Package ‘variancePartition’

May 10, 2023

Type Package

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

Version 1.30.0

Date 2023-04-24

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

Encoding UTF-8

URL `http://bioconductor.org/packages/variancePartition`, `https://DiseaseNeuroGenomics.github.io/variancePartition`

BugReports `https://github.com/DiseaseNeuroGenomics/variancePartition/issues`

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

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

Depends R (>= 4.0.0), ggplot2, limma, BiocParallel

Imports MASS, pbkrtest (>= 0.4-4), lmerTest, Matrix (>= 1.4.0), iterators, foreach, doParallel, gplots, RhpcBLASctl, progress, reshape2, remaCor (>= 0.0.11), aod, scales, Rdpack, rlang, lme4 (>= 1.1-10), grDevices, graphics, Biobase, methods, utils, stats

RoxygenNote 7.2.3

RdMacros Rdpack

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

git_branch RELEASE_3_17

git_last_commit 1d94eb7

git_last_commit_date 2023-04-25

Date/Publication 2023-05-09

Author Gabriel Hoffman [aut, cre]

R topics documented:

`.getAllUniContrasts` *Get all univariate contrasts*

Description

Get all univariate contrasts

Usage

```
.getAllUniContrasts(formula, data)
```

Arguments

`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)
```

Arguments

`formula`	model formula

`.standard_transform` *Compute standard post-processing values*

Description

These values are typically computed by eBayes

Usage

```
.standard_transform(fit, sigma = fit$sigma)
```

Arguments

`fit`	result of dream (MArrayLM2)
`sigma`	vector of standard errors used to compute t-statistic. Can be maximum likelihood estimates, or posterior means

Value

MArrayLM2 object with values computed

`applyQualityWeights` *Apply pre-specified sample weights*

Description

Apply pre-specified sample weights by scaling existing precision weights

Usage

```
applyQualityWeights(vobj, weights)
```

Arguments

`vobj`	`EList` from `voom` or `voomWithDreamWeights`.
`weights`	sample level weights

Details

Apply pre-specified sample-level weights to the existing precision weights estimated from the data. While the `limma::voomWithQualityWeights` function of Lui et al. (2015) estimates the sample-level weights from `voom` fit, here the weights are fixed beforehand.

References

Liu R, Holik AZ, Su S, Jansz N, Chen K, Leong HS, Blewitt ME, Asselin-Labat M, Smyth GK, Ritchie ME (2015). “Why weight? Modelling sample and observational level variability improves power in RNA-seq analyses.” *Nucleic acids research*, **43**(15), e97–e97.

See Also

`limma::voomWithQualityWeights`

`as.data.frame.varPartResults` *Convert to data.frame*

Description

Convert varPartResults to data.frame

Usage

```
## S3 method for class '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

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, showWarnings = TRUE, returnFractions = TRUE, ...)

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

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

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

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

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

Arguments

`fit`	model fit from lm() or lmer()
`showWarnings`	show warnings about model fit (default TRUE)
`returnFractions`	default: TRUE. If TRUE return fractions that sum to 1. Else return unscaled variance components.
`...`	additional arguments (not currently used)

Details

For linear model, variance fractions are computed based on the sum of squares explained by each component. For the linear mixed model, 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

For the Poisson distribution with rate λ, the variance is $log(1 + 1/\lambda)$.

For the negative binomial distribution with rate λ and shape θ, the variance is $log(1 + 1/\lambda + 1/\theta)$.

Variance decomposition is reviewed by Nakagawa and Schielzeth (2012), and expanded to other GLMs by Nakagawa, Johnson and Schielzeth (2017). See McKelvey and Zavoina (1975) for early work on applying to GLMs. Also see DeMaris (2002)

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).

Value

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

References

Nakagawa S, Johnson PC, Schielzeth H (2017). "The coefficient of determination R 2 and intraclass correlation coefficient from generalized linear mixed-effects models revisited and expanded." *Journal of the Royal Society Interface*, **14**(134), 20170213.

Nakagawa S, Schielzeth H (2013). “A general and simple method for obtaining R2 from generalized linear mixed-effects models.” *Methods in ecology and evolution*, **4**(2), 133–142.

