4 method for signature 'MArrayLM2'
residuals(object, ...)
```

Arguments

`object`	MArrayLM2 object from dream
`...`	other arguments, currently ignored

Value

results of residuals

`residuals,VarParFitList-method` *Residuals from model fit*

Description

Extract residuals for each gene from model fit with fitVarPartModel()

Usage

```
## S4 method for signature 'VarParFitList'
residuals(object, ...)
```

Arguments

`object`	object produced by fitVarPartModel()
`...`	other arguments.

Details

If model is fit with missing data, residuals returns NA for entries that were missing in the original data

Value

Residuals extracted from model fits stored in object

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
modelFit <- fitVarPartModel( geneExpr, form, info )
```

```
# Extract residuals of model fit
res <- residuals( modelFit )
```

---

sortCols *Sort variance partition statistics*

Description

Sort columns returned by `extractVarPart()` or `fitExtractVarPartModel()`

Usage

```
sortCols(
  x,
  FUN = median,
  decreasing = TRUE,
  last = c("Residuals", "Measurement.error"),
  ...
)

## S4 method for signature 'matrix'
sortCols(
  x,
  FUN = median,
  decreasing = TRUE,
  last = c("Residuals", "Measurement.error"),
  ...
)

## S4 method for signature 'data.frame'
sortCols(
  x,
  FUN = median,
  decreasing = TRUE,
  last = c("Residuals", "Measurement.error"),
  ...
)

## S4 method for signature 'varPartResults'
sortCols(
  x,
  FUN = median,
  decreasing = TRUE,
  last = c("Residuals", "Measurement.error"),
  ...
)
```

Arguments

`x`	object returned by `extractVarPart()` or `fitExtractVarPartModel()`
`FUN`	function giving summary statistic to sort by. Defaults to median
`decreasing`	logical. Should the sorting be increasing or decreasing?
`last`	columns to be placed on the right, regardless of values in these columns
`...`	other arguments to sort

Value

data.frame with columns sorted by mean value, with Residuals in last column

Examples

```
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
# sort columns by median value
plotVarPart( sortCols( varPart ) )
```

VarParCIList-class *Class VarParCIList*

Description

Class `VarParCIList`

VarParFitList-class *Class VarParFitList*

Description

Class `VarParFitList`

varParFrac-class *Class varParFrac*

Description

Class `varParFrac`

varPartConfInf *Linear mixed model confidence intervals*

Description

Fit linear mixed model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables. Then perform parametric bootstrap sampling to get a 95% confidence intervals for each variable for each gene.

Usage

```
varPartConfInf(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  colinearityCutoff = 0.999,
 control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"),
  nsim = 1000,
  ...
)
```

Arguments

`exprObj`	matrix of expression data (g genes x n samples), or `ExpressionSet`, or `EList` returned by `voom()` from the `limma` package
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)`
`data`	`data.frame` with columns corresponding to formula
`REML`	use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option, but here for compatibility.
`useWeights`	if TRUE, analysis uses heteroskedastic error estimates from `voom()`. Value is ignored unless exprObj is an `EList` from `voom()` or `weightsMatrix` is specified
`weightsMatrix`	matrix the same dimension as exprObj with observation-level weights from `voom()`. Used only if useWeights is TRUE
`showWarnings`	show warnings about model fit (default TRUE)
`colinearityCutoff`	cutoff used to determine if model is computationally singular
`control`	control settings for `lmer()`
`nsim`	number of bootstrap datasets
`...`	Additional arguments for `lmer()` or `lm()`

Details

A linear mixed model is fit for each gene, and `bootMer()` is used to generate parametric bootstrap confidence intervals. `use.u=TRUE` is used so that the

$$\hat{u}$$

values from the random effects are used as estimated and are not re-sampled. This gives confidence intervals as if additional data were generated from these same current samples. Conversely, `use.u=FALSE` assumes that this dataset is a sample from a larger population. Thus it simulates

$$\hat{u}$$

based on the estimated variance parameter. This approach gives confidence intervals as if additional data were collected from the larger population from which this dataset is sampled. Overall, `use.u=TRUE` gives smaller confidence intervals that are appropriate in this case.

