variancePartition: Quantifying and interpreting drivers of variation in multilevel gene expression experiments

Gabriel Hoffman

Pamela Sklar Division of Psychiatric Genomics
Icahn Institute for Genomics and Multiscale Biology
Department of Genetics and Genomic Sciences
Icahn School of Medicine at Mount Sinai

October 1, 2019

Abstract

Gene expression datasets are complicated and have multiple sources of biological and technical variation. These datasets have recently become more complex as it is now feasible to assay gene expression from the same individual in multiple tissues or at multiple time points. The *variancePartition* package implements a statistical method to quantify the contribution of multiple sources of variation and decouple within/between-individual variation. In addition, *variancePartition* produces results at the gene-level to identity genes that follow or deviate from the genome-wide trend.

variancePartition version: 1.14.1

Contents

1 Overview

The *variancePartition* package provides a general framework for understanding drivers of variation in gene expression in experiments with complex designs. A typical application would consider a dataset of gene expression from individuals sampled in multiple tissues or multiple time points where the goal is to understand variation within versus between individuals and tissues. *variancePartition* use a linear mixed model to partition the variance attributable to multiple variables in the data. The analysis is built on top of the *lme4* package [1], and some basic knowledge about linear mixed models will give you some intuition about the behavior of *variancePartition* [2, 3]

1.1 Inputs

There are three components to an analysis:

1) **Gene expression data:** In general, this is a matrix of normalized gene expression values with genes as rows and experiments as columns.

   – Count-based quantification: `featureCounts` [4], `HTSeq` [5]
   Counts mapping to each gene can be normalized using counts per million (CPM), reads per kilobase per million (RPKM) or fragments per kilobase per million (FPKM). These count results can be processed with *limma*/`voom` [6] to model the precision of each observation or *DESeq2* [7].

   – Isoform quantification: `kallisto` [8], `sailfish` [9], `salmon` [10], `RSEM` [11], `cufflinks` [12]
   These perform isoform-level quantification using reads that map to multiple transcripts. Quantification values can be read directly into *R*, or processed with *ballgown* [13] or *tximport* [14].

   – Microarray data: any standard normalization such as `rma` in the *oligo* [15] package can be used.

2) **Meta-data about each experiment:** A `data.frame` with information about each experiment such as patient ID, tissue, sex, disease state, time point, batch, etc.

2) **Formula indicating which meta-data variables to consider:** An R formula such as `~ Age + (1|Individual) + (1|Tissue) + (1|Batch)` indicating which meta-data variables should be used in the analysis.

variancePartition will assess the contribution of each meta-data variable to variation in gene expression and can report the intra-class correlation for each variable.

2 Running an analysis

A typical analysis with *variancePartition* is only a few lines of *R* code, assuming the expression data has already been normalized. Normalization is a separate topic, and I address it briefly in Section 5.

The simulated dataset included as an example contains measurements of 200 genes from 100 samples. These samples include assays from 3 tissues across 25 individuals processed in 4 batches. The individuals range in age from 36 to 73. A typical *variancePartition* analysis will assess the contribution of each aspect of the study design (i.e. individual, tissue, batch, age) to the expression variation of each gene. The analysis will prioritize these axes of variation based on a genome-wide summary and give results at the gene-level to identity genes that follow or deviate from this genome-wide trend. The results can be visualized using custom plots and can be used for downstream analysis.

2.1 Standard application

```
# 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 model it as a fixed effect
# Individual and Tissue are both categorical,
# so model them as random effects
# Note the syntax used to specify random effects
form <- ~ Age + (1|Individual) + (1|Tissue) + (1|Batch)

# Fit model and extract results
# 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 fixed effects,
# a linear model is used
# each entry in results is a regression model fit on a single gene
# 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 variables
# after correcting for all other variables
# Note that geneExpr can either be a matrix,
# and EList output by voom() in the limma package,
# or an ExpressionSet
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# sort variables (i.e. columns) by median fraction
# of variance explained
vp <- sortCols( varPart )

# Figure 1a
# Bar plot of variance fractions for the first 10 genes
plotPercentBars( vp[1:10,] )
#
# Figure 1b
# violin plot of contribution of each variable to total variance
plotVarPart( vp )
```

variancePartition includes a number of custom plots to visualize the results. Since *variancePartition* attributes the fraction of total variation attributable to each aspect of the study design, these fractions naturally sum to 1. `plotPercentBars` plots the partitioning results for a subset of genes (Figure 1a), and `plotVarPart` shows a genome-wide violin plot of the distribution of variance explained by each variable across all genes (Figure 1b). (Note that these plots show results in terms of percentage of variance explained, while the results are stored in terms of the fraction.)

The core functions of *variancePartition* work seemlessly with gene expression data stored as a `matrix`, `data.frame`, `EList` from *limma* or `ExpressionSet` from *Biobase*. `fitExtractVarPartModel()` returns an object that stores the variance fractions for each gene and each variable in the formula specified. These fractions can be accessed just like a `data.frame`:

```
# Access first entries
head(varPart)
```

```
##          Batch Individual Tissue      Age Residuals
## gene1 0.000158      0.890 0.0247 4.53e-05    0.0847
## gene2 0.000000      0.806 0.1009 3.34e-04    0.0926
## gene3 0.002423      0.890 0.0356 1.47e-03    0.0704
```

```
## gene4 0.000000      0.769 0.1253 1.01e-03    0.1048
## gene5 0.000000      0.700 0.2091 3.87e-05    0.0912
## gene6 0.002347      0.722 0.1679 2.72e-03    0.1049

# Access first entries for Individual
head(varPart$Individual)

## [1] 0.890 0.806 0.890 0.769 0.700 0.722

# sort genes based on variance explained by Individual
head(varPart[order(varPart$Individual, decreasing=TRUE),])

##           Batch Individual   Tissue      Age Residuals
## gene43  0.00000      0.914 1.17e-02 3.78e-04    0.0735
## gene174 0.00000      0.911 9.72e-03 2.01e-03    0.0770
## gene111 0.00000      0.907 8.39e-03 9.74e-04    0.0839
## gene127 0.00000      0.904 1.38e-02 5.08e-04    0.0821
## gene151 0.00608      0.903 1.43e-10 1.35e-05    0.0910
## gene91  0.00000      0.900 1.41e-02 1.11e-06    0.0856
```

2.1.1 Saving plot to file

In order to save the plot to a file, use the `ggsave` function:

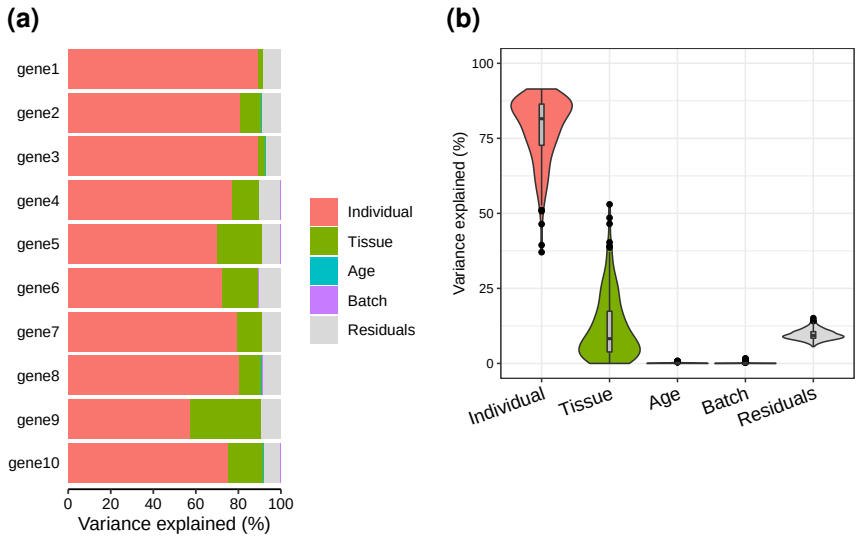


Figure 1: ***variancePartition*** **example on simulated data**

```
      fig <- plotVarPart( vp )
      ggsave(file, fig)
```