McKelvey RD, Zavoina W (1975). “A statistical model for the analysis of ordinal level dependent variables.” *Journal of mathematical sociology*, **4**(1), 103–120.

DeMaris A (2002). “Explained variance in logistic regression: A Monte Carlo study of proposed measures.” *Sociological Methods & Research*, **31**(1), 27–74.

Hoffman GE, Schadt EE (2016). “variancePartition: interpreting drivers of variation in complex gene expression studies.” *BMC bioinformatics*, **17**(1), 1–13.

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]])
```

---

deviation *Deviation from expectation for each observation*

Description

Given a model fit for each features, residuals are computed and transformed based on an absolute value or squaring transform.

Usage

```
deviation(fit, method = c("AD", "SQ"), scale = c("leverage", "none"))

## S4 method for signature 'MArrayLM'
deviation(fit, method = c("AD", "SQ"), scale = c("leverage", "none"))
```

Arguments

`fit`	model fit from `dream()`
`method`	transform the residuals using absolute deviation ("AD") or squared deviation ("SQ").
`scale`	scale each observation by "leverage", or no scaling ("none")

See Also

`diffVar()`

Examples

```
# library(variancePartition)
library(edgeR)
data(varPartDEdata)

# filter genes by number of counts
isexpr = rowSums(cpm(countMatrix)>0.1) >= 5

# Standard usage of limma/voom
dge = DGEList( countMatrix[isexpr,] )
dge = calcNormFactors( dge )

# make this vignette faster by analyzing a subset of genes
dge = dge[1:1000,]

# regression formula
form <- ~ Disease
```

```
# estimate precision weights
vobj = voomWithDreamWeights( dge, form, metadata )

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

# Compute deviation from expection for each observation
# using model residuals
z = deviation(fit)
z[1:4, 1:4]
```

`diffVar` *Test differential variance*

Description

Test the association between a covariate of interest and the response's deviation from expectation.

Usage

```
diffVar(
  fit,
  method = c("AD", "SQ"),
  scale = c("leverage", "none"),
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'MArrayLM'
diffVar(
  fit,
  method = c("AD", "SQ"),
  scale = c("leverage", "none"),
  BPPARAM = SerialParam(),
  ...
)
```

Arguments

`fit`	model fit from `dream()`
`method`	transform the residuals using absolute deviation ("AD") or squared deviation ("SQ").
`scale`	scale each observation by "leverage", or no scaling ("none")
`BPPARAM`	parameters for parallel evaluation
`...`	other parameters passed to `dream()`

Details

This method performs a test of differential variance between two subsets of the data, in a way that generalizes to multiple categories, continuous variables and metrics of spread beyond variance. For the two category test, this method is simular to Levene's test. This model was adapted from Phipson, et al (2014), extended to linear mixed models, and adapted to be compatible with `dream()`.

This method is composed of multiple steps where 1) a typical linear (mixed) model is fit with `dream()`, 2) residuals are computed and transformed based on an absolute value or squaring transform, 3) a second regression is performed with `dream()` to test if a variable is associated with increased deviation from expectation. Both regression take advantage of the `dream()` linear (mixed) modelling framework followed by empirical Bayes shrinkage that extends the `limma::voom()` framework.

Note that `diffVar()` takes the results of the first regression as a parameter to use as a starting point.

References

Phipson B, Oshlack A (2014). "DiffVar: a new method for detecting differential variability with application to methylation in cancer and aging." *Genome biology*, **15**(9), 1–16.

See Also

`missMethyl::diffVar()`, `car::leveneTest()`

Examples

```
# library(variancePartition)
library(edgeR)
data(varPartDEdata)

# filter genes by number of counts
isexpr = rowSums(cpm(countMatrix)>0.1) >= 5

# Standard usage of limma/voom
dge = DGEList( countMatrix[isexpr,] )
dge = calcNormFactors( dge )

# make this vignette faster by analyzing a subset of genes
dge = dge[1:1000,]

# regression formula
form <- ~ Disease

# estimate precision weights
vobj = voomWithDreamWeights( dge, form, metadata )

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

# fit differential variance model
res = diffVar( fit )
```

```
# extract results for differential variance based on Disease
topTable(res, coef = "Disease1", number=3)