Value

`list()` of where each entry is the result for a gene. Each entry is a matrix of the 95% confidence interval of the variance fraction for each variable

Examples

```
# load library
# library(variancePartition)

# Intialize parallel backend with 4 cores
library(BiocParallel)
register(SnowParam(4))

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Compute bootstrap confidence intervals for each variable for each gene
resCI <- varPartConfInf( geneExpr[1:5,], form, info, nsim=100 )
```

`varPartData` *Simulation dataset for examples*

Description

A simulated dataset of gene expression and metadata

A simulated dataset of gene counts

info about study design

Normalized expression data

Usage

```
data(varPartData)

data(varPartData)

data(varPartData)

data(varPartData)
```

Format

A dataset of 100 samples and 200 genes

A dataset of 100 samples and 200 genes

A dataset of 100 samples and 200 genes

A dataset of 100 samples and 200 genes

Details

- geneCounts gene expression in the form of RNA-seq counts
- geneExpr gene expression on a continuous scale
- info metadata about the study design

- geneCounts gene expression in the form of RNA-seq counts
- geneExpr gene expression on a continuous scale
- info metadata about the study design

- geneCounts gene expression in the form of RNA-seq counts
- geneExpr gene expression on a continuous scale
- info metadata about the study design

- geneCounts gene expression in the form of RNA-seq counts
- geneExpr gene expression on a continuous scale
- info metadata about the study design

`varPartDEdata` *Simulation dataset for dream example*

Description

Gene counts from RNA-seq

metadata matrix of sample information

Usage

```
data(varPartDEdata)

data(varPartDEdata)
```

Format

A dataset of 24 samples and 19,364 genes

A dataset of 24 samples and 19,364 genes

Details

- countMatrix gene expression in the form of RNA-seq counts
- metadata metadata about the study design

- countMatrix gene expression in the form of RNA-seq counts
- metadata metadata about the study design

`varPartResults-class` *Class varPartResults*

Description

Class `varPartResults`

`voomWithDreamWeights` *Transform RNA-Seq Data Ready for Linear Mixed Modelling with* `dream()`

Description

Transform count data to log2-counts per million (logCPM), estimate the mean-variance relationship and use this to compute appropriate observation-level weights. The data are then ready for linear mixed modelling with `dream()`. This method is the same as limma::voom(), except that it allows random effects in the formula

Usage

```
voomWithDreamWeights(
  counts,
  formula,
  data,
  lib.size = NULL,
  normalize.method = "none",
  span = 0.5,
  plot = FALSE,
  save.plot = FALSE,
  quiet = FALSE,
  BPPARAM = bpparam(),
  ...
)
```

Arguments

`counts`	a numeric `matrix` containing raw counts, or an `ExpressionSet` containing raw counts, or a `DGEList` object. Counts must be non-negative and NAs are not permitted.
`formula`	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: `~ a + b + (1|c)` Formulas with only fixed effects also work, and `lmFit()` followed by contrasts.fit() are run.
`data`	`data.frame` with columns corresponding to formula

`lib.size`	numeric vector containing total library sizes for each sample. Defaults to the normalized (effective) library sizes in `counts` if `counts` is a `DGEList` or to the columnwise count totals if `counts` is a matrix.
`normalize.method`	the microarray-style normalization method to be applied to the logCPM values (if any). Choices are as for the `method` argument of `normalizeBetweenArrays` when the data is single-channel. Any normalization factors found in `counts` will still be used even if `normalize.method="none"`.
`span`	width of the lowess smoothing window as a proportion.
`plot`	logical, should a plot of the mean-variance trend be displayed?
`save.plot`	logical, should the coordinates and line of the plot be saved in the output?
`quiet`	suppress message, default FALSE
`BPPARAM`	parameters for parallel evaluation
`...`	other arguments are passed to `lmer`.

Details

Adapted from `vomm()` in `limma` v3.40.2

Value

An `EList` object just like the result of `limma::voom()`

See Also

limma::voom()

Examples

```
# library(variancePartition)
library(edgeR)
library(BiocParallel)

data(varPartDEdata)

# normalize RNA-seq counts
dge = DGEList(counts = countMatrix)
dge = calcNormFactors(dge)

# specify formula with random effect for Individual
form <- ~ Disease + (1|Individual)

# compute observation weights
vobj = voomWithDreamWeights( dge[1:20,], form, metadata)

# fit dream model
res = dream( vobj, form, metadata)

# extract results
```

```
topTable(res, coef="Disease1")
```

Index