DGE.Tools2 Sample Workflow

John Thompson (john.thompson@bms.com)

March 14 2019

Contents

1 Setup

Load packages, set the working dir and output path.

```
rm(list=ls()) #Clear the workspace
invisible(gc()) #garbage collection to maximize available memory
startTime = Sys.time() #used to time the run

library(tidyverse)
library(magrittr)
library(DGEobj)
library(DGE.Tools2)
library(JRTutil)

# change this to your working directory
# my practice is to set the working directory based on a relative path from the git repo root.
# setwd("~/R/lib/pkgsrc/DGE.Tools2")

# set WD to git root
setwd (here::here())
# descend into .Rmd folder for this markdown
setwd("./vignettes")

outputPath <- "./output"
```

2 Initializing or loading a DGEobj

Several chunks are provided to show how to get data from various sources.

Set eval=TRUE in the chunk header for the method you wish to use.

2.1 Load project from Omicsoft S3 bucket

This builds a DGEobj with the mimimal set of raw data. Here raw data is defined as a matrix of counts with associated dataframes to annotate the genes (rowData) and samples (colData) data.

The buildOmicsoftDGEobj function understands the folder structure of the ArrayServer S3 bucket and contructs the path ot the 3 tab-delimited text files retrieved from the ExportedViewsAndTables folder for the specified project.

The Omicsoft data is stored in BMS S3 bucket bmsrd-ngs-arrayserver. You need to mount the S3 bucket to the local file system. On Mac/Linux, you can use the opensource tool s3fs. On the PC you can use Cloudberry Drive (commercial software; $40).

```
OmicsoftProjectID = "BDL_Rat_LiverSlice_P-20170808-0001_03Dec2017"
mountPoint <- "y:"

dgeObj <- JRTutil::buildOmicsoftDGEobj(projectName = OmicsoftProjectID, level="gene", mountPoint=mountP

knitr::kable(inventory(dgeObj))
```

ItemName	ItemType	BaseType	Parent	Class	DateCreated
counts_orig	counts_orig	meta		matrix	2019-03-14 08:52:25
counts	counts	assay	counts_orig	matrix	2019-03-14 08:52:25
design_orig	design_orig	meta		data.frame	2019-03-14 08:52:25
design	design	col	design_orig	data.frame	2019-03-14 08:52:25
geneData_orig	geneData_orig	meta		data.frame	2019-03-14 08:52:25
geneData	geneData	row	geneData_orig	data.frame	2019-03-14 08:52:25
granges_orig	granges_orig	meta	geneData_orig	GRanges	2019-03-14 08:52:25
granges	granges	row	geneData	GRanges	2019-03-14 08:52:25

2.2 Load a DGEobj from the DGEobj_library folder on Stash

The DGEobj RDS files for all projects loaded into GECO are stored in /stash/data/nonclin/DGEobj_library. The getRDSobjFromStash uses this path as the default to retrieve the named RDS file. You can specify a different folderPath to retrieve other .RDS file data from anywhere else in stash.

Note that the stash root path is different on Mac/Linux and Windows. The getRDSobjFromStash function checks the OS and sets the path accordingly and thus provides an OS-indepedent way to load RDS files.

```
RDSfile <- "BDL_Rat_LiverSlice_P-20170808-0001_03Dec2017.RDS"

dgeObj_FromStash <- JRTutil::getRDSobjFromStash(RDSfile)

knitr::kable(inventory(dgeObj))
```

The data from the DGEobj_library folder has already been analyzed. We'll use function resetDGEobj to restore it to it's original state so we can go through the whole workflow.

```
dgeObj_FromStash <- reset(dgeObj)

knitr::kable(inventory(dgeObj))
```

3 Load project from Xpress

The Xpress2DGEO function used here is a wrapper around rXpress that retrieves all data for a specified project using an Xpress ID. The Xpress ID is found at the end of the Xpress URL for the project of interest. For this project:

http://xpress.pri.bms.com/CGI/project_summary.cgi?project=20261

The Xpress ID is the 20261.

Using just the Xpress project ID returns the available levels for this project.
Choose one and run the command again with the levels argument.

Recently, the rXpress package was updated to take a version id for the design information and an Xpress project can have multiple design tables. This change has not been implemented in Xpress2DGEO yet. So expect a warning from Xpress2DGEO and Xpress2DGEO will still return the design table tagged as DEFAULT.

```
library(Xpress2R)

# check which levels are available
dgeObj_FromXpress <- Xpress2DGEO(20261)

# plug the desired level into this call
dgeObj_FromXpress <- Xpress2DGEO(20261, level = "rn6ERCC-ensembl82-genes")
```

```
##
## Available DOE versions for Project  20261  :
##   DEFAULT                      DOE_VERSION                DATE
## 1    TRUE 3fd8ab0f3b72ace91951f69cd68be7e0 2019-02-27 11:08:01
## NOTE: non-compliant call made for 'getXpressData'
##   (Legacy support for un-specified annotation versions -  Please set versionSampleAnnotation= to one
## Available DOE versions for Project  20261  :
##   DEFAULT                      DOE_VERSION                DATE
## 1    TRUE 3fd8ab0f3b72ace91951f69cd68be7e0 2019-02-27 11:08:01
## NOTE: non-compliant call made for 'getXpressData'
##   (Legacy support for un-specified annotation versions -  Please set versionSampleAnnotation= to one
```

4 Custom Cleanup

Here's a good spot to look at your annotation and perform any custom curation needed before you start your analysis.

Careful here. If you use tidyverse functions to manipulate these tables you must take care to preserve the rownames in the design and gene annotation dataframes.

4.1 Gene annotation cleanup

There's one column we'll rename in the Xpress design table to be consistent with the column name in the Omicsoft data.

```
dgeObj_FromXpress$geneData %<>% rownames_to_column() %>%
  mutate(Source = Biotype) %>%
  column_to_rownames()

dgeObj_FromXpress$geneData_orig %<>% rownames_to_column() %>%
  mutate(Source = Biotype) %>%
  column_to_rownames()

# Reality check to make sure we didn't screw up the rownames
all(rownames(dgeObj$geneData) == rownames(dgeObj))
```

```
## [1] TRUE
```