# Box plot of top hit
# Since ASCL3 has a negative logFC,
# the deviation from expectation is *smaller* in
# Disease==1 compared to baseline.
gene = "ENST00000325884.1 gene=ASCL3"
boxplot(vobj$E[gene,] ~ metadata$Disease, main=gene)
```

`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("adaptive", "Satterthwaite", "Kenward-Roger"),
  useWeights = TRUE,
  weightsMatrix = NULL,
  control = vpcontrol,
  suppressWarnings = FALSE,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  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 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`	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. "adaptive" (Default) uses KR for <= 10 samples.
`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 (Hoffman and Roussos, 2021). 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)

References

Hoffman GE, Roussos P (2021). “dream: Powerful differential expression analysis for repeated measures designs.” *Bioinformatics*, **37**(2), 192–201.

Examples

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

# load simulated data:
# geneExpr: matrix of *normalized* 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
# NOTE: dream() runs on *normalized* data
fit = dream( geneExpr[1:10,], form, info)
fit = eBayes(fit)

# view top genes
topTable( fit, coef="Batch2", number=3 )

# get contrast matrix testing if the coefficient for Batch3 is
# different from coefficient for Batch2
# Name this comparison as 'compare_3_2'
# The variable of interest must be a fixed effect
L = makeContrastsDream(form, info, contrasts=c(compare_3_2 = "Batch3 - Batch2"))

# plot contrasts
plotContrasts( L )

# Fit linear mixed model for each gene
# run on just 10 genes for time
fit2 = dream( geneExpr[1:10,], form, info, L)
fit2 = eBayes(fit2)

# view top genes for this contrast
topTable( fit2, coef="compare_3_2", number=3)

# 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)
fit3 = eBayes(fit3)
```

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

# view top genes
topTable( fit4, coef="compare_3_2", number=3 )

# Compute residuals using dream
residuals(fit4)[1:4, 1:4]
```

`dscchisq` *Scaled chi-square*

Description

Scaled chi-square density using a gamma distribution

Usage

```
dscchisq(x, a, b)
```

Arguments

`x`	vector of quantiles.
`a`	scale
`b`	degrees of freedom

`eBayes,MArrayLM2-method` *eBayes for MArrayLM2*

Description

eBayes for result of linear mixed model for with `dream()` using residual degrees of freedom approximated with `rdf.merMod()`

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

Value

results of eBayes using approximated residual degrees of freedom

See Also

dream rdf.merMod

`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)

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 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 = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'matrix'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
```

```
  ...
)

## S4 method for signature 'data.frame'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'EList'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'ExpressionSet'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)
```

```
## S4 method for signature 'sparseMatrix'
fitExtractVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)
```

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. 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)

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 <- Biobase::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 = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'matrix'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'data.frame'
```

```
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'EList'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'ExpressionSet'
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'sparseMatrix'
```

```
fitVarPartModel(
  exprObj,
  formula,
  data,
  REML = FALSE,
  useWeights = TRUE,
  weightsMatrix = NULL,
  showWarnings = TRUE,
  fxn = identity,
  control = vpcontrol,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)
```

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. 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)

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 <- Biobase::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 as 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
```

`getTreat` *Test if coefficient is different from a specified value*

Description

Test if coefficient is different from a specified value

Usage

```
getTreat(fit, lfc = log2(1.2), coef = 1, number = 10, sort.by = "p")

## S4 method for signature 'MArrayLM'
getTreat(fit, lfc = log2(1.2), coef = 1, number = 10, sort.by = "p")

## S4 method for signature 'MArrayLM2'
getTreat(fit, lfc = log2(1.2), coef = 1, number = 10, sort.by = "p")
```

Arguments

`fit`	fit
`lfc`	a minimum log2-fold-change below which changes not considered scientifically meaningful
`coef`	which coefficient to test
`number`	number of genes to return
`sort.by`	column to sort by

Value

results of getTreat

Examples

```
data(varPartData)

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

fit = dream( geneExpr, form, info)
fit = eBayes(fit)

coef = 'Age'

# Evaluate treat()/topTreat() in a way that works seamlessly for dream()
getTreat(fit, lfc=log2(1.03), coef, sort.by="none", number=3)
```

`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)
```

`hatvalues,MArrayLM-method` *Compute hatvalues*

Description

Compute hatvalues from dream fit

Usage

```
## S4 method for signature 'MArrayLM'
hatvalues(model, vobj, ...)

## S4 method for signature 'MArrayLM2'
hatvalues(model, ...)
```

Arguments

`model`	model fit from `dream()`
`vobj`	`EList` returned by `voom()` or `voomWithDreamWeights()`.
`...`	other arguments, currently ignored

`isRunableFormula` *Test if formula is full rank on this dataset*

Description

Test if formula is full rank on this dataset

Usage

```
isRunableFormula(exprObj, formula, data)
```

Arguments

`exprObj`	expression object
`formula`	formula
`data`	data

`makeContrastsDream` *Construct Matrix of Custom Contrasts*

Description

Construct the contrast matrix corresponding to specified contrasts of a set of parameters. Each specified set of contrast weights must sum to 1.

Usage

```
makeContrastsDream(
  formula,
  data,
  ...,
  contrasts = NULL,
  suppressWarnings = FALSE,
  nullOnError = FALSE
)
```

Arguments

`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
`...`	expressions, or character strings which can be parsed to expressions, specifying contrasts
`contrasts`	character vector specifying contrasts
`suppressWarnings`	(default FALSE). suppress warnings for univariate contrasts
`nullOnError`	(default FALSE). When a contrast entry is invalid, throw warning and return NULL for that contrast entry

Details

This function expresses contrasts between a set of parameters as a numeric matrix. The parameters are usually the coefficients from a linear (mixed) model fit, so the matrix specifies which comparisons between the coefficients are to be extracted from the fit. The output from this function is usually used as input to `dream()`.

This function creates a matrix storing the contrasts weights that are applied to each coefficient.

Consider a variable `v` with levels `c('A', 'B', 'C')`. A contrast comparing `A` and `B` is `'vA - vB'` and tests whether the difference between these levels is different than zero. Coded for the 3 levels this has weights `c(1, -1, 0)`. In order to compare `A` to the other levels, the contrast is `'vA - (vB + vC)/2'` so that `A` is compared to the average of the other two levels. This is encoded as `c(1, -0.5, -0.5)`. This type of proper matching in testing multiple levels is enforced by ensuring that the contrast weights sum to 1. Based on standard regression theory only weighted sums of the estimated coefficients are supported.

This function is inspired by `limma::makeContrasts()` but is designed to be compatible with linear mixed models for `dream()`

Names in ... and contrasts will be used as column names in the returned value.

Value

matrix of linear contrasts between regression coefficients

See Also

`plotContrasts()`

Examples

```
# load library
# library(variancePartition)

library(BiocParallel)

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

form <- ~ 0 + Batch + (1|Individual) + (1|Tissue)

# Define contrasts
# Note that for each contrass, the weights sum to 1
L = makeContrastsDream( form, info, contrasts = c(Batch1_vs_2 = "Batch1 - Batch2", Batch3_vs_4 = "Batch3 - Batch4",

# show contrasts matrix
L

# Plot to visualize contrasts matrix
plotContrasts(L)

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

# examine contrasts after fitting
head(coef(fit))

# show results from first contrast
topTable(fit, coef="Batch1_vs_2")

# show results from second contrast
topTable(fit, coef="Batch3_vs_4")
```

```
# show results from third contrast
topTable(fit, coef="Batch1_vs_34")
```

`MArrayLM2-class` *Class MArrayLM2*

Description

Class `MArrayLM2`

`mvTest` *Multivariate tests on results from* `dream()`

Description

Evaluate multivariate tests on results from `dream()` using `vcov()` to compute the covariance between estimated regression coefficients across multiple responses. A joint test to see if the coefficients are jointly different from zero is performed using meta-analysis methods that account for the covariance.

Usage