2.2 Plot expression stratified by other variables

variancePartition also includes plotting functions to visualize the variation across a variable of interest. `plotStratify` plots the expression of a gene stratified by the specified variable. In the example dataset, users can plot a gene expression trait stratified by Tissue (Figure 2a) or Individual (Figure 2b).

```
# get gene with the highest variation across Tissues
# create data.frame with expression of gene i and Tissue
# type for each sample
i <- which.max( varPart$Tissue )
GE <- data.frame( Expression = geneExpr[i,], Tissue = info$Tissue)

# Figure 2a
# plot expression stratified by Tissue
plotStratify( Expression ~ Tissue, GE, main=rownames(geneExpr)[i])
#
# get gene with the highest variation across Individuals
# create data.frame with expression of gene i and Tissue
# type for each sample
i <- which.max( varPart$Individual )
GE <- data.frame( Expression = geneExpr[i,],
        Individual = info$Individual)

# Figure 2b
# plot expression stratified by Tissue
label <- paste("Individual:", format(varPart$Individual[i]*100,
        digits=3), "%")
main <- rownames(geneExpr)[i]
plotStratify(  Expression ~ Individual, GE, colorBy=NULL,
        text=label, main=main)
```

For gene141, variation across tissues explains 52.9% of variance in gene expression. For gene43, variation across Individuals explains 91.4% of variance in gene expression.

2.3 Intuition about the backend

At the heart of `variancePartition`, a regression model is fit for each gene separately and summary statistics are extracted and reported to the user for visualization and downstream analysis. For a single model fit, `calcVarPart`

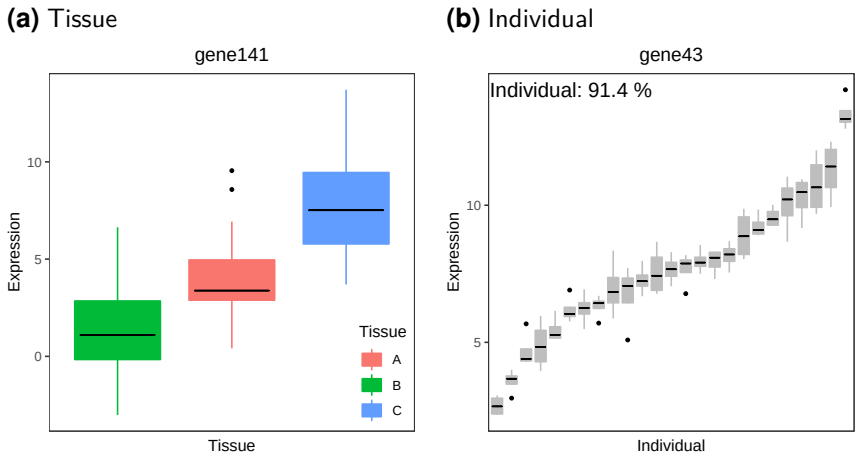


Figure 2: Plot gene expression stratified by a) Tissue and b) Individual

computes the fraction of variance explained by each variable. `calcVarPart` is defined by this package, and computes these statistics from either a fixed effects model fit with `lm` or a linear mixed model fit with `lmer`. `fitExtractVarPart` loops over each gene, fits the regression model and returns the variance fractions reported by `calcVarPart`.

Fitting the regression model and extracting variance statistics can also be done directly:

```
library('lme4')

# fit regression model for the first gene
form_test <- geneExpr[1,] ~ Age + (1|Individual) + (1|Tissue)
fit <- lmer(form_test, info, REML=FALSE )

# extract variance statistics
calcVarPart(fit)

## Individual     Tissue        Age  Residuals
##   8.90e-01   2.47e-02   4.35e-05   8.50e-02
```

3 Interpretation

variancePartition fits a linear (mixed) model thats jointly considers the contribution of all specified variables on the expression of each gene. It uses a multiple regression model so that the effect of each variable is assessed while correcting for all others. Therefore, fitting the model with each variable separately will give very different results from the considering all variables jointly. I have found joint analysis the best option in most cases.

The results of *variancePartition* give insight into the expression data at multiple levels. Moreover, a single statistic often has multiple equivalent interpretations while only one is relevant to the biological question. Analysis of the example data in Figure 1 gives some strong interpretations.

Considering the median across all genes,

1) variation across individuals explains a median of 81.5% of the variation in expression, after correcting for tissue, batch and age
2) variation across tissues explains a median of 8.2% of the variation in expression, after correcting for other the variables
3) variation across batches is negligible after correcting for variation due to other variables
4) the effect of age is negligible after correcting for other variables
5) correcting for individual, tissue, batch and age leaves a median of 9.3% of the total variance in expression.

These statistics also have a natural interpretation in terms of the intra-class correlation (ICC), the correlation between observations made from samples in the same group.

Considering the median across across all genes and all experiments,

1) the ICC for individual is 81.5%.
2) the ICC for tissue is 8.2%.
3) two randomly selected gene measurements from same individual, but regardless of tissue, batch or age, have a correlation of 81.5%.
4) two randomly selected gene measurements from same tissue, but regardless of individual, batch or age, have a correlation of 8.2%.
5) two randomly selected gene measurements from the same individual *and* same tissue, but regardless of batch and age, have an correlation of 81.5% + 8.2% = 89.7%.

Note that that the ICC here is interpreted as the ICC after correcting for all other variables in the model.

These conclusions are based on the genome-wide median across all genes, but the same type of statements can be made at the gene-level. Moreover, care must be taken in the interpretation of nested variables. For example, `Age` is nested within `Individual` since the multiple samples from each individual are taken at the same age. Thus the effect of `Age` removes some variation from being explained by `Individual`. This often arises when considering variation across individuals and across sexes: any cross-sex variation is a component of the cross-individual variation. So the total variation across individuals is the sum of the fraction of variance explained by `Sex` and `Individual`. This nesting/summing of effects is common for variables that are properties of the individual rather than the sample. For example, sex and ethnicity are always properties of the individual. Variables like age and disease state can be properties of the individual, but could also vary in time-course or longitudinal experiments. The the interpretation depends on the experimental design.

The real power of *variancePartition* is to identify specific genes that follow or deviate from the genome-wide trend. The gene-level statistics can be used to identify a subset of genes that are enriched for specific biological functions. For example, we can ask if the 500 genes with the highest variation in expression across tissues (i.e. the long tail for tissue in Figure 1a) are enriched for genes known to have high tissue-specificity.

3.1 Should a variable be modeled as fixed or random effect?

Categorical variables should (almost) always be modeled as a random effect. The difference between modeling a categorical variable as a fixed versus random effect is minimal when the sample size is large compared to the number of categories (i.e. levels). So variables like disease status, sex or time point will not be sensitive to modeling as a fixed versus random effect. However, variables with many categories like `Individual` *must* be modeled as a random effect in order to obtain statistically valid results. So to be on the safe side, categorical variable should be modeled as a random effect.

variancePartition fits two types of models:

1) linear mixed model where all categorical variables are modeled as random effects and all continuous variables are fixed effects. The function `lmer` from *lme4* is used to fit this model.

2) fixed effected model, where all variables are modeled as fixed effects. The function `lm` is used to fit this model.

3.2 Which variables should be included?

In my experience, it is useful to include all variables in the first analysis and then drop variables that have minimal effect. However, like all multiple regression methods, *variancePartition* will divide the contribution over multiple variables that are strongly correlated. So, for example, including both sex and height in the model will show sex having a smaller contribution to variation gene expression than if height were omitted, since there variables are strongly correlated. This is a simple example, but should give some intuition about a common issue that arises in analyses with *variancePartition*.

variancePartition can naturally assess the contribution of both individual and sex in a dataset. As expected, genes for which sex explains a large fraction of variation are located on chrX and chrY. If the goal is to interpret the impact of sex, then there is no issue. But recall the issue with correlated variables and note that individual is correlated with sex, because each individual is only one sex regardless of how many samples are taken from a individual. It follows that including sex in the model reduces the apparent contribution of individual to gene expression. In other words, the ICC for individual will be different if sex is included in the model.

In general, including variables in the model that do not vary within individual will reduce the apparent contribution of individual as estimated by *variancePartition*. For example, sex and ethnicity never vary between multiple samples from the same individual and will always reduce the apparent contribution of individual. However, disease state and age may or may not vary depending on the study design.

In biological datasets technical variability (i.e. batch effects) can often reduce the apparent biological signal. In RNA-seq analysis, it is common for the the impact of this technical variability to be removed before downstream analysis. Instead of including these batch variable in the *variancePartition* analysis, it is simple to complete the expression residuals with the batch effects removed and then feeds these residuals to *variancePartition*. This will increase the fraction of variation explained by biological variables since technical variability is reduced.

3.2.1 Assess correlation between all pairs of variables

Evaluating the correlation between variables in a important part in interpreting variancePartition results. When comparing two continuous variables, Pearson correlation is widely used. But variancePartition includes categorical variables in the model as well. In order to accommodate the correlation between a continuous and a categorical variable, or two categorical variables we used canonical correlation analysis.

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.

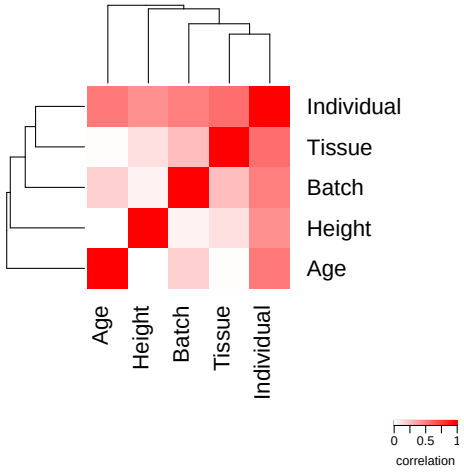
Statistically, let rho be the array of correlation values returned by the standard R function cancor to compute CCA. canCorPairs returns rho / sum(rho) which is the fraction of the maximum possible correlation. Note that CCA returns correlations values between 0 and 1

```
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 )
```

Figure 3: Assess correlation between all pairs of variables



4 Advanced analysis

4.1 Extracting additional information from model fits

Advanced users may want to perform the model fit and extract results in separate steps in order to examine the fit of the model for each gene. Thus the work of `fitExtractVarPart` can be divided into two steps: 1) fit the regression model, and 2) extracting variance statistics.

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

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

# Extract results
varPart <- extractVarPart( results )
```

Note that storing the model fits can use a lot of memory (~10Gb with 20K genes and 1000 experiments). I do not recommend unless you have a specific need for storing the entire model fit.

Instead, `fitVarPartModel` can extract any desired information using any function that accepts the model fit from `lm`/`lmer`. The results are stored in a `list` and can be used for downstream analysis.

```
# Fit model and run summary() function on each model fit
vpSummaries <- fitVarPartModel( geneExpr, form, info, fxn=summary )
```

```
# Show results of summary() for the first gene
vpSummaries[[1]]
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: expr ~ Age + (1 | Individual) + (1 | Tissue) + (1 | Batch)
##    Data: data2
## Weights: gene14643$weights
## Control: control
##
##      AIC      BIC   logLik deviance df.resid
##      397      413     -193      385       94
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.0580 -0.5865  0.0147  0.6603  1.9708
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  Individual (Intercept) 10.82431 3.2900
##  Batch      (Intercept)  0.00192 0.0438
##  Tissue     (Intercept)  0.30008 0.5478
##  Residual                1.02993 1.0149
## Number of obs: 100, groups:  Individual, 25; Batch, 4; Tissue, 3
##
## Fixed effects:
##              Estimate Std. Error t value
## (Intercept) -10.60242    1.09494   -9.68
## Age           0.00318    0.01610    0.20
##
## Correlation of Fixed Effects:
##     (Intr)
## Age -0.739
```

4.2 Removing batch effects before fitting model

Gene expression studies often have substantial batch effects, and *variancePartition* can be used to understand the magnitude of the effects. However, we often want to focus on biological variables (i.e. individual, tissue, disease, sex)

after removing the effect of technical variables. Depending on the size of the batch effect, I have found it useful to correct for the batch effect first and then perform a *variancePartition* analysis afterward. Subtracting this batch effect can reduce the total variation in the data, so that the contribution of other variables become clearer.

Standard analysis:

```
form <- ~ (1|Tissue) + (1|Individual) + (1|Batch) + Age
varPart <- fitExtractVarPartModel( geneExpr, form, info )
```

Analysis on residuals:

```
library('limma')
# subtract out effect of Batch
fit <- lmFit( geneExpr, model.matrix(~ Batch, info))
res <- residuals( fit, geneExpr)

# fit model on residuals
form <- ~ (1|Tissue) + (1|Individual) + Age

varPartResid <- fitExtractVarPartModel( res, form, info )
```

Remove batch effect with linear mixed model

```
# subtract out effect of Batch with linear mixed model
modelFit <- fitVarPartModel( geneExpr, ~ (1|Batch), info )
res <- residuals( modelFit )

# fit model on residuals
form <- ~ (1|Tissue) + (1|Individual) + Age

varPartResid <- fitExtractVarPartModel( res, form, info )
```

If the two-step process requires too much memory, the residuals can be computed more efficiently. Here, run the `residuals` function inside the call to `fitVarPartModel` to avoid storing the large intermediate results.

```
# extract residuals directly without storing intermediate results
residList <- fitVarPartModel( geneExpr, ~ (1|Batch), info,
        fxn=residuals )

# convert list to matrix
residMatrix = do.call(rbind, residList)
```

4.3 Variation within multiple subsets of the data

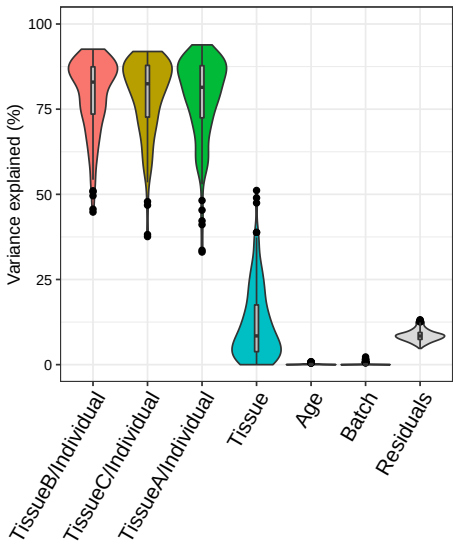
So far, we have focused on interpreting one variable at a time. But the linear mixed model behind *variancePartition* is a very powerful framework for analyzing variation at multiple levels. We can easily extend the previous analysis of the contribution of individual and tissue on variation in gene expression to examine the contribution of individual *within* each tissue. This analysis is as easy as specifying a new formula and rerunning *variancePartition*. Note that is analysis will only work when there are replicates for at least some individuals within each tissue in order to assess cross-individual variance with in a tissue.

```
# specify formula to model within/between individual variance
# separately for each tissue
# Note that including +0 ensures each tissue is modeled explicitly
# Otherwise, the first tissue would be used as baseline
form <- ~ (Tissue+0|Individual) + Age + (1|Tissue) + (1|Batch)

# fit model and extract variance percents
varPart <- fitExtractVarPartModel( geneExpr, form, info, showWarnings=FALSE )

# violin plot
plotVarPart( sortCols(varPart), label.angle=60 )
```

Figure 4: Variation across individuals within each tissue



This analysis corresponds to a varying coefficient model, where the correlation between individuals varies for each tissue [2]. Since the variation across individuals is modeled within each tissue, the total variation explained does not sum

to 1 as it does for standard application of *variancePartition*. So interpretation as intra-class does not strictly apply and use of `plotPercentBars` is no longer applicable. Yet the variables in the study design are still ranked in terms of their genome-wide contribution to expression variation, and results can still be analyzed at the gene level. See Section 7.3 for statistical details.

4.4 Detecting problems caused by collinearity of variables

Including variables that are highly correlated can produce misleading results and overestimate the contribution of variables modeled as fixed effects. This is usually not an issue, but can arise when statistically redundant variables are included in the model. In this case, the model is "degenerate" or "computationally singular" and parameter estimates from this model are not meaningful. Dropping one or more of the covariates will fix this problem.

A check of collinearity is built into `fitVarPartModel` and `fitExtractVarPartModel`, so the user will be warned if this is an issue.

Alternatively, the user can use the `colinearityScore` function to evaluate whether this is an issue for a single model fit:

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

# fit model
res <- fitVarPartModel( geneExpr[1:4,], form, info )
```

```
# evaluate the collinearity score on the first model fit
# this reports the correlation matrix between coefficient estimates
# for fixed effects
# the collinearity score is the maximum absolute correlation value
# If the collinearity score > .99 then the variance partition
# estimates may be problematic
# In that case, a least one variable should be omitted
colinearityScore( res[[1]] )
```

```
## [1] 0.777
## attr(,"vcor")
##             (Intercept)     Age  Height
## (Intercept)       1.000 -0.4191 -0.7774
## Age              -0.419  1.0000 -0.0575
```

```
## Height           -0.777 -0.0575  1.0000
```

4.5 Including weights computed separately

variancePartition automatically used precision weights computed by *voom*, but the user can also specify custom weights using the `weightsMatrix` argument.

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

# Specify custom weights
# In this example the weights are simulated from a
# uniform distribution and are not meaningful.
weights <- matrix(runif(length(geneExpr)), nrow=nrow(geneExpr))

# Specify custom weights
res <- fitExtractVarPartModel( geneExpr[1:4,], form, info,
        weightsMatrix=weights[1:4,] )
```

In addition, setting the `useWeights=FALSE` will suppress usage of the weights in all cases, i.e. when the weights are specified manually or implicitly with the results of `voom`.

4.6 Including interaction terms

Typical analysis assumes that the effect of each variable on gene expression does not depend on other variables in the model. Sometimes this assumption is too strict, and we want to model an interaction effect whereby the effect of `Batch` depends on `Tissue`. This can be done easly by specifying an interaction term, `(1|Batch:Tissue)`. Since `Batch` has 4 categories and `Tissue` has 3, this interaction term implicity models a new `3*4 = 12` category variable in the analysis. This new interaction term will absorb some of the variance from the `Batch` and `Tissue` term, so an interaction model should always include the two constituent variables.

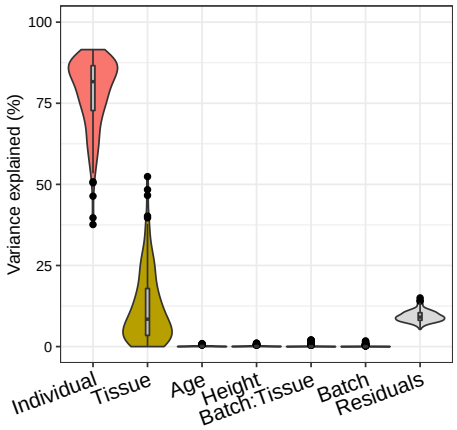
Here we fit an interaction model, but we observe that interaction between `Batch` and `Tissue` does not explain much expression variation.

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

```
# fit model
vpInteraction <- fitExtractVarPartModel( geneExpr, form, info )

plotVarPart( sortCols( vpInteraction ) )
```

Figure 5: Fit interaction term



5 Applying *variancePartition* to RNA-seq expression data

variancePartition works with gene expression data that has already been processed and normalized as for differential expression analysis.

5.1 Gene-level counts

`featureCounts` [4] and `HTSeq` [5] report the number of reads mapping to each gene (or exon). These results are easily read into *R*. *limma*/`voom` and *DESeq2* are widely used for differential expression analysis of gene- and exon-level counts and can be used to process data before analysis with *variancePartition*. This section addresses processing and normalization of gene-level counts, but the analysis is the same for exon-level counts.

5.1.1 *limma*/`voom`

Read RNA-seq counts into *R*, normalize for library size within and between experiments with TMM [16], estimate precision weights with *limma*/`voom`.

```
library('limma')
library('edgeR')

# identify genes that pass expression cutoff
isexpr <- rowSums(cpm(geneCounts)>1) >= 0.5 * ncol(geneCounts)

# create data structure with only expressed genes
gExpr <- DGEList(counts=geneCounts[isexpr,])

# Perform TMM normalization
gExpr <- calcNormFactors(gExpr)

# Specify variables to be included in the voom() estimates of
# uncertainty.
# Recommend including variables with a small number of categories
# that explain a substantial amount of variation
design <- model.matrix( ~ Batch, info)

# Estimate precision weights for each gene and sample
```

```
# This models uncertainty in expression measurements
vobjGenes <- voom(gExpr, design )

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

# variancePartition seamlessly deals with the result of voom()
# by default, it seamlessly models the precision weights
# This can be turned off with useWeights=FALSE
varPart <- fitExtractVarPartModel( vobjGenes, form, info )
```

5.1.2 *DESeq2*

Process and normalize the gene-level counts before running *variancePartition* analysis.

```
library('DESeq2')

# create DESeq2 object from gene-level counts and metadata
dds <- DESeqDataSetFromMatrix(countData = geneCounts,
        colData = info,
        design = ~ 1)

# Estimate library size correction scaling factors
dds <- estimateSizeFactors(dds)

# identify genes that pass expression cutoff
isexpr <- rowSums(fpm(dds)>1) >= 0.5 * ncol(dds)

# compute log2 Fragments Per Million
# Alternatively, fpkm(), vst() or rlog() could be used
quantLog <- log2( fpm( dds )[isexpr,] + 1)

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

# Run variancePartition analysis
varPart <- fitExtractVarPartModel( quantLog, form, info)
```

Note that DESeq2 does not compute precision weights like *limma*/`voom`, so they are not used in this version of the analysis.

5.2 Isoform quantification

Other software performs isoform-level quantification using reads that map to multiple transcripts. These include `kallisto` [8], `sailfish` [9], `salmon` [10], `RSEM` [11] and `cufflinks` [12].

5.2.1 *tximport*

Quantifications from `kallisto`, `salmon`, `sailfish` and `RSEM` can be read into *R* and processed with the Bioconductor package *tximport*. The gene- or transcript-level quantifications can be used directly in *variancePartition*.

```
library('tximportData')
library('tximport')
library('readr')

# Get data from folder where tximportData is installed
dir <- system.file("extdata", package = "tximportData")
samples <- read.table(file.path(dir, "samples.txt"), header = TRUE)
files <- file.path(dir, "kallisto", samples$run, "abundance.tsv")
names(files) <- paste0("sample", 1:6)

tx2gene <- read.csv(file.path(dir, "tx2gene.csv"))

# reads results from kallisto
txi <- tximport(files, type = "kallisto", tx2gene = tx2gene,
        countsFromAbundance = "lengthScaledTPM")

# define metadata (usually read from external source)
info_tximport <- data.frame( Sample = sprintf("sample%d", 1:6),
        Disease=c("case", "control")[c(rep(1, 3), rep(2, 3) )] )

# Extract counts from kallisto
y <- DGEList( txi$counts )

# compute library size normalization
y <- calcNormFactors(y)

# apply voom to estimate precision weights
design <- model.matrix( ~ Disease, data = info_tximport)
vobj <- voom(y, design)
```

```
# define formula
form <- ~ (1|Disease)

# Run variancePartition analysis (on only 10 genes)
varPart_tx <- fitExtractVarPartModel( vobj[1:10,], form,
        info_tximport)
```

Code to process results from `sailfish`, `salmon`, `RSEM` is very similar.

See tutorial at http://bioconductor.org/packages/release/bioc/vignettes/tximport/inst/doc/tximport.html for more details.

5.2.2 *ballgown*

Quantifications from Cufflinks/Tablemaker and RSEM can be processed and read into *R* with the Bioconductor package *ballgown*.