4.2 Design table cleanup

In this case, I just don't like one of the column names. The "treatment" column in the design table is more acurately called "pretreatment" as it defines which animals were pretreated with a disease-inducing surgery and were later treated with potentially efficacious compounds.

```
# saveRDS(dgeObj_FromXpress, file.path(outputPath, "dgeObj_FromXpress.RDS"))
# saveRDS(dgeObj, file.path(outputPath, "dgeObj.RDS"))
#
# dgeObj_FromXpress <- readRDS (file.path(outputPath, "dgeObj_FromXpress.RDS"))
# dgeObj <- readRDS (file.path(outputPath, "dgeObj.RDS"))

all(rownames(dgeObj$design) == colnames(dgeObj))
```

```
## [1] TRUE
```

```
# Let's fix the Omicsoft DGEobj first
dgeObj$design %<>% rownames_to_column() %>%
  mutate(pretreatment = treatment) %>%
  column_to_rownames()

all(rownames(dgeObj$design) == colnames(dgeObj))
```

```
## [1] TRUE
```

```
#let's do the same to the original design table
dgeObj$design_orig %<>% rownames_to_column() %>%
  mutate(pretreatment = treatment) %>%
  column_to_rownames()

# ReplicateGroup is the adopt name for this field; correct the spelling/case
dgeObj$design %<>% rownames_to_column() %>%
```

```
  mutate(ReplicateGroup = replicate_group) %>%
  column_to_rownames()

dgeObj$design_orig %<>% rownames_to_column() %>%
  mutate(ReplicateGroup = replicate_group) %>%
  column_to_rownames()

# let's do the same for the DGEobj from Xpress
# create more intuitive column names for the main experiment factors in the design table
dgeObj_FromXpress$design %<>% rownames_to_column() %>%
  mutate(pretreatment = BMS_Compound_Dose_1) %>%
  column_to_rownames()

dgeObj_FromXpress$design_orig %<>% rownames_to_column() %>%
  mutate(pretreatment = BMS_Compound_Dose_1) %>%
  column_to_rownames()

# creating the ReplicateGroup column from existing compound column
# (Generic_Compound_Dose_1) is rather complicated. So we're going to pull the
# compound and ReplicateGroup columns from the Omicsoft dataset using the common
# EP.Well=EP_Well columns as the key for a left_join.

# get the columns we want to add
OScols <- select(dgeObj$design, EP_Well=EP.Well, compound, ReplicateGroup)
dgeObj_FromXpress$design %<>% rownames_to_column() %>%
  left_join(OScols, by="EP_Well") %>%
  column_to_rownames()

# Reality check to make sure we didn't screw up the rownames
all(rownames(dgeObj$design) == colnames(dgeObj))
```

```
## [1] TRUE
```

5 Low Intensity Filtering

Typically, genes with near zero counts are removed before further analysis. They contain no useful information, increase the multiple test burden, and could (under some conditions) compromise the normalization and violate asumptions implicit in linear modeling.

Three methods for low intensity filtering are supplied; min counts, zFPKM and FPK. The lowIntFilter function will use any combination of these methods. The sampleFraction argument defines the proportion of samples that must meet all specified criteria to keep a gene.

Minmum counts = 10 is commonly used to filter low intensity genes. But mincounts is biased against short genes. So zFPKM and/or FPK provide intensity measures that are not length biased.

FPKM density curves of gene level data show a bimodal distribution. Hart et al. used ChIP-Seq data from ENCODE that define open and closed chromation configurations to orthogonally define which genes were expressed. This work showed that the upper FPKM peak corresponds closely to the open (active) chromatin conformation. They developed a method to fit a curve to just the upper peak and then use Z-score analysis to define a low expression cutoff. Ron Ammar implemented this method in the zFPKM package available in CRAN. zFPKM >= -3 is the recommended threshold to use as a cutoff for "detected" genes.

The method to fit the upper intensity peak in zFPKM sometimes fails to properly identify the upper peak. Thus, to rely on zFPKM, it is important that you inspect the FPKM density curve and visually verify that the algorithm properly identified and fit the upper peak in the density curve. The red curve is a fit to the upper peak of the bimodal FPKM density curve (blue).

```
counts <- DGEobj::getItem(dgeObj, "counts")
geneAnno <- DGEobj::getItem(dgeObj, "geneData")

fpkm <- DGE.Tools2::convertCounts(counts, unit="FPKM", log=FALSE, geneLength=geneAnno$ExonLength) %>%
    as.data.frame

library(zFPKM)
# Plot first to visually confirm proper fit to the upper peak
zFPKM_plot <- zFPKM::zFPKMPlot(fpkm)
ggsave(zFPKM_plot, "zFPKM_plot.PNG")
```

Fragments Per Killobase (FPK) is the third filtering criterion. For an average size 2kb mRNA FPK = 5 is equivalent to counts = 10, however, FPK is not length biased. Another useful property of FPK is that FPK can be calculated for integenic DNA and thus provides an empirical background level of stochastic transcription. This estimate is surely conservative in that there is very likely some real transcription going on in intergenic regions. Typically, the intergenic FPK level is below 5 so FPK >= 5 is a good criterion.

I typically use the combination of FPK + mincount filters to define detected genes. You then have to decide how to integrate this information across experiment groups. You can get more sophisticated and do this on a group-wise basis so you can include genes that were expressed in at least one of your treatment groups. I leave that as an exercise for the reader and note that groupwise filtering introduces a bias that affects your pvalue calibration. To avoid such bias, we simply require XX% of samples to pass the intensity threshold and you can modify the percentage to adjust the stringency.

If you use length adjusted measures (zFPKM or FPK) you must also supply the geneLength argument.

Dimensions before filtering:
32883, 48

```
fracThreshold <- 0.5

# low expression filter
dgeObj <- lowIntFilter(dgeObj,
                       fpkThreshold = 5,
```

```
                        countThreshold = 10,
                        sampleFraction = fracThreshold,
                        genelength = dgeObj$geneData$ExonLength)
```

Dimensions after filtering: 13735, 48

6 Filter for Protein Coding Genes

Often our analysis is focused on protein coding genes. Here we use the biotype column of gene annotation to keep only protein coding genes.

Here we'll use square bracket subsetting to select protein coding genes.

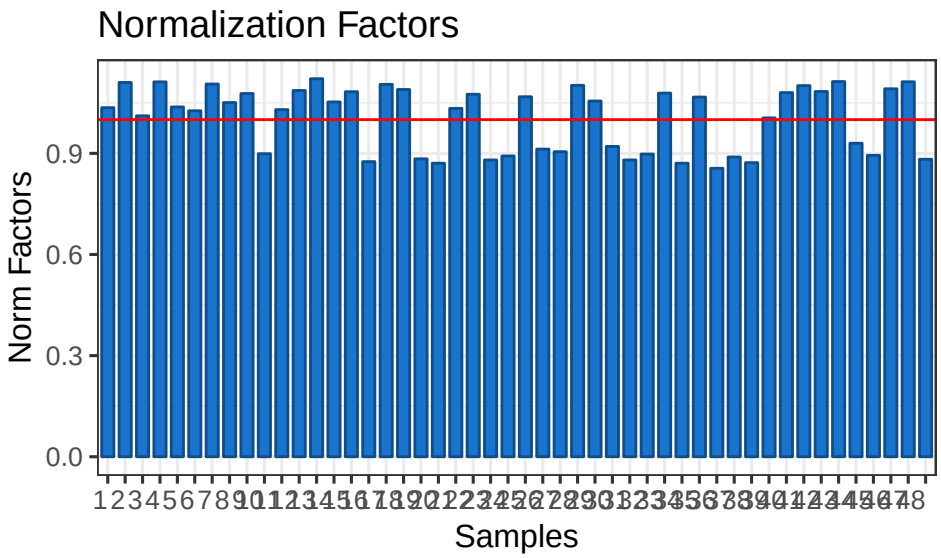
```
idx <- dgeObj$geneData$Source == "protein_coding"
dgeObj <- dgeObj[idx,]
```

Dimensions after filtering for protein_coding:
13081, 48

7 EdgeR Normalization

This step simply applies edgeR::calcNormFactors to effect TMM normalization. This results in a DGEList object being added to the DGEobj. Note that the counts matrix within the DGEList object is NOT normalized counts. Rather a separate item in the DGEList contains the norm factors and you should use the edgeR cpm function to extract normalized counts from the DGEList. More conveniently, you can use the DGE.Tools2::convertCounts function to produce normalized counts.

```
dgeObj <- runEdgeRNorm(dgeObj, plotFile = file.path(outputPath, "TMM_NormFactors.PNG"))
```



ItemName	ItemType	BaseType	Parent	Class	DateCreated
counts_orig	counts_orig	meta		matrix	2019-03-14 08:52:25
counts	counts	assay	counts_orig	matrix	2019-03-14 08:52:25
design_orig	design_orig	meta		data.frame	2019-03-14 08:52:25
design	design	col	design_orig	data.frame	2019-03-14 08:52:25
geneData_orig	geneData_orig	meta		data.frame	2019-03-14 08:52:25
geneData	geneData	row	geneData_orig	data.frame	2019-03-14 08:52:25
granges_orig	granges_orig	meta	geneData_orig	GRanges	2019-03-14 08:52:25
granges	granges	row	geneData	GRanges	2019-03-14 08:52:25
DGEList	DGEList	assay	counts	DGEList	2019-03-14 08:52:48

8 Define the model formula

Provide a formula and construct the design matrix.