```
mvTest(
  fit,
  vobj,
  features,
  coef,
  method = c("FE", "RE2C", "tstat", "sidak", "fisher"),
  progressbar = TRUE
)

## S4 method for signature 'MArrayLM,EList,vector'
mvTest(
  fit,
  vobj,
  features,
  coef,
  method = c("FE", "RE2C", "tstat", "sidak", "fisher"),
  progressbar = TRUE
)

## S4 method for signature 'MArrayLM,EList,missing'
mvTest(
  fit,
```

```
  vobj,
  features,
  coef,
  method = c("FE", "RE2C", "tstat", "sidak", "fisher"),
  progressbar = TRUE
)

## S4 method for signature 'MArrayLM,EList,list'
mvTest(
  fit,
  vobj,
  features,
  coef,
  method = c("FE", "RE2C", "tstat", "sidak", "fisher"),
  progressbar = TRUE
)
```

Arguments

`fit`	`MArrayLM` or `MArrayLM2` returned by `dream()`
`vobj`	matrix or `EList` object returned by `voom()`
`features`	a) indeces or names of features to perform multivariate test on, b) list of indeces or names. If missing, perform joint test on all features.
`coef`	name of coefficient or contrast to be tested
`method`	statistical method used to perform multivariate test. See details. `'FE'` is a fixed effect test that models the covariance between coefficients. `'RE2C'` is a random effect test of heterogeneity of the estimated coefficients that models the covariance between coefficients, and also incorporates a fixed effects test too. `'tstat'` combines the t-statistics and models the covariance between coefficients. `'sidak'` returns the smallest p-value and accounting for the number of tests. `'fisher'` combines the p-value using Fisher's method assuming independent tests.
`progressbar`	if TRUE, show progress bar

Details

See package `remaCor` for details about the `remaCor::RE2C()` test, and see `remaCor::LS()` for details about the fixed effect test. When only 1 feature is selected, the original p-value is returned and the test statistic is set to `NA`.

For the `"RE2C"` test, the final test statistic is the sum of a test statistic for the mean effect (`stat.FE`) and heterogeneity across effects (`stat.het`).

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
fit = dream( vobj, form, metadata)
fit = eBayes(fit)

# Multivariate test of features 1 and 2
mvTest(fit, vobj, 1:2, coef="Disease1")

# Test multiple sets of features
lst = list(a = 1:2, b=3:4)
mvTest(fit, vobj, lst, coef="Disease1")
```

`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)

library(BiocParallel)

# 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

Details

This plot shows the contrasts weights that are applied to each coefficient.

Consider a variable `v` with levels `c('A', 'B', 'C')`. A contrast comparing `A` and `B` is `'vA - vB'` and tests whether the difference between these levels is different than zero. Coded for the 3 levels this has weights `c(1, -1, 0)`. In order to compare `A` to the other levels, the contrast is `'vA - (vB + vC)/2'` so that `A` is compared to the average of the other two levels. This is encoded as `c(1, -0.5, -0.5)`. This type of proper matching in testing multiple levels is enforced by ensuring that the contrast weights sum to 1. Based on standard regression theory only weighted sums of the estimated coefficients are supported.

Value

ggplot2 object

See Also

`makeContrastsDream()`

Examples

```
# load library
# library(variancePartition)

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

# 1) get contrast matrix testing if the coefficient for Batch2 is different from Batch3
form <- ~ Batch + (1|Individual) + (1|Tissue)
L = makeContrastsDream(form, info, contrasts=c(Batch_3_vs_2 = "Batch3 - Batch2"))

# plot contrasts
plotContrasts( L )
```

`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)

library(BiocParallel)