```
library('ballgown')

# Get data from folder where ballgown is installed
data_directory <- system.file('extdata', package='ballgown')

# Load results of Cufflinks/Tablemaker
bg <- ballgown(dataDir=data_directory, samplePattern='sample',
        meas='all')

# extract gene-level FPKM quantification
# Expression can be convert to log2-scale if desired
gene_expression <- gexpr(bg)

# extract transcript-level FPKM quantification
# Expression can be convert to log2-scale if desired
transcript_fpkm <- texpr(bg, 'FPKM')

# define metadata (usually read from external source)
info_ballgown <- data.frame( Sample = sprintf("sample%02d", 1:20),
        Batch = rep(letters[1:4], 5),
        Disease=c("case", "control")[c(rep(1, 10), rep(2, 10) )] )

# define formula
form <- ~ (1|Batch) + (1|Disease)
```

```
# Run variancePartition analysis
# Gene-level analysis
varPart_gene <- fitExtractVarPartModel( gene_expression, form,
        info_ballgown)

# Transcript-level analysis
varPart_transcript <- fitExtractVarPartModel( transcript_fpkm, form,
        info_ballgown)
```

Note that `ballgownrsem` can be used for a similar analysis of `RSEM` results.

See tutorial at http://bioconductor.org/packages/release/bioc/vignettes/ballgown/inst/doc/ballgown.html for more details.

6 Comparison with other methods on simulated data

Characterizing drivers of variation in gene expression data has typically relied on principal components analysis (PCA) and hierarchical clustering. Here I apply these methods to two simulated datasets to demonstrate the additional insight from an analysis with *variancePartition*. Each simulated dataset comprises 60 experiments from 10 individuals and 3 tissues with 2 biological replicates. In the first dataset, tissue is the major driver of variation in gene expression(Figure 6). In the second dataset, individual is the major driver of variation in gene expression (Figures 7).

Analysis of simulated data illustrates that PCA identifies the major driver of variation when tissue is dominant and there are only 3 categories. But the results are less clear when individual is dominant because there are now 10 categories. Meanwhile, hierarchical clustering identifies the major driver of variation in both cases, but does not give insight into the second leading contributor.

Analysis with *variancePartition* has a number of advantages over these standard methods:

- *variancePartition* provides a natural interpretation of multiple variables
  - figures from PCA/hierarchical clustering allow easy interpretation of only one variable
- *variancePartition* quantifies the contribution of each variable
  - PCA/hierarchical clustering give only a visual representation
- *variancePartition* interprets contribution of each variable to each gene individually for downstream analysis
  - PCA/hierarchical clustering produces genome-wide summary and does not allow gene-level interpretation
- *variancePartition* can assess contribution of one variable (i.e. Individual) separately in subset of the data defined by another variable (i.e. Tissue)

Figure 6: **Similarity within Tissue is dominant**

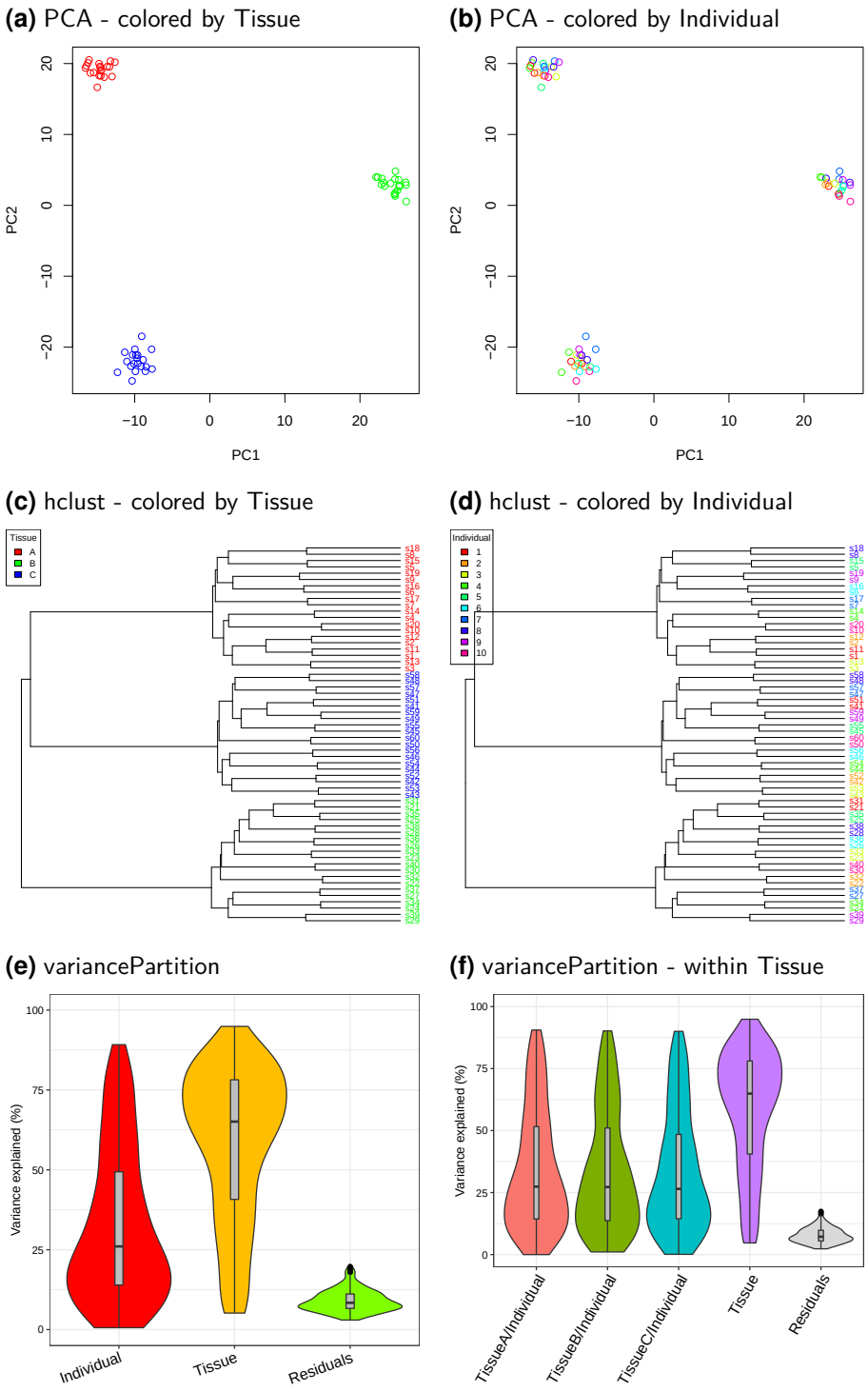
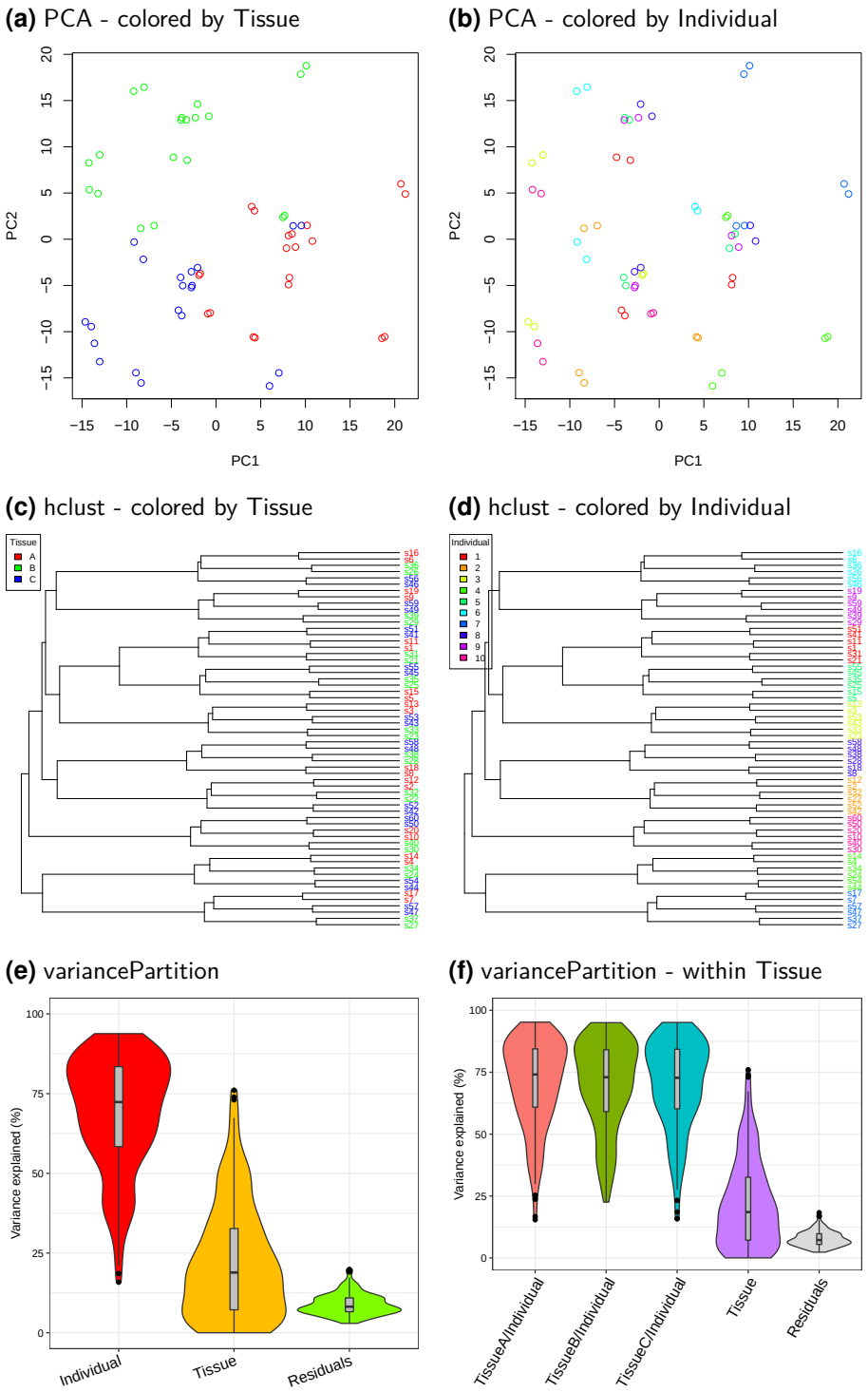


Figure 7: Similarity within Individual is dominant



7 Statistical details

A *variancePartition* analysis evaluates the linear (mixed) model

$$y = \sum_j X_j \beta_j + \sum_k Z_k \alpha_k + \varepsilon \quad (1)$$

$$\alpha_k \sim \mathcal{N}(0, \sigma^2_{\alpha_k}) \quad (2)$$

$$\varepsilon \sim \mathcal{N}(0, \sigma^2_{\varepsilon}) \quad (3)$$

where y is the expression of a single gene across all samples, X_j is the matrix of j^{th} fixed effect with coefficients β_j, Z_k is the matrix corresponding to the k^{th} random effect with coefficients α_k drawn from a normal distribution with variance $\sigma^2_{\alpha_k}$. The noise term, ε, is drawn from a normal distribution with variance σ^2_{ε}. Parameters are estimated with maximum likelihood, rather than REML, so that fixed effect coefficients, β_j, are explicitly estimated.

I use the term "linear (mixed) model" here since *variancePartition* works seamlessly when a fixed effects model (i.e. linear model) is specified.

Variance terms for the fixed effects are computed using the *post hoc* calculation

$$\hat{\sigma}^2_{\beta_j} = var(X_j \hat{\beta}_j). \quad (4)$$

For a fixed effects model, this corresponds to the sum of squares for each component of the model.

For a standard application of the linear mixed model, where the effect of each variable is additive, the fraction of variance explained by the j^{th} fixed effect is

$$\frac{\hat{\sigma}^2_{\beta_j}}{\sum_j \hat{\sigma}^2_{\beta_j} + \sum_k \hat{\sigma}^2_{\alpha_k} + \hat{\sigma}^2_{\varepsilon}}, \quad (5)$$

by the k^{th} random effect is

$$\frac{\hat{\sigma}^2_{\alpha_k}}{\sum_j \hat{\sigma}^2_{\beta_j} + \sum_k \hat{\sigma}^2_{\alpha_k} + \hat{\sigma}^2_{\varepsilon}}, \quad (6)$$

and the residual variance is

$$\frac{\hat{\sigma}^2_{\varepsilon}}{\sum_j \hat{\sigma}^2_{\beta_j} + \sum_k \hat{\sigma}^2_{\alpha_k} + \hat{\sigma}^2_{\varepsilon}}. \quad (7)$$

7.1 Implementation in *R*

An *R* formula is used to define the terms in the fixed and random effects, and `fitVarPartModel` fits the specified model for each gene separately. If random effects are specified, `lmer` from *lme4* is used behind the scenes to fit the model, while `lm` is used if there are only fixed effects. `fitVarPartModel` returns a list of the model fits, and `extractVarPart` returns the variance partition statistics for each model in the list. `fitExtractVarPartModel` combines the actions of `fitVarPartModel` and `extractVarPart` into one function call. `calcVarPart` is called behind the scenes to compute variance fractions for both fixed and mixed effects models, but the user can also call this function directly on a model fit with `lm`/`lmer`.

7.2 Interpretation of percent variance explained

