DGE.Tools2

John Thompson (john.thompson@bms.com)

March 24 2017

Contents

1 Load Libraries

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

library(dplyr)
library(edgeR)
library(limma)
library(magrittr)
library(DGEobj)
library(DGE.Tools2)
```

2 Get raw count data and annotation from an Omicsoft project

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 (row) and samples (col) data (3 tabbed text files).

```
#change this to your working directory
# setwd("~/R/lib/pkgsrc/DGE.Tools2")
setwd("./")

#this should point to the installed library folder containing sample data
rawDataPath <- paste(.libPaths()[[1]], "/DGE.Tools2/extdata", sep="")
dgeObj <- OmicsoftToDgeObj(path=rawDataPath)
```

3 Get raw count data and annotation from an Xpress project

Find your data in Xpress and note the Xpress ID (at the end of the URL):
Example: http://xