Package 'variancePartition'

April 23, 2016

Type Package

Title Quantifying and interpreting drivers of variation in complex gene expression studies

Version 1.0.7

Date 2016-2-18

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 linear mixed model to quantify variation in gene expression attributable to individual, tissue, time point, or technical variables.

VignetteBuilder knitr

License GPL (>= 2)

Suggests edgeR, knitr, BiocStyle

biocViews RNASeq, GeneExpression, Regression, Software

Depends ggplot2, foreach, Biobase, methods

Imports lme4, iterators, reshape2, doParallel, limma, dendextend

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

`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

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 )
```

---

colinearityScore *Colinearity score*

Description

Colinearity 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 colinearity score between 0 and 1, where a score > 0.999 means the degree of colinearity is too high. This function reports the correlation matrix between coefficient estimates for fixed effects. The colinearity 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 colinearity score on the first model fit
# this reports the correlation matrix between coefficients estimates
# for fixed effects
# the colinearity score is the maximum absolute correlation value
# If the colinearity score > .999 then the variance parition
# estimates may be problematic
# In that case, a least one variable should be omitted
colinearityScore(res[[1]])
```

`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 variables. This overrides the previous argument, so all variables are include in adjust.
`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)
# 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 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 )

# stop cluster
stopCluster(cl)
```

`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, adjust = NULL,
  adjustAll = FALSE, showWarnings = TRUE,
  control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
  "stop.deficient"), ...)

## S4 method for signature 'matrix'
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"), ...)

## 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"), ...)

## 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"), ...)

## 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"), ...)
```

Arguments

exprObj	matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package
formula	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: ~ a + b + (1|c)
data	data.frame with columns corresponding to formula
REML	use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option
useWeights	if TRUE, analysis uses heteroskadatistic 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 variables. This overrides the previous argument, so all variables are include in adjust.
showWarnings	show warnings about model fit (default TRUE)
control	control settings for lmer()
...	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 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 omited 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)

# 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)

# 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 ) )

# 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 )

# stop cluster
stopCluster(cl)
```

`fitVarPartModel` *Fit linear (mixed) model*

Description

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

Usage

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

## S4 method for signature 'matrix'
fitVarPartModel(exprObj, formula, data, REML = FALSE,
  useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE,
  fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE,
```

```
  check.rankX = "stop.deficient"), ...)

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

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

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

Arguments

exprObj	matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package
formula	specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: ~ a + b + (1|c)
data	data.frame with columns corresponding to formula
REML	use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option
useWeights	if TRUE, analysis uses heteroskadatistic 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()
...	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 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 omited 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

```
# 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)

# 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
# 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 )

# stop cluster
stopCluster(cl)
```

`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

```
# 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 )

# stop cluster
stopCluster(cl)
```

---

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)
```

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), "#bebebe99"))
```

Arguments

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

Value

Returns ggplot2 barplot

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)
# 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")

# stop cluster
stopCluster(cl)
```

plotStratifyBy *plotStratifyBy*

Description

Plot gene expression stratified by another variable

Usage

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

Arguments

geneExpr	data.frame of gene expression values and another variable for each sample. If there are multiple columns, the user can specify which one to use
xval	name of column in geneExpr to be used along x-axis to stratify gene expression
yval	name of column in geneExpr indicating gene expression
xlab	label x-asis. Defaults to value of xval
ylab	label y-asis. Defaults to value of yval
main	main label
sortBy	name of column in geneExpr to sort samples by. Defaults to xval
colorBy	name of column in geneExpr to color box plots. Defaults to xval
sort	if TRUE, sort boxplots by median value, else use default ordering
text	plot text on the top left of the plot
text.y	indicate position of the text on the y-axis as a fraction of the y-axis range
text.size	size of text
pts.cex	size of points
ylim	specify range of y-axis
legend	show legend
x.labels	show x axis labels

Value

ggplot2 object

Examples

```
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

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

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

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

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

`plotVarPart` *Violin plot of variance fractions*

Description

Violin plot of variance fraction for each gene and each variable

Usage

```
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1), "#bebebe99"),
  label.angle = 20, ylim = c(0, 100), main = "", ...)

## S4 method for signature 'matrix'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
  "#bebebe99"), label.angle = 20, ylim = c(0, 100), main = "", ...)

## S4 method for signature 'data.frame'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
  "#bebebe99"), label.angle = 20, ylim = c(0, 100), main = "", ...)

## S4 method for signature 'varPartResults'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
  "#bebebe99"), label.angle = 20, ylim = c(0, 100), main = "", ...)
```

Arguments

obj	varParFrac object returned by fitExtractVarPart or extractVarPart
col	vector of colors
label.angle	angle of labels on x-axis
ylim	limits of y-axis
main	title of plot
...	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 ) )

# stop cluster
stopCluster(cl)
```

`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)

# 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 )

# stop cluster
stopCluster(cl)
```

---

sortCols *Sort variance parition statistics*

Description

Sort columns returned by extractVarPart() or fitExtractVarPartModel()

Usage

```
sortCols(x, FUN = median, decreasing = TRUE, ...)

## S4 method for signature 'matrix'
sortCols(x, FUN = median, decreasing = TRUE, ...)

## S4 method for signature 'data.frame'
sortCols(x, FUN = median, decreasing = TRUE, ...)

## S4 method for signature 'varPartResults'
sortCols(x, FUN = median, decreasing = TRUE, ...)
```

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?
...	other arguments to sort

Value

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

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

# stop cluster
stopCluster(cl)
```

---

varPartData *Simulation dataset for examples*

Description

A simulated dataset of gene expression and metadata

Usage

```
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

Value

varPartData

Index