Package ‘variancePartition’

December 22, 2019

Type Package

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

Version 1.16.0

Date 2019-10-08

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)

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 (>= 3.5.0), ggplot2, limma, foreach, scales, Biobase, methods

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

RoxygenNote 6.1.1

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

git_branch RELEASE_3_10

git_last_commit 4df4cb

git_last_commit_date 2019-10-29

Date/Publication 2019-12-21
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.getAllUniContrasts</td>
<td>3</td>
</tr>
<tr>
<td>.isMixedModelFormula</td>
<td>3</td>
</tr>
<tr>
<td>.standard_transform</td>
<td>4</td>
</tr>
<tr>
<td>as.data.frame, varPartResults-method</td>
<td>4</td>
</tr>
<tr>
<td>as.matrix, varPartResults-method</td>
<td>5</td>
</tr>
<tr>
<td>calcVarPart</td>
<td>6</td>
</tr>
<tr>
<td>canCorPairs</td>
<td>7</td>
</tr>
<tr>
<td>classifyTestsF</td>
<td>8</td>
</tr>
<tr>
<td>classifyTestsF, MArrayLM2-method</td>
<td>9</td>
</tr>
<tr>
<td>colinearityScore</td>
<td>9</td>
</tr>
<tr>
<td>dream</td>
<td>10</td>
</tr>
<tr>
<td>eBayes, MArrayLM2-method</td>
<td>12</td>
</tr>
<tr>
<td>ESS</td>
<td>13</td>
</tr>
<tr>
<td>extractVarPart</td>
<td>14</td>
</tr>
<tr>
<td>fitExtractVarPartModel</td>
<td>15</td>
</tr>
<tr>
<td>fitVarPartModel</td>
<td>18</td>
</tr>
<tr>
<td>getContrast</td>
<td>20</td>
</tr>
<tr>
<td>getVarianceComponents</td>
<td>21</td>
</tr>
<tr>
<td>ggColorHue</td>
<td>22</td>
</tr>
<tr>
<td>MArrayLM2-class</td>
<td>23</td>
</tr>
<tr>
<td>plotCompareP</td>
<td>23</td>
</tr>
<tr>
<td>plotContrasts</td>
<td>24</td>
</tr>
<tr>
<td>plotCorrMatrix</td>
<td>25</td>
</tr>
<tr>
<td>plotCorrStructure</td>
<td>26</td>
</tr>
<tr>
<td>plotPercentBars</td>
<td>27</td>
</tr>
<tr>
<td>plotStratify</td>
<td>28</td>
</tr>
<tr>
<td>plotStratifyBy</td>
<td>29</td>
</tr>
<tr>
<td>plotVarPart</td>
<td>31</td>
</tr>
<tr>
<td>residuals, VarParFitList-method</td>
<td>32</td>
</tr>
<tr>
<td>sortCols</td>
<td>33</td>
</tr>
<tr>
<td>topTable, MArrayLM2-method</td>
<td>35</td>
</tr>
<tr>
<td>VarParCIList-class</td>
<td>35</td>
</tr>
<tr>
<td>VarParFitList-class</td>
<td>35</td>
</tr>
<tr>
<td>varParFrac-class</td>
<td>36</td>
</tr>
<tr>
<td>varPartConfInf</td>
<td>36</td>
</tr>
<tr>
<td>varPartData</td>
<td>37</td>
</tr>
<tr>
<td>varPartDEdata</td>
<td>38</td>
</tr>
<tr>
<td>varPartResults-class</td>
<td>39</td>
</tr>
<tr>
<td>voomWithDreamWeights</td>
<td>39</td>
</tr>
<tr>
<td>[.MArrayLM2</td>
<td>40</td>
</tr>
</tbody>
</table>

**Index** 42
.getAllUniContrasts  

Get all univariate contrasts

Description

Get all univariate contrasts

Usage

.getAllUniContrasts(exprObj, formula, data)

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 as a response. e.g.: ~ a + b + (1|c) Formulas with only fixed effects also work
data        data.frame with columns corresponding to formula

Value

Matrix testing each variable one at a time. Contrasts are on rows

.isMixedModelFormula  

Check if model contains a random effect

Description

Check if model contains a random effect

Usage

.isMixedModelFormula(formula, data)

Arguments

formula     model formula
data        data.frame
.standard_transform  
*Compute standard post-processing values*

**Description**

These values are typically computed by eBayes

**Usage**

```r
.standard_transform(fit)
```

**Arguments**

- `fit`: result of `dream (MArrayLM2)`

**Value**

MArrayLM2 object with values computed

---

**as.data.frame, varPartResults-method**

*Convert to data.frame*

**Description**

Convert `varPartResults` to `data.frame`

**Usage**

```r
## 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)

## S4 method for signature 'Var

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**

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.

**Usage**

```
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, 
showWarnings = TRUE, ...)
```

```
## S4 method for signature 'lm'
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, 
showWarnings = TRUE, ...)
```

```
## S4 method for signature 'lmerMod'
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, 
showWarnings = TRUE, ...)
```

**Arguments**

- **fit**: model fit from `lm()` or `lmer()`
- **adjust**: remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables
- **adjustAll**: adjust for all variables. This computes the adjusted ICC with respect to all variables
- **showWarnings**: show warnings about model fit (default TRUE)
- **...**: additional arguments (not currently used)

**Value**

Fraction of variance explained / ICC for each variable in the model
canCorPairs

Examples

```r
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

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

Arguments

- **formula**: standard 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 matrices 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 rho / sum(rho) which is the fraction of the maximum possible correlation.