The percent variance explained can be interpreted as the intra-class correlation (ICC) when a special case of Equation 1 is used. Consider the simplest example of the i^{th} sample from the k^{th} individual

$$y_{i,k} = \mu + Z\alpha_{i,k} + e_{i,k} \quad [8]$$

with only an intercept term, one random effect corresponding to individual, and an error term. In this case ICC corresponds to the correlation between two samples from the same individual. This value is equal to the fraction of variance explained by individual. For example, consider the correlation between samples from the same individual:

$$\begin{aligned} ICC &= cor(y_{1,k}, y_{2,k}) & [9] \\ &= cor(\mu + Z\alpha_{1,k} + e_{1,k}, \mu + Z\alpha_{2,k} + e_{2,k}) & [10] \\ &= \frac{cov(\mu + Z\alpha_{1,k} + e_{1,k}, \mu + Z\alpha_{2,k} + e_{2,k})}{\sqrt{var(\mu + Z\alpha_{1,k} + e_{1,k}) var(\mu + Z\alpha_{2,k} + e_{2,k})}} & [11] \\ &= \frac{cov(Z\alpha_{1,k}, Z\alpha_{2,k})}{\sigma_\alpha^2 + \sigma_\varepsilon^2} & [12] \\ &= \frac{\sigma_\alpha^2}{\sigma_\alpha^2 + \sigma_\varepsilon^2} & [13] \end{aligned}$$

The correlation between samples from different individuals is:

$$
\begin{aligned}
&= cor(y_{1,1}, y_{1,2}) && (14)\\
&= cor(\mu + Z\alpha_{1,1} + e_{1,1}, \mu + Z\alpha_{1,2} + e_{1,2}) && (15)\\
&= \frac{cov(Z\alpha_{1,1}, Z\alpha_{1,2})}{\sigma^2_\alpha + \sigma^2_\varepsilon} && (16)\\
&= \frac{0}{\sigma^2_\alpha + \sigma^2_\varepsilon} && (17)\\
&= 0 && (18)
\end{aligned}
$$

This interpretation in terms of fraction of variation explained (FVE) naturally generalizes to multiple variance components. Consider two sources of variation, individual and cell type with variances σ^2_{id} and σ^2_{cell}, respectively. Applying a generalization of the the previous derivation, two samples are correlated according to:

Individual	cell type	variance	Interpretation	Correlation value
same	different	$\frac{\sigma^2_{id}}{\sigma^2_{id}+\sigma^2_{cell}+\sigma^2_\varepsilon}$	FVE by individual	$ICC_{individual}$
different	same	$\frac{\sigma^2_{cell}}{\sigma^2_{id}+\sigma^2_{cell}+\sigma^2_\varepsilon}$	FVE by cell type	ICC_{cell}
same	same	$\frac{\sigma^2_{id}+\sigma^2_{cell}}{\sigma^2_{id}+\sigma^2_{cell}+\sigma^2_\varepsilon}$	sum of FVE by individual & cell type	$ICC_{individual,cell}$
different	different	$\frac{0}{\sigma^2_{id}+\sigma^2_{cell}+\sigma^2_\varepsilon}$	sample are independent	

Notice that the correlation between samples from the same individual and same cell type corresponds to the sum of the fraction explained by individual + fraction explained by cell type. This defines ICC for individual and tissue, as well as the combined ICC and relates these values to FVE.

In order to illustrate how this FVE and ICC relate to the correlation between samples in multilevel datasets, consider a simple example of 5 samples from 2 individuals and 2 tissues:

Sample	Individual	Cell type
a	1	T-Cell
b	1	T-Cell
c	1	monocyte
d	2	T-Cell
e	2	monocyte

Modeling the separate effects of individual and tissue gives the following covariance structure between samples when a linear mixed model is used:

$$cov(y) = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} \sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon & & & & \\ \sigma^2_{id} + \sigma^2_{cell} & \sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon & & & \\ \sigma^2_{id} & \sigma^2_{id} & \sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon & & \\ \sigma^2_{cell} & \sigma^2_{cell} & 0 & \sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon & \\ 0 & 0 & \sigma^2_{cell} & \sigma^2_{id} & \sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon \end{pmatrix} \end{array}$$

The covariance matrix is symmetric so that blank entries take the value on the opposite side of the diagonal. The covariance can be converted to correlation by dividing by $\sigma^2_{id} + \sigma^2_{cell} + \sigma^2_\varepsilon$, and this gives the results from above. This example generalizes to any number of variance components [2].

7.3 Variation with multiple subsets of the data

The linear mixed model underlying *variancePartition* allows the effect of one variable to depend on the value of another variable. Statistically, this is called a varying coefficient model [2, 3]. This model arises in *variancePartition* analysis when the variation explained by individual depends on tissue or cell type.

A given sample is only from one cell type, so this analysis asks a question about a subset of the data. The the data is implicitly divided into subsets base on cell type and variation explained by individual is evaluated within each subset. The data is not actually divided onto subset, but the statistical model

essentially examples samples with each cell type. This subsetting means that the variance fractions do not sum to 1.

Consider a concrete example with variation from across individual and cell types (T-cells and monocytes) with data from the i^{th} sample from the k^{th} individual, sex of s and cell type c. Modeling the variation across individuals within cell type corresponds to

$$y_{i,k,s,c} = \mu + Z^{(sex)}\alpha_{i,s} + Z^{(Tcell|id)}\alpha_{i,k,c} + Z^{(monocyte|id)}\alpha_{i,k,c} + e_{i,k,s,c} \quad (19)$$

with corresponding variance components:

Variance component	Interpretation
σ^2_{sex}	variance across sex (which is the same for all cell types)
$\sigma^2_{(Tcell\|id)}$	variance across individuals within T-cells
$\sigma^2_{(monocyte\|id)}$	variance across individuals within monocytes
σ^2_{ε}	residual variance

Since the dataset is now divided into multiple subsets, direct interpretation of the fraction of variation explained (FVE) as intra-class correlation does not apply. Instead, we compute a "pseudo-FVE" by approximating the total variance attributable to cell type by using a weighted average of the within cell type variances weighted by the sample size within each cell type. Thus the values of pseudo-FVE do not have the simple interpretation as in the standard application of *variancePartition*, but allows ranking of variables based on genome-wide contribution to variance and analysis of gene-level results.