# 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 gene fractions*

Description

Bar plot of fractions for a subset of genes

Usage

```
plotPercentBars(
  x,
  col = c(ggColorHue(ncol(x) - 1), "grey85"),
  genes = rownames(x),
  width = NULL,
  ...
)

## S4 method for signature 'matrix'
plotPercentBars(
  x,
  col = c(ggColorHue(ncol(x) - 1), "grey85"),
```

```
  genes = rownames(x),
  width = NULL,
  ...
)

## S4 method for signature 'data.frame'
plotPercentBars(
  x,
  col = c(ggColorHue(ncol(x) - 1), "grey85"),
  genes = rownames(x),
  width = NULL,
  ...
)

## S4 method for signature 'varPartResults'
plotPercentBars(
  x,
  col = c(ggColorHue(ncol(x) - 1), "grey85"),
  genes = rownames(x),
  width = NULL,
  ...
)
```

Arguments

`x`	object storing fractions
`col`	color of bars for each variable
`genes`	name of genes to plot
`width`	specify width of bars
`...`	other arguments

Value

Returns ggplot2 barplot

Examples

```
# 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
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)
```

`plotVarianceEstimates` *Plot Variance Estimates*

Description

Plot Variance Estimates

Usage

```
plotVarianceEstimates(
  fit,
  fitEB,
  var_true = NULL,
  xmax = quantile(fit$sigma^2, 0.999)
)
```

Arguments

`fit`	model fit from `dream()`
`fitEB`	model fit from `eBayes()`
`var_true`	array of true variance values from simulation (optional)
`xmax`	maximum value on the x-axis

`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)

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)

varPart <- fitExtractVarPartModel( geneExpr, form, info )

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

`rdf.merMod` *Approximate residual degrees of freedom*

Description

For a linear model with n samples and p covariates, $RSS/\sigma^2 \sim \chi^2_\nu$ where $\nu = n - p$ is the residual degrees of freedom. In the case of a linear mixed model, the distribution is no longer exactly a chi-square distribution, but can be approximated with a chi-square distribution.

Given the hat matrix, H, that maps between observed and fitted responses, the approximate residual degrees of freedom is $\nu = tr((I - H)^T (I - H))$. For a linear model, this simplifies to the well known form $\nu = n - p$. In the more general case, such as a linear mixed model, the original form simplifies only to $n - 2tr(H) + tr(HH)$ and is an approximation rather than being exact. The third term here is quadratic time in the number of samples, n, and can be computationally expensive to evaluate for larger datasets. Here we develop a linear time algorithm that takes advantage of the fact that H is low rank.

H is computed as $A^T A + B^T B$ for `A=CL` and `B=CR` defined in the code. Since A and B are low rank, there is no need to compute H directly. Instead, the terms $tr(H)$ and $tr(HH)$ can be computed using the eigen decompositions of AA^T and BB^T which is linear time in the number of samples.

Usage

```
rdf.merMod(model, method = c("linear", "quadratic"))
```

Arguments

`model`	An object of class `merMod`
`method`	Use algorithm that is "linear" (default) or quadratic time in the number of samples

Details

Compute the approximate residual degrees of freedom from a linear mixed model.

See Also

rdf_from_matrices

Examples

```
library(lme4)

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

# Evaluate the approximate residual degrees of freedom
rdf.merMod(fit)
```

`rdf_from_matrices` *Fast approximate residual degrees of freedom*

Description

Defining $H = A^T A + B^T B$ where A and B are low rank, compute $n - 2tr(H) + tr(HH)$ in $O(np^2)$ instead of $O(n^2p^2)$.

Usage

```
rdf_from_matrices(A, B)
```

Arguments

`A`	a `matrix` or `sparseMatrix`
`B`	a `matrix` or `sparseMatrix`

See Also

rdf.merMod

`reOnly` *Adapted from lme4:::reOnly*

Description

Adapted from lme4:::reOnly

Usage

```
reOnly(f, response = FALSE)
```

Arguments

`f`	formula
`response`	(FALSE) is there a response in the formula

`residuals,MArrayLM-method`

residuals for MArrayLM

Description

residuals for MArrayLM

Usage

```
## S4 method for signature 'MArrayLM'
residuals(object, y, ..., type = c("response", "pearson"))
```

Arguments

`object`	MArrayLM object from dream
`y`	`EList` object used in `dream()`
`...`	other arguments, currently ignored
`type`	compute either response or pearson residuals