Defining the best possible formula is beyond the scope of this tutorial. We recommend use of the variancePartition package to evaluate the proportion of variance carried by each experimental factors and determine which factors are colinear. Then make an informed decision of which terms to include in the model. There are very good vignettes associated with the variancePartition package.

Rodent experiments typically are genetically pure, on a controlled diet and thus the formula can often be quite simple. In this experiment we have sham group and a bile duct ligation group (BDL). Then there are three compounds We have two key design terms, treatment (BDL; more accurately, this is the disease-inducing surgical pre-treatment) and compound (one or three compounds applied to BDL animals).

We defined a new column, ReplicateGroup, that concatenates the treatment and compound columns.

ReplicateGroup
BDL_Sora
BDL
BDL_BIBF-1120
BDL_EXT-1024
Sham

There are no other known covariants to examine so the formula choice is simply:

~ 0 + ReplicateGroup

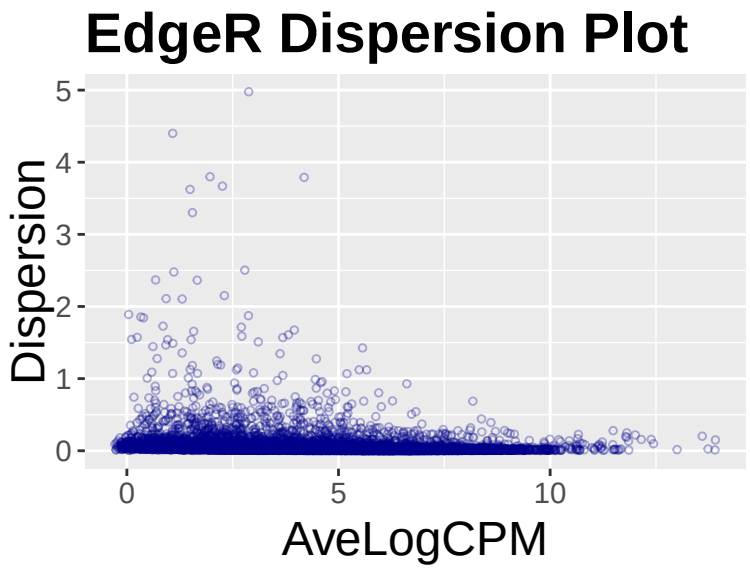
This is just one way to paramerterize this model. "~ 1 + treatment + compound" is equally valid but the "~ 0 ReplicateGroup" formula makes it easier to specify contrasts with different baselines and the contrast specified by ReplicateGroup are more intuitive to our biologist collaborators.

```
#define a formula and construct a design matrix
design <- getItem(dgeObj, "design")

# Next step is probably unecessary unless you have a numeric column that
# you want to treat as a factor.
design$ReplicateGroup %<>% as.factor

# The next step is only necessary  if you want to use a ~1 formula, then
# the baseline must be the first factor.
design$ReplicateGroup %<>% relevel("Sham")

formula <- '~ 0 + ReplicateGroup'

#build the designMatrix
designMatrix <- model.matrix (as.formula(formula), design)

#give this design a name
designMatrixName <- "PreTreat_Compound"

# Make sure the column names in the design matrix are legal
# convert spaces and other disallowed chars to underscores or dots
colnames(designMatrix) <- make.names(colnames(designMatrix))

#capture the formula as an attribute of the design matrix
designMatrix <- setAttributes(designMatrix, list(formula=formula))
```

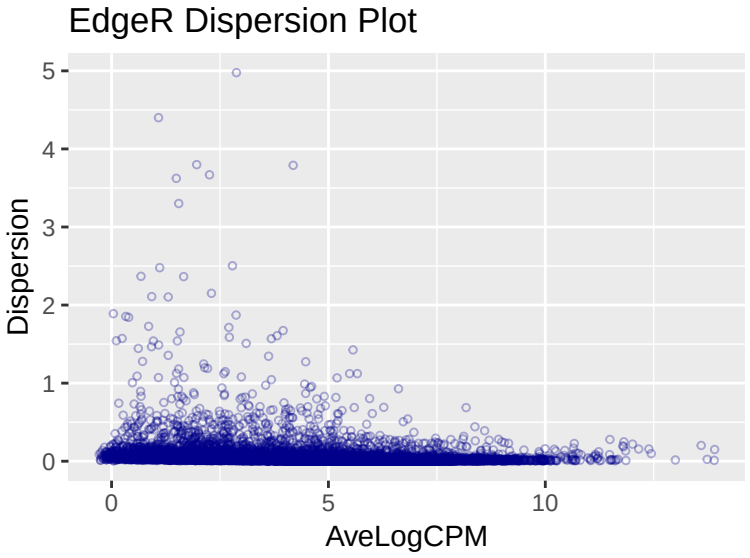
```
#manually add the designMatrix to the DGEobj
dgeObj <- addItem(dgeObj, item=designMatrix,
                  itemName=designMatrixName,
                  itemType="designMatrix",
                  parent="design",
                  overwrite=TRUE)
```

9 QC: Dispersion Plot

```
dispPlot <- plotDisp(getItem(dgeObj, "DGEList"), designMatrix)

# print the plot to the console (and change the base font size)
dispPlot + baseTheme(14)
```




```
# save the plot with ggsave
ggsave(file.path(outputPath, "dispersion.png"), dispPlot + baseTheme(18))

# print to console and save the plot to the png file.
# Larger font (user settable), more suitable for ppt slides, is used for the png.
DGE.Tools2::printAndSave(dispPlot, file.path(outputPath, "dispersion.png"))
```



10 Check for Surrogate Variables (unaccounted for variation)

SVA looks for hidden structure in the data using PCA-like methods. It defines surrogate variables that can be added to your model to account for systematic trends that don't map to known experiment factors. This can improve you power to detect changes due to factors of interest.

```
# Let's save a snapshot of the DGEobj at this point
saveRDS (dgeObj, file.path(outputPath, "dgeobj.RDS"))
```

```
#sva test
dgeObj_sva <- runSVA(dgeObj, designMatrixName="PreTreat_Compound")
```

```
## Number of significant surrogate variables is:  9
## Iteration (out of 5 ):1  2  3  4  5
```

SVA found 9 surrogate variables in this dataset.

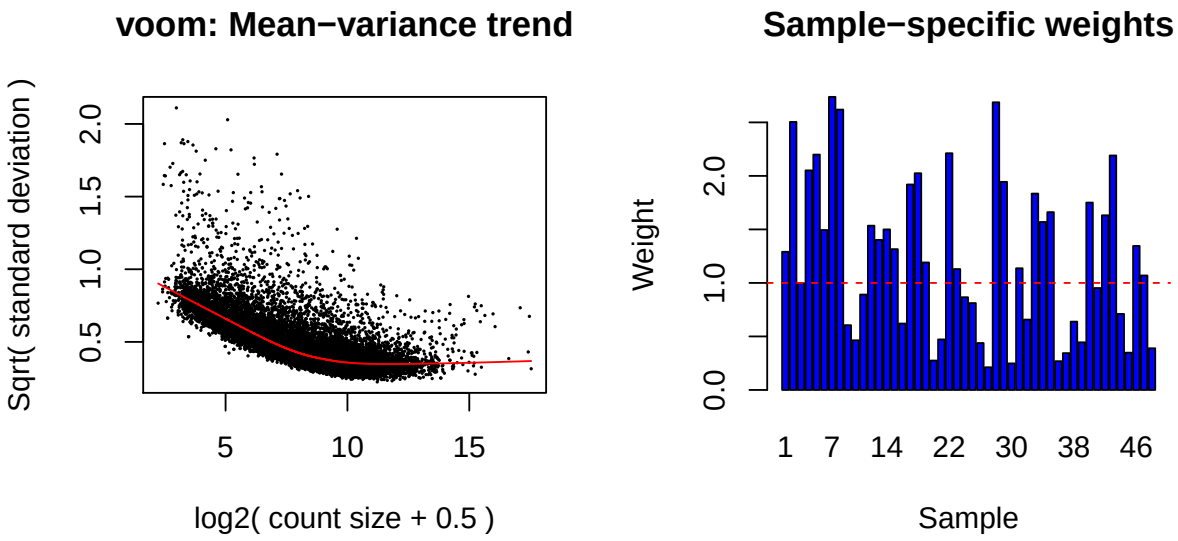
If SVA analysis finds 1 or more variables, it adds a column for each SV to both the design matrix and design table. You can include the SV columns in your formula.

Especially if there are more than a few SV found, we suggest you use the variantPartition package to evaluate the variance fraction associated with each SV and consider using just the top few SVs in your formula. To accomplish this, you need to manually edit the design matrix and remove SV columns you don't want to use.

11 Run Voom and fit the model (lmfit)

For now we'll ignore the sva results (dgeObj_sva) and continue analysis with the dgeObj data structure.

```
dgeObj <- runVoom(dgeObj, designMatrixName)
```




```
# note: qualityWeights = TRUE,  runEBayes = TRUE and robust = TRUE are the defaults.
```

Expect an inconsequential warning(s) at the point. Not a problem in this case.

```
Warning messages:
1: In at[itemName] <- attribs[[i]] :
  number of items to replace is not a multiple of replacement length
```

The Mean-variance trend nicely shows the heterscedasticity typical of RNA-Seq data (variance dependent on intensity).

If you notice a downward hook on the left end of this distribution, it means you have not applied a sufficiently stringent low intensity filter.

11.1 Variation: Apply limma duplicateCorrelation for Repeated Measures

If you have repeated measures from the same subject (e.g. time points from the same subject and similar senarios), you should use the limma::duplicateCorrelation method to account for within subject correlation.

To invoke the duplicateCorrelation method you simply supply a blocking vector that identifies samples from the same subject (typically a subjectID).

```
# restore the pre-voom DGEobj
dgeobj <- readRDS (file.path(outputPath, "prevoom_dgeobj.RDS"))

# This dataset does not have duplicate measures but here's how to invoke it.
# Typically you would use a sample identifier for this.
block <- dgeObj$design$Sample.Name
# but since we don't have duplicate here let's just make up a blocking pattern
block <- rep(c("a", "b", "c", "d"), times=ncol(dgeObj)/4)
```

```
dgeObj_dupcor <- runVoom(dgeObj, designMatrixName,
              dupcorBlock = block)
```

Since we made up the blocking var, the correlation reported is understandably low.

11.2 Variation: var.design for blocking quality weights

By default voomWithQualityWeights treats each sample separately. voomWithQualityWeights also produces a plot of the sample quality weights. If you notice that samples within replicate groups have similar quality weights and distinct from other quality weights, then you can supply a design matrix to the var.design argument. When var.design is used, quality weights are calculated for each group rather than each sample and this will be evident in the quality weights plot.

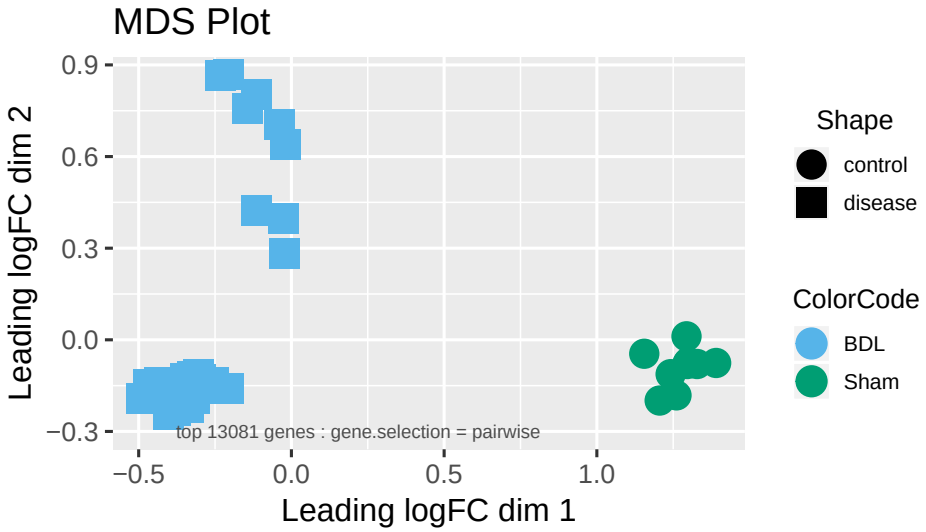
```
# The vd design matrix is used to block the quality weight determination
# Typically use a design column to define the blocking
vd <- model.matrix(as.formula("~ Disease.Status"), design)

dgeObj_vd <- runVoom(dgeObj, designMatrixName,
              qualityWeights = TRUE,
              var.design=vd)
```

```
# Let's save a snapshot of the DGEobj at this point
saveRDS (dgeObj, file.path(outputPath, "dgeobj.RDS"))
```

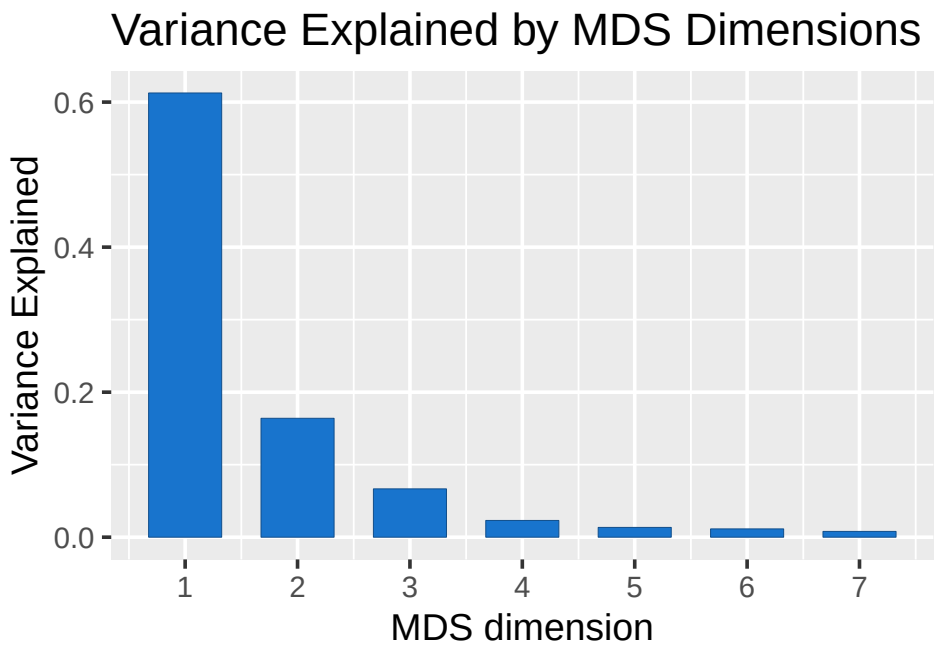
12 Data Exploration: MDS plot

Principal Component Analysis is a subset of Multi-Dimensional Scaling that uses correlation as a distance metric. The limma mdsPlot function yields similar results to PCA but uses a intensity based distance metric. Thus, if you perform MDS on log2cpm data, the units of the plot are in log2cpm units. So the scale of an MDS plot is more interpretable that correlation-based methods.

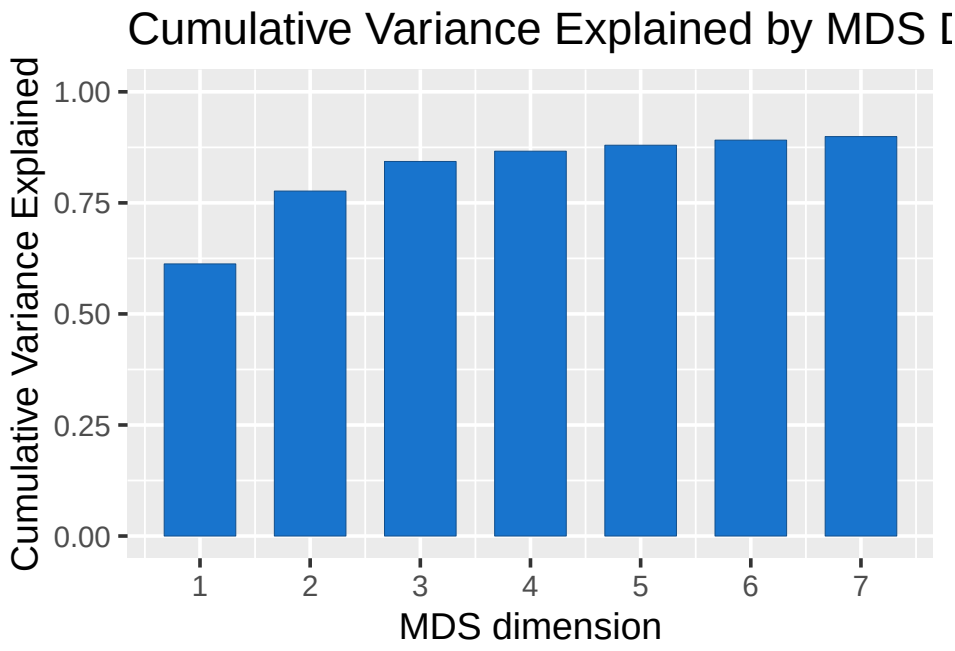
```
#use color and shape with labels and labelSize
# See ?ggplotMDS for more options
m <- DGE.Tools2::ggplotMDS(dgeObj, colorBy = design$treatment,
               shapeBy = design$Disease.Status, symSize =5,
               labels=design$VendorBarcode,
               labelSize=3,
               dim.plot=c(1,2))
# ggplotMDS returns a list of 2.  Item 1 is the gggplot; Item 2 is the data object
# returned by limma::plotMDS

printAndSave(m[[1]], file.path(outputPath, "MDSplot.PNG"))
```




```
# Now let's plot the amount of variance explained by each component
results <- MDS_var_explained(m[[2]])
# results is a list of 3 items:
#   1) Var Explained bar plot
#   2) Cumulative variance explained
#   3) The dataframe used for the plot

printAndSave(results[[1]], file.path(outputPath, "varExplained.PNG"))
```




```
printAndSave(results[[2]], file.path(outputPath, "cumVarExplained.PNG"))
```



13 Set up and run contrasts

Function runContrasts takes a named list of contrasts. The values in the list should correspond to columns in the design matrix.

Checking the inventory we see that PreTreat_Compound is the itemName of our design matrix:

ItemName	ItemType	BaseType	Parent	Class	DateCreated
counts_orig	counts_orig	meta		matrix	2019-03-14 08:52:25
counts	counts	assay	counts_orig	matrix	2019-03-14 08:52:25
design_orig	design_orig	meta		data.frame	2019-03-14 08:52:25
design	design	col	design_orig	data.frame	2019-03-14 08:52:25
geneData_orig	geneData_orig	meta		data.frame	2019-03-14 08:52:25
geneData	geneData	row	geneData_orig	data.frame	2019-03-14 08:52:25
granges_orig	granges_orig	meta	geneData_orig	GRanges	2019-03-14 08:52:25
granges	granges	row	geneData	GRanges	2019-03-14 08:52:25
DGEList	DGEList	assay	counts	DGEList	2019-03-14 08:52:48
PreTreat_Compound	designMatrix	col	design	matrix	2019-03-14 08:52:52
PreTreat_Compound_Elist	Elist	assay	DGEList	EList	2019-03-14 08:53:44
PreTreat_Compound_fit	fit	row	PreTreat_Compound_Elist	MArrayLM	2019-03-14 08:53:44

Here's the column names we can use to build out contrasts:

ReplicateGroups
ReplicateGroupSham
ReplicateGroupBDL
ReplicateGroupBDL_BIBF.1120
ReplicateGroupBDL_EXT.1024
ReplicateGroupBDL_Sora

In this experiment we have a disease state signature (BDL vs Normal). We'll also specify contrasts of drug treated diseased (BDL) animals vs the untreated BDL animals baseline (disease reversal signatures).