7.4 Relationship between *variancePartition* and differential expression

Differential expression (DE) is widely used to identify gene which show difference is expression between two subsets of the data (i.e. case versus controls). For a single gene, DE analysis measures the difference in mean expression between the two subsets. (Since expression is usually analyzed on a log scale, DE results are usually shown in terms of log fold changes between the two subsets). In Figure 8, consider two simulated examples of a gene whose expression differs between males and females. The mean expression in males is 0 and the

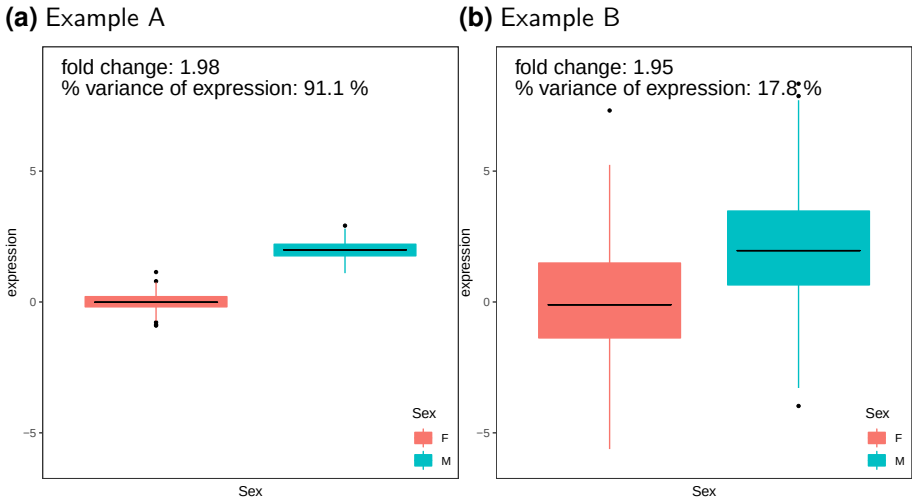
mean expression in females is 2 in both cases. Therefore, the fold change is 2 in both cases.

However, the fraction of expression variation explained by sex is very different in these two examples. In example A, there is very little variation *within* each sex, so that variation *between* sexes is very high at 91.1%. Conversely, example B shows high variation *within* sexes, so that variation *between* sexes is only 17.8%.

The fact that the fold change or the fraction of variation is significantly different from 0 indicates differential expression between the two sexes. Yet these two statistics have different interpretations. The fold change from DE analysis tests a difference in means between two sexes. The fraction of variation explained compares the variation explained by sex to the total variation.

Thus the fraction of variation explained reported by *variancePartition* reflects as different aspect of the data not captured by DE analysis.

Figure 8: **Compare variancePartition and differential expression**



7.5 Modelling error in gene expression measurements

Uncertainty in the measurement of gene expression can be modeled with precision weights and tests of differentially expression using `voom` in *limma* model this uncertainty directly with a heteroskedastic linear regression [6]. *variancePartition* can use these precision weights in a heteroskedastic linear mixed model implemented in *lme4* [1]. These precision weights are used seamlessly by calling `fitVarPartModel` or `fitExtractVarPartModel` on the output of `voom`. Otherwise the user can specify the weights with the `weightsMatrix` parameter.

8 Frequently asked questions

Note that many warnings and errors can be overridden by specifying `suppressWarnings=TRUE` for `dream()` and `showWarnings=FALSE` for `fitExtractVarPartModel()` and `fitVarPartModel()`.

Interpreting warnings and errors from `fitVarPartModel` and `fitExtractVarPartModel`:

8.1 Warnings

- `No Intercept term was specified in the formula:`
  `The results will not behave as expected and may be very wrong!!`

An intercept (i.e. mean term) must be specified order for the results to be statistically valid. Otherwise, the variance percentages will be *very* overestimated.

- `Categorical variables modeled as fixed effect:`
  `The results will not behave as expected and may be very wrong!!`

If a linear mixed model is used, all categorical variables must be modeled as a random effect. Alternatively, a fixed effect model can be used by modeling all variables as fixed.

- `Cannot have more than one varying coefficient term:`
  `The results will not behave as expected and may be very wrong!!`

Only one varying coefficient term can be specified. For example, the formula `~(Tissue+0|Individual) + (Batch+0|Individual)` contains two varying coefficient terms and the results from this analysis are not easily interpretable. Only a formula with one term like `(Tissue+0|Individual)` is allowed.

- `executing %dopar% sequentially: no parallel backend registered`

These functions are optimized to run in parallel using doParallel/doMC. This warning indicates that a parallelization was not enabled. This is not a problem, but analysis will take more time.

8.2 Errors

- `Colinear score > .99: Covariates in the formula are so strongly correlated that the parameter estimates from this model are not meaningful. Dropping one or more of the covariates will fix this problem`
- `Error in asMethod(object) : not a positive definite matrix`
- `In vcov.merMod(fit) : Computed variance-covariance matrix problem:`
  `not a positive definite matrix; returning NA matrix`
- `fixed-effect model matrix is rank deficient so dropping 26 columns / coefficients`

Including variables that are highly correlated can produce misleading results (see Section **??**). In this case, parameter estimates from this model are not meaningful. Dropping one or more of the covariates will fix this problem.

- `Error in checkNlevels(reTrms$flist, n = n, control) : number of levels of each grouping factor must be < number of observations`

This arises when using a varying coefficient model that examines the effect of one variable inside subsets of the data defined by another: `~(A+0|B)`. See Section 4.3. There must be enough observations of each level of the variable B with each level of variable A. Consider an example with samples from multiple tissues from a set of individual where we are interested in the variation across individuals within each tissue using the formula: `~ (Tissue+0|Individual)`. This analysis will only work if there are multiple samples from the same individual in at least one tissue. If all tissues only have one sample per individual, the analysis will fail and `variancePartition` will give this error.

- `Problem with varying coefficient model in formula: should have form (A+0|B)`

When analyzing the variation of one variable inside another (Section 4.3), the formula most be specified as `(Tissue+0|Individual)`. This error occurs when the formula contains `(Tissue|Individual)` instead.

- `fatal error in wrapper code`

- `Error in mcfork() : unable to fork, possible reason: Cannot al locate memory`
- `Error: cannot allocate buffer`