Value

results of residuals

`residuals,MArrayLM2-method`

residuals for MArrayLM2

Description

residuals for MArrayLM2

Usage

```
## S4 method for signature 'MArrayLM2'
residuals(object, y, type = c("response", "pearson"), ...)
```

Arguments

`object`	MArrayLM2 object from dream
`y`	`EList` object used in `dream()`
`type`	compute either response or pearson residuals
`...`	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)

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)

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

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

`residuals.MArrayLM2` *Residuals for result of dream*

Description

Residuals for result of dream

Usage

```
residuals.MArrayLM2(object, y, ..., type = c("response", "pearson"))
```

Arguments

`object`	See `?stats::residuals`
`y`	`EList` object used in `dream()`
`...`	See `?stats::residuals`
`type`	compute either response or pearson residuals

`shrinkageMetric` *Shrinkage metric for eBayes*

Description

Evaluates the coefficient from the linear regression of `s2.post ~ sigmaSq`. When there is no shrinkage, this value is 1. Values less than 1 indicate the amount of shrinkage.

Usage

```
shrinkageMetric(sigmaSq, s2.post)
```

Arguments

`sigmaSq`	maximum likelihood residual variance for every gene
`s2.post`	empirical Bayes posterior estimate of residual variance for every gene

Details

Shrinkage metric for eBayes quantifying the amount of shrinkage that is applied to shrink the maximum likelihood residual variance to the empirical Bayes posterior estimate

`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)

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
# sort columns by median value
plotVarPart( sortCols( varPart ) )
```

`topTable` *Table of Top Genes from Linear Model Fit*

Description

topTable generic

topTable generic MArrayLM

topTable generic MArrayLM2

Usage

```
topTable(
  fit,
  coef = NULL,
  number = 10,
  genelist = fit$genes,
  adjust.method = "BH",
  sort.by = "B",
  resort.by = NULL,
  p.value = 1,
  lfc = 0,
  confint = FALSE
)

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

## S4 method for signature 'MArrayLM2'
topTable(
  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`	fit
`coef`	coef
`number`	number

`genelist`	genelist
`adjust.method`	adjust.method
`sort.by`	sort.by
`resort.by`	resort.by
`p.value`	p.value
`lfc`	lfc
`confint`	confint

Value

results of toptable

results of toptable

results of toptable

`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 = vpcontrol,
  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 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, 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)

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)

# 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

A simulated dataset of gene counts

A simulated dataset of gene counts

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` *A simulated dataset of gene counts*

Description

- 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

Usage

```
data(varPartData)
```

```
data(varPartData)
```

Format

A dataset of 24 samples and 19,364 genes

A dataset of 24 samples and 19,364 genes

`varPartResults-class` *Class varPartResults*

Description

Class `varPartResults`

`vcov,MArrayLM-method` *Co-variance matrix for* `dream()` *fit*

Description

Define generic `vcov()` for result of `lmFit()` and `dream()`

Usage

```
## S4 method for signature 'MArrayLM'
vcov(object, vobj, coef)
```

Arguments

`object`	`MArrayLM` object return by `lmFit()` or `dream()`
`vobj`	`EList` object returned by `voom()`
`coef`	name of coefficient to be extracted

`vcov,MArrayLM2-method` *Co-variance matrix for* `dream()` *fit*

Description

Define generic `vcov()` for result of `lmFit()` and `dream()`

Usage

```
## S4 method for signature 'MArrayLM2'
vcov(object, vobj, coef)
```

Arguments

`object`	`MArrayLM` object return by `lmFit()` or `dream()`
`vobj`	`EList` object returned by `voom()`
`coef`	name of coefficient to be extracted

`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,
  weights = NULL,
  plot = FALSE,
  save.plot = FALSE,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)
```

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.
`weights`	Can be a numeric matrix of individual weights of same dimensions as the `counts`, or a numeric vector of sample weights with length equal to `ncol(counts)`
`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)
res = eBayes(res)

# extract results
topTable(res, coef="Disease1", number=3)
```

`[.MArrayLM2` *Subseting for MArrayLM2*

Description

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

Arguments

`object`	MArrayLM2
`i`	row
`j`	col

Value

subset

Index