The names for the contrasts should be as short as possible but also human recognizable as these names are the default labels when plotting contrast data.

```
# runContrast testing
contrastList  <- list(BDL_vs_Normal = "ReplicateGroupBDL - ReplicateGroupSham",
                      BIBF1120_vs_BDL = "ReplicateGroupBDL_BIBF.1120 - ReplicateGroupBDL",
                      EXT1024_vs_BDL = "ReplicateGroupBDL_EXT.1024  - ReplicateGroupBDL",
                      Sors_vs_BDL = "ReplicateGroupBDL_Sora  - ReplicateGroupBDL"
)

dgeObj <- runContrasts(dgeObj,
                       designMatrixName="PreTreat_Compound",
                       contrastList=contrastList,
                       Qvalue=TRUE,
                       IHW = TRUE)
```

ItemName	ItemType	BaseType	Parent	Class	DateCreated
counts_orig	counts_orig	meta		matrix	2019-03-14 08:52:25
counts	counts	assay	counts_orig	matrix	2019-03-14 08:52:25
design_orig	design_orig	meta		data.frame	2019-03-14 08:52:25
design	design	col	design_orig	data.frame	2019-03-14 08:52:25
geneData_orig	geneData_orig	meta		data.frame	2019-03-14 08:52:25
geneData	geneData	row	geneData_orig	data.frame	2019-03-14 08:52:25
granges_orig	granges_orig	meta	geneData_orig	GRanges	2019-03-14 08:52:25
granges	granges	row	geneData	GRanges	2019-03-14 08:52:25
DGEList	DGEList	assay	counts	DGEList	2019-03-14 08:52:48
PreTreat_Compound	designMatrix	col	design	matrix	2019-03-14 08:52:52
PreTreat_Compound_Elist	Elist	assay	DGEList	EList	2019-03-14 08:53:44
PreTreat_Compound_fit	fit	row	PreTreat_Compound_Elist	MArrayLM	2019-03-14 08:53:44
PreTreat_Compound_fit_cm	contrastMatrix	meta	PreTreat_Compound_fit	matrix	2019-03-14 08:54:16
PreTreat_Compound_fit_cf	contrast_fit	row	PreTreat_Compound_fit	MArrayLM	2019-03-14 08:54:16
BDL_vs_Normal	topTable	row	PreTreat_Compound_fit_cf	data.frame	2019-03-14 08:54:16
BIBF1120_vs_BDL	topTable	row	PreTreat_Compound_fit_cf	data.frame	2019-03-14 08:54:16
EXT1024_vs_BDL	topTable	row	PreTreat_Compound_fit_cf	data.frame	2019-03-14 08:54:16
Sors_vs_BDL	topTable	row	PreTreat_Compound_fit_cf	data.frame	2019-03-14 08:54:16

After runContrasts, the topTable dataframes are present in the DGEobj.

13.1 Alternative FDR scores

topTable provides a BH FDR value (adj.P.Val). You can also add other optional FDR measures to the topTable output using the Qvalue and IHW arguments.

Qvalue = TRUE adds “qvalue” and “qvalue.lfdr” columns to the topTable output.

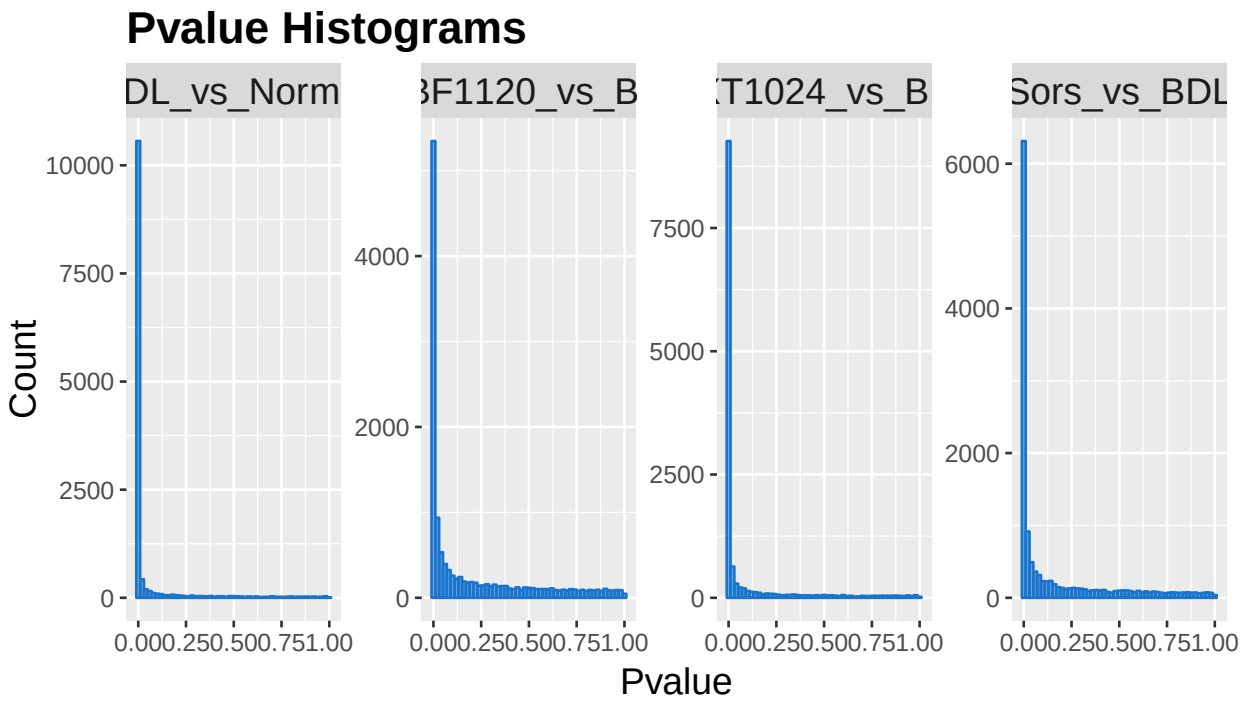
IHW = TRUE adds columns “ihw.adj_pvalue” and “ihw.weight”.

Browse the vignettes for the qvalue and IHW packages for more details on these alternative FDR measures.

14 Inspect pvalue histograms

Pvalue histograms can reveal problems with your model. David Robinson has a good webpage describing how to interpret p-value histograms.

```
# We need to collect a pvalue column from each topTable dataframe
listOfTopTable <- getType(dgeObj, "topTable")
MyPvalMatrix <- DGE.Tools2::extractCol(listOfTopTable, colName="P.Value")
plotPvalHist (MyPvalMatrix)
```



15 Count Significant differential genes

Table 1: No fold change threshold; p < 0.01, FDR thresholds = 0.1

```
# We need a list of the topTable dataframes.
listOfTopTable <- getType(dgeObj, "topTable")
knitr::kable(DGE.Tools2::summarizeSigCounts(listOfTopTable))
```

	P.Value	adj.P.Val	Qvalue	qvalue.lfdr	ihw.adj_pvalue
BDL_vs_Normal	10562	11439	13081	11006	11439
BIBF1120_vs_BDL	5347	6864	8726	5897	6847
EXT1024_vs_BDL	9264	10498	12262	9844	10512
Sors_vs_BDL	6314	7926	10112	7117	7915

Table 2: fold change threshold = 2; p < 0.01, FDR thresholds = 0.1

```
# We need a list of the topTable dataframes.
listOfTopTable <- getType(dgeObj, "topTable")
knitr::kable(DGE.Tools2::summarizeSigCounts(listOfTopTable, fcThreshold = 2))
```