This error occurs when `fitVarPartModel` uses too many threads and takes up too much memory. The easiest solution is to use `fitExtractVarPartModel` instead. Occasionally there is an issue in the parallel backend that is out of my control. Using fewer threads or restarting *R* will solve the problem.

8.2.1 Errors: Problems removing samples with NA/NaN/Inf values

variancePartition fits a regression model for each gene and drops samples that have NA/NaN/Inf values in each model fit. This is generally seamless but can cause an issue when a variable specified in the formula no longer varies within the subset of samples that are retained. Consider an example with variables for sex and age where age is NA for all males samples. Dropping samples with invalid values for variables included in the formula will retain only female samples. This will cause *variancePartition* to throw an error because there is now no variation in sex in the retained subset of the data. This can be resolved by removing either age or sex from the formula.

This situtation is indicated by the following errors

- `Error: grouping factors must have > 1 sampled level`
- `Error: Invalid grouping factor specification, Individual`
- `Error in ‘contrasts<-‘(‘*tmp*‘, value = contr.funs[1 + isOF[nn]]):`

  `contrasts can be applied only to factors with 2 or more levels`
- `Error in checkNlevels(reTrms$flist, n = n, control):`
  `grouping factors must have > 1 sampled level`

Session Info

- R version 3.6.1 (2019-07-05), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`

- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.44.0, BiocGenerics 0.30.0, BiocParallel 1.18.1, DESeq2 1.24.0, DelayedArray 0.10.0, GenomeInfoDb 1.20.0, GenomicRanges 1.36.1, IRanges 2.18.3, Matrix 1.2-17, S4Vectors 0.22.1, SummarizedExperiment 1.14.1, ballgown 2.16.0, dendextend 1.12.0, edgeR 3.26.8, foreach 1.4.7, ggplot2 3.2.1, knitr 1.25, limma 3.40.6, lme4 1.1-21, matrixStats 0.55.0, pander 0.6.3, r2glmm 0.1.2, scales 1.0.0, variancePartition 1.14.1
- Loaded via a namespace (and not attached): AnnotationDbi 1.46.1, BiocManager 1.30.4, BiocStyle 2.12.0, Biostrings 2.52.0, DBI 1.0.0, Formula 1.2-3, GenomeInfoDbData 1.2.1, GenomicAlignments 1.20.1, Hmisc 4.2-0, KernSmooth 2.23-15, MASS 7.3-51.4, R6 2.4.0, RColorBrewer 1.1-2, RCurl 1.95-4.12, RSQLite 2.1.2, Rcpp 1.0.2, Rsamtools 2.0.1, XML 3.98-1.20, XVector 0.24.0, acepack 1.4.1, annotate 1.62.0, assertthat 0.2.1, backports 1.1.4, base64enc 0.1-3, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.2.0, boot 1.3-23, caTools 1.17.1.2, checkmate 1.9.4, cluster 2.1.0, codetools 0.2-16, colorRamps 2.3, colorspace 1.4-1, compiler 3.6.1, crayon 1.3.4, data.table 1.12.2, digest 0.6.21, doParallel 1.0.15, dplyr 0.8.3, evaluate 0.14, foreign 0.8-72, gdata 2.18.0, genefilter 1.66.0, geneplotter 1.62.0, glue 1.3.1, gplots 3.0.1.1, grid 3.6.1, gridExtra 2.3, gtable 0.3.0, gtools 3.8.1, highr 0.8, hms 0.5.1, htmlTable 1.13.2, htmltools 0.3.6, htmlwidgets 1.3, iterators 1.0.12, labeling 0.3, lattice 0.20-38, latticeExtra 0.6-28, lazyeval 0.2.2, lmerTest 3.1-0, locfit 1.5-9.1, magrittr 1.5, memoise 1.1.0, mgcv 1.8-29, minqa 1.2.4, munsell 0.5.0, nlme 3.1-141, nloptr 1.2.1, nnet 7.3-12, numDeriv 2016.8-1.1, pbkrtest 0.4-7, pillar 1.4.2, pkgconfig 2.0.3, plyr 1.8.4, prettyunits 1.0.2, progress 1.2.2, purrr 0.3.2, reshape2 1.4.3, rlang 0.4.0, rmarkdown 1.16, rpart 4.1-15, rstudioapi 0.10, rtracklayer 1.44.4, snow 0.4-3, splines 3.6.1, statmod 1.4.32, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, sva 3.32.1, tibble 2.1.3, tidyselect 0.2.5, tools 3.6.1, vctrs 0.2.0, viridis 0.5.1, viridisLite 0.3.0, withr 2.1.2, xfun 0.10, xtable 1.8-4, yaml 2.2.0, zeallot 0.1.0, zlibbioc 1.30.0

References

[1] D. Bates, M. Machler, B. Bolker, and S. Walker. Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, 67(1), 2015. doi:10.18637/jss.v067.i01.

[2] J. C. Pinheiro and D. M. Bates. *Mixed-Effects Models in S and S-Plus*. Springer, New York, 2000.

[3] A. Galecki and T. Burzykowski. *Linear Mixed Effects Modeling using R*. Springer, 2013.

[4] Yang Liao, Gordon K. Smyth, and Wei Shi. FeatureCounts: An efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930, 2014. doi:10.1093/bioinformatics/btt656.

[5] Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. HTSeq-A Python framework to work with high-throughput sequencing data. *Bioinformatics*, 31(2):166–169, 2015. doi:10.1093/bioinformatics/btu638.

[6] C. W. Law, Y. Chen, W. Shi, and G. K. Smyth. Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology*, 15(2):R29, 2014. doi:10.1186/gb-2014-15-2-r29.

[7] Michael I. Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):550, 2014. doi:10.1186/s13059-014-0550-8.

[8] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34:525–527, 2016. doi:10.1038/nbt.3519.

[9] Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature Biotechnology*, 32(5):462–464, apr 2014. doi:10.1038/nbt.2862.

[10] Rob Patro, Geet Duggal, and Carl Kingsford. Accurate, fast, and model-aware transcript expression quantification with Salmon. *bioarxiv*, 2015. doi:10.1101/021592.

[11] Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, jan 2011. doi:10.1186/1471-2105-12-323.

[12] Cole Trapnell, Brian a Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, may 2010. doi:10.1038/nbt.1621.

[13] Alyssa C Frazee, Geo Pertea, Andrew E Jaffe, Ben Langmead, Steven L Salzberg, and Jeffrey T Leek. Ballgown bridges the gap between transcriptome assembly and expression analysis. *Nature Biotechnology*, 33(3):243–246, 2015. doi:10.1038/nbt.3172.

[14] Charlotte Soneson, Michael I. Love, and Mark D. Robinson. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, 4(0):1521, 2015. doi:10.12688/f1000research.7563.2.

[15] Benilton S Carvalho and Rafael A Irizarry. A framework for oligonucleotide microarray preprocessing. *Bioinformatics*, 26(19):2363–7, 2010. doi:10.1093/bioinformatics/btq431.

[16] M. D. Robinson and A. Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, 11(3):R25, 2010. doi:10.1186/gb-2010-11-3-r25.