Note that CCA returns correlations 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**

```r
## 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

```r
# 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
collinearityScore(res[[1]])
```

dream

**Differential expression with linear mixed model**

Description

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

Usage

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

Specify "Satterthwaite" or "Kenward-Roger" method to estimate effective degrees 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.

REML

use restricted maximum likelihood to fit linear mixed model. default is TRUE. Strongly discourage against changing this option

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

...

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 is used. For example if formula is ~ a + b + (1|c), then to model is

\[
\text{fit} \leftarrow \text{lmer}(\text{exprObj}[j,] \sim a + b + (1|c), \text{data}=\text{data})
\]

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 foreach/dopar to run loops 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 produced by lmerTest and pbkrtest packages

Value

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

Examples

# load library
# library(variancePartition)
l library(BiocParallel)

# 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 )

eBayes,MArrayLM2-method
eBayes for MArrayLM2

description

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

Value
results of eBayes

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

"full" method: if $V_x = \text{var}(Y;x)$ 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: $\text{sum}(\text{solve}(V_x))$. In practice, this can be evaluated as $\text{sum}(w)$, 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 $m^*k / (1+\rho^*(m-1))$

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, adjust = NULL, adjustAll = FALSE,
               showWarnings = TRUE, ...)

Arguments

modelList     list of lmer() model fits
adjust        remove variation from specified variables from the denominator. This computes
              the adjusted ICC with respect to the specified variables
adjustAll     adjust for all variables. This computes the adjusted ICC with respect to all vari-
              ables. This overrides the previous argument, so all variables are include in ad-
              just.
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)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
on.exit(stopCluster(cl))
```
# or by using the doSNOW package

# 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 ) )

# 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**

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

### S4 method for signature 'matrix'

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

REML = FALSE, useWeights = TRUE, weightsMatrix = NULL,
adjust = NULL, adjustAll = FALSE, 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,
adjust = NULL, adjustAll = FALSE, 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,
adjust = NULL, adjustAll = FALSE, 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,
adjust = NULL, adjustAll = FALSE, 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 as 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

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

adjust remove variation from specified variables from the denominator. This computes
the adjusted ICC with respect to the specified variables

adjustAll adjust for all variables. This computes the adjusted ICC with respect to all vari-
ables. This overrides the previous argument, so all variables are included in ad-
just.

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 is used. For example if formula is ~ a + b + (1|c), then to 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 foreach/dopar to run loops 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.

Value

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

Examples

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

# 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**

```r
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'
```r
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'
```r
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'
```r
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'
```r
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 ELList 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 as 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

useWeights  if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an ELList() 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 is used. For example if formula is \~ a + b + (1|c), then to model is
defaults = data)

If there are no random effects, so formula is \~ a + b + c, a 'standard' linear model is used:
defaults = 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 foreach/dopar to run loops 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().

Value

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

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

# 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

getVarianceComponents

Extract variance terms

Description

Extract variance terms from a model fit with lm() or lmer()

Usage

getVarianceComponents(fit)

Arguments

fit  
list of lmer() model fits
**Value**

variance explained by each variable

**Examples**

```r
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

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

fit <- modellist[[1]]
getVarianceComponents( fit )
```

---

**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)`
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)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# 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**

**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

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)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# 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 structure based on Individual, reordering samples with hclust
plotCorrStructure( fit, "Individual" )

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

# plot correlation structure 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

```r
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# 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")
```

Description

Plot gene expression stratified by another variable

Usage

```r
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-axis. Defaults to value of xval
- `ylab`: label y-axis. 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
plotStratifyBy

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 convenient 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**

**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-axis. Defaults to value of xval.
- **ylab**: label y-axis. 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**

```r
# 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)
# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# 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,VarParFitList-method

Residuals from model fit

Description

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

Usage

```r
## 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

```r
# load library
library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# 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**

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

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

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

### S4 method for signature 'varPartResults'
```r
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

```r
# library(variancePartition)
# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# 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 ) )
```
Description

toptable for MArrayLMM_lmer

Usage

```r
## S4 method for signature 'MArrayLM2'
toTable(fit, coef = NULL, number = 10,
genelist = fit$genes, adjust.method = "BH", sort.by = "p",
resort.by = NULL, p.value = 1, lfc = 0, confint = FALSE)
```