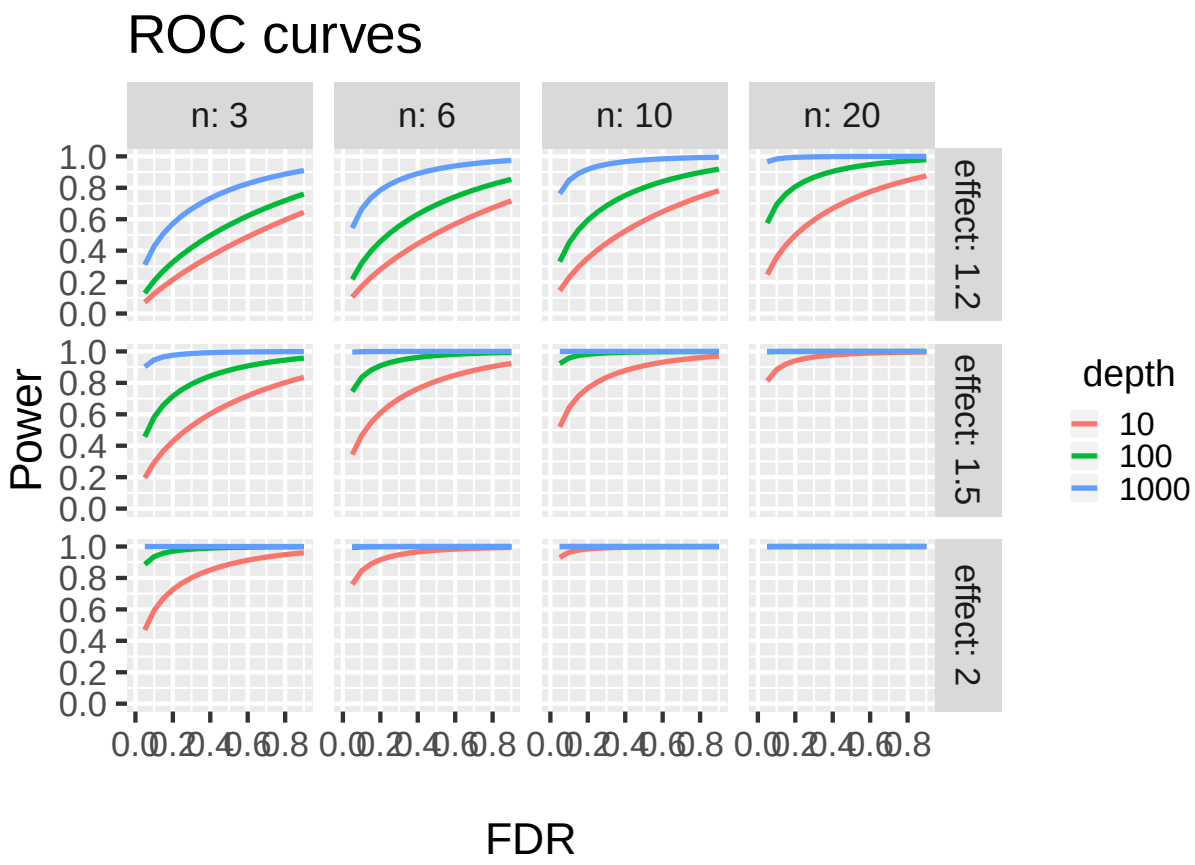
	P.Value	adj.P.Val	Qvalue	qvalue.lfdr	ihw.adj_pvalue
BDL_vs_Normal	4787	4847	4876	4824	4848
BIBF1120_vs_BDL	691	704	716	695	704
EXT1024_vs_BDL	2824	2911	2949	2877	2924
Sors_vs_BDL	693	702	717	700	702

16 Run a Power Analysis

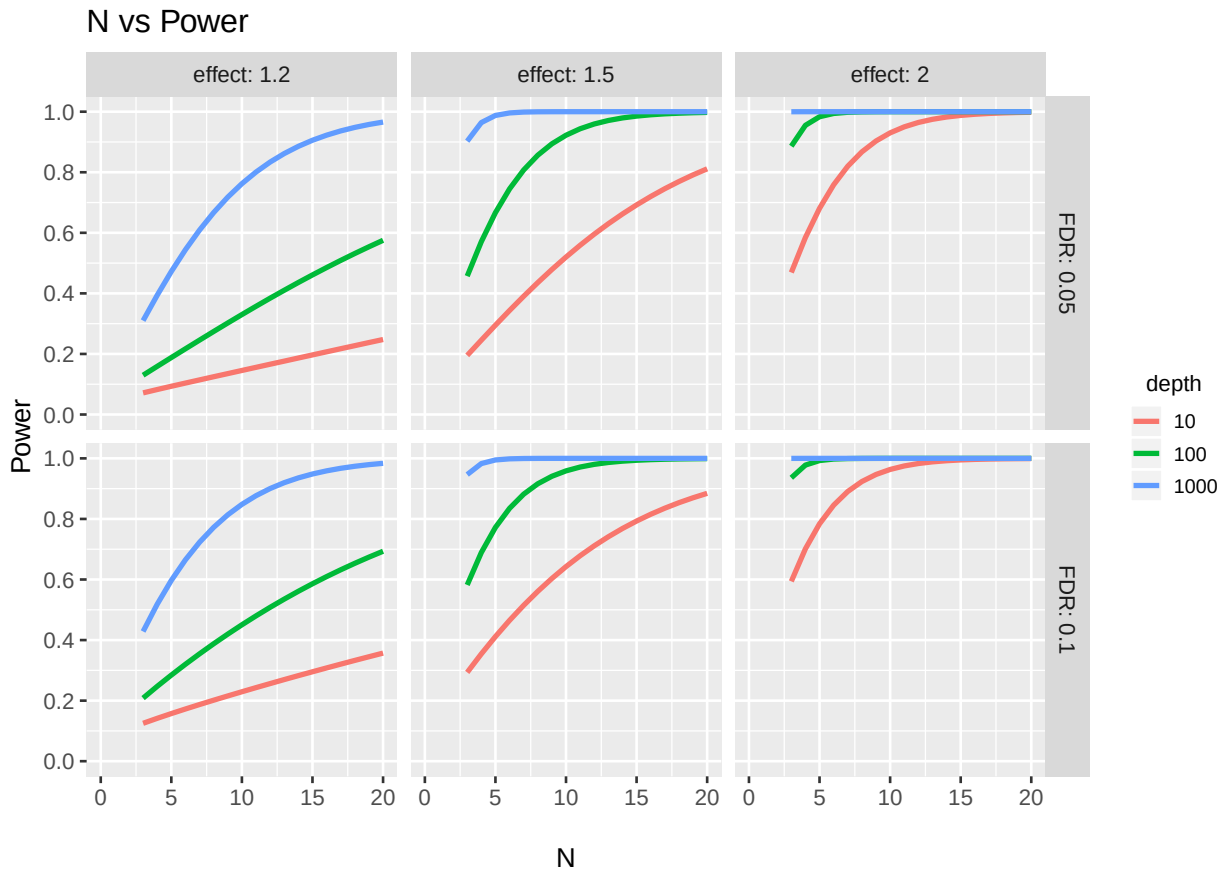
Interpretation of DGE data is improved by an understanding of the proportion of true positives detected as well as the degree of false postives expected. Traditional power analysis is unbiased with regard to intensity. However, we know from experience that a 2X change in a high intensity gene is more reliable that a 2X change in a gene near the detection limit. The rnapower takes the intensity level into consideration in estimating power in RNA-Seq data.

The depth argument in this process refers to raw counts and the default levels of 10, 100, 1000 correspond roughly to detection limit, middle low expression and median expression levels. You'll see that estimated power increases with increasing intensity level.

```
counts <- getItem(dgeObj, "counts")
designMatrix <- getItem(dgeObj, "PreTreat_Compound")

pwr <- DGE.Tools2::runPower(counts, designMatrix)
# result is a list of objects depending on the return argument
# Default objects returned:
#    PowerData: the dataframe generated
#    ROC:  ggplot ROC curve
#    NuP:  plots emphasizing the relationship of N to power

printAndSave(pwr$ROC, file.path(outputPath, "power_ROC.PNG"))
```



```
printAndSave(pwr$NvP, file.path(outputPath, "power_NvP.PNG"))
```



17 Wrapup and other documentation

This completes the DGE calculations.

The **training slides** are available.

See the **DGE.Tools Plot Gallery pdf** for additional data exploration plots.

See browseVignettes("DGEobj") for detailed documentation on the DGEobj datastructure and associated functions.

See browseVignettes("variancePartition") for instructions on evaluating the variance contribution of experiment factors to inform formula selection.

Install the latest stable versions of the DGEobj, DGE.Tools2 and JRTutil packages from **BRAN** using install.packages. See the **BRAN webpage** for instructions in adding the BRAN repository to your repos list.

Install the dev version of these packages from the respective Biogit repos and inspect the commit messages to see what's new.

18 Session Info

Time required to process this report: *2.525571 mins*

R Session Info

```
#Don's envDoc replacement for sessionInfo()
# library(envDocument)
# library(knitr)
# myenv = env_doc("return")
# kable(myenv)
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] DGE.Tools2_0.9.63   zFPKM_1.4.1         annotables_0.1.95
##  [4] Xpress2R_1.1.0      usethis_1.4.0       devtools_2.0.1
##  [7] sva_3.30.1          BiocParallel_1.16.6 genefilter_1.64.0
## [10] mgcv_1.8-26         nlme_3.1-137        JRTutil_0.9.35
## [13] Biobase_2.42.0      BiocGenerics_0.28.0 DGEobj_0.9.33
## [16] magrittr_1.5        forcats_0.4.0       stringr_1.4.0
## [19] dplyr_0.8.0.1       purrr_0.3.1         readr_1.3.1
## [22] tidyr_0.8.3         tibble_2.0.1        ggplot2_3.1.0
## [25] tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.1.4                  CatMisc_1.1.1
##   [3] tidyselect_0.2.5            RSQLite_2.1.1
##   [5] AnnotationDbi_1.44.0        htmlwidgets_1.3
##   [7] grid_3.5.2                  IHW_1.10.1
##   [9] aws.signature_0.4.4         munsell_0.5.0
##  [11] codetools_0.2-16            statmod_1.4.30
##  [13] withr_2.1.2                 colorspace_1.4-0
##  [15] highr_0.7                   knitr_1.21
##  [17] rstudioapi_0.9.0            stats4_3.5.2
##  [19] assertive.base_0.0-7        NMF_0.21.0
##  [21] labeling_0.3                slam_0.1-45
##  [23] GenomeInfoDbData_1.2.0      SiteMinderBMS_1.6.1
##  [25] lpsymphony_1.10.0           mnormt_1.5-5
```

```
##  [27] canvasXpress_1.23.3         bit64_0.9-7
##  [29] pheatmap_1.0.12             rprojroot_1.3-2
##  [31] generics_0.0.2              xfun_0.5
##  [33] R6_2.4.0                    doParallel_1.0.14
##  [35] GenomeInfoDb_1.18.2         assertive.sets_0.0-3
##  [37] locfit_1.5-9.1              ggiraph_0.6.0
##  [39] rex_1.1.2                   bitops_1.0-6
##  [41] DelayedArray_0.8.0          assertthat_0.2.0
##  [43] scales_1.0.0                gtable_0.2.0
##  [45] processx_3.2.1              rlang_0.3.1
##  [47] pathological_0.1-2          splines_3.5.2
##  [49] lazyeval_0.2.1              broom_0.5.1
##  [51] checkmate_1.9.1             yaml_2.2.0
##  [53] reshape2_1.4.3              modelr_0.1.4
##  [55] backports_1.1.3             qvalue_2.14.1
##  [57] tools_3.5.2                 psych_1.8.12
##  [59] gridBase_0.4-7              readat_1.8.0
##  [61] assertive.strings_0.0-3     RColorBrewer_1.1-2
##  [63] sessioninfo_1.1.1           assertive.reflection_0.0-4
##  [65] Rcpp_1.0.0                  plyr_1.8.4
##  [67] base64enc_0.1-3             progress_1.2.0
##  [69] zlibbioc_1.28.0             RCurl_1.95-4.11
##  [71] ps_1.3.0                    prettyunits_1.0.2
##  [73] S4Vectors_0.20.1            SummarizedExperiment_1.12.0
##  [75] haven_2.1.0                 ggrepel_0.8.0
##  [77] cluster_2.0.7-1             fs_1.2.6
##  [79] here_0.1                    tinytex_0.10
##  [81] data.table_1.12.0           openxlsx_4.1.0
##  [83] rXpress_1.4.18              packrat_0.5.0
##  [85] matrixStats_0.54.0          pkgload_1.0.2
##  [87] evaluate_0.13               hms_0.4.2
##  [89] xtable_1.8-3                XML_3.98-1.17
##  [91] readxl_1.3.0                IRanges_2.16.0
##  [93] gridExtra_2.3               testthat_2.0.1
##  [95] compiler_3.5.2              biomaRt_2.38.0
##  [97] crayon_1.3.4                htmltools_0.3.6
##  [99] sendmailR_1.2-1             lubridate_1.7.4
## [101] aws.s3_0.3.12               DBI_1.0.0
## [103] assertive.files_0.0-2       assertive.numbers_0.0-2
## [105] Matrix_1.2-15               RNASeqPower_1.22.1
## [107] cli_1.0.1                   assertive.types_0.0-3
## [109] gdata_2.18.0                igraph_1.2.4
## [111] GenomicRanges_1.34.0        pkgconfig_2.0.2
## [113] getPass_0.2-2               registry_0.5
## [115] foreign_0.8-71              xml2_1.2.0
## [117] roxygen2_6.1.1              foreach_1.4.4
## [119] annotate_1.60.0             rngtools_1.3.1
## [121] pkgmaker_0.27               XVector_0.22.0
## [123] bibtex_0.4.2                rvest_0.3.2
## [125] callr_3.1.1                 digest_0.6.18
## [127] rmarkdown_1.11              cellranger_1.1.0
## [129] edgeR_3.24.3                gdtools_0.1.7
## [131] supervisedCompliance_0.0.5  gtools_3.8.1
## [133] commonmark_1.7              rjson_0.2.20
```

```
## [135] jsonlite_1.6                fansi_0.4.0
## [137] desc_1.2.0                  limma_3.38.3
## [139] pillar_1.3.1                lattice_0.20-38
## [141] httr_1.4.0                  pkgbuild_1.0.2
## [143] survival_2.43-3             glue_1.3.0
## [145] remotes_2.0.2               zip_2.0.0
## [147] fdrtool_1.2.15              fortunes_1.5-4
## [149] iterators_1.0.10            bit_1.1-14
## [151] assertive.properties_0.0-4  stringi_1.3.1
## [153] blob_1.1.1                  memoise_1.1.0
```