Arguments

- `fit`
- `coef`
- `number`
- `genelist`
- `adjust.method`
- `sort.by`
- `resort.by`
- `p.value`
- `lfc`
- `confint`

Value

results of toptable

VarParCIList-class

Class VarParCIList

Description

Class VarParCIList

VarParFitList-class

Class VarParFitList

Description

Class VarParFitList
**varParFrac-class**

*Class varParFrac*

**Description**

Class varParFrac

**Description**

Linear mixed model confidence intervals

**Usage**

```r
varPartConfInf(exprObj, formula, data, REML = FALSE, useWeights = TRUE,
weightsMatrix = NULL, adjust = NULL, adjustAll = FALSE,
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 ELList 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 as 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
- **useWeights**: if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an ELList 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
- **adjust**: remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables
- **adjustAll**: adjust for all variables. This computes the adjusted ICC with respect to all variables. This overrides the previous argument, so all variables are included in adjust.
- **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

```r
# load library
# library(varianc Partition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# 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 )
```
Usage
data(varPartData)
data(varPartData)
data(varPartData)
data(varPartData)
data(varPartData)

Format
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

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
varPartResults-class

**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**

```r
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 as 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
```r
# 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")
```

Description
Enable subsetting on MArrayLM2 object. Same as for MArrayLM, but apply column subsetting to
df.residual and cov.coefficients.list
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>MArrayLM2</td>
</tr>
<tr>
<td>i</td>
<td>row</td>
</tr>
<tr>
<td>j</td>
<td>col</td>
</tr>
</tbody>
</table>

### Value

- subset
Index

*Topic datasets
  varPartData, 37
  varPartDEdata, 38
  .getAllUniContrasts, 3
  .isMixedModelFormula, 3
  .standard_transform, 4
  [.MArrayLM2, 40

  as.data.frame
    (as.data.frame, varPartResults-method), 4
  as.data.frame, varPartResults-method, 4
  as.matrix
    (as.matrix, varPartResults-method), 5
  as.matrix, varPartResults-method, 5
  calcVarPart, 6
  calcVarPart, lm-method (calcVarPart), 6
  calcVarPart, lmerMod-method
    (calcVarPart), 6
  canCorPairs, 7
  classifyTestsF, 8
  classifyTestsF, MArrayLM2-method, 9
  colinearityScore, 9
  countMatrix (varPartDEdata), 38
  
dream, 10

  eBayes, MArrayLM2-method, 12
  ESS, 13
  ESS, lmerMod-method (ESS), 13
  extractVarPart, 14

  fitExtractVarPartModel, 15
  fitExtractVarPartModel, data.frame-method
    (fitExtractVarPartModel), 15
  fitExtractVarPartModel, EList-method
    (fitExtractVarPartModel), 15
  fitExtractVarPartModel, ExpressionSet-method
    (fitExtractVarPartModel), 15
  fitExtractVarPartModel, matrix-method
    (fitExtractVarPartModel), 15
  fitVarPartModel, 18

  fitVarPartModel, data.frame-method
    (fitVarPartModel), 18
  fitVarPartModel, EList-method
    (fitVarPartModel), 18
  fitVarPartModel, ExpressionSet-method
    (fitVarPartModel), 18
  fitVarPartModel, matrix-method
    (fitVarPartModel), 18

  geneCounts (varPartData), 37
  geneExpr (varPartData), 37
  getContrast, 20
  getVarianceComponents, 21
  ggColorHue, 22
  info (varPartData), 37

  MArrayLM2-class, 23
  metadata (varPartDEdata), 38

  plotCompareP, 23
  plotContrasts, 24
  plotCorrMatrix, 25
  plotCorrStructure, 26
  plotPercentBars, 27
  plotStratify, 28
  plotStratifyBy, 29
  plotVarPart, 31
  plotVarPart, data.frame-method
    (plotVarPart), 31
  plotVarPart, matrix-method
    (plotVarPart), 31
  plotVarPart, varPartResults-method
    (plotVarPart), 31

  residuals
    (residuals, VarParFitList-method), 32
  residuals, VarParFitList-method, 32

  sortCols, 33
  sortCols, data.frame-method (sortCols), 33
  sortCols, matrix-method (sortCols), 33

42
INDEX

sortCols, varPartResults-method
(sortCols), 33
subset.MArrayLM2, MArrayLM2-method
(subset.MArrayLM2), 40

topTable, MArrayLM2-method, 35
to pedestrian, MArrayLM2-method
(topTable, MArrayLM2-method), 35

VarParCIList-class, 35
VarParFitList-class, 35
varParFrac-class, 36
varPartConfInf, 36
varPartData, 37
varPartDEdata, 38
varPartResults-class, 39
voomWithDreamWeights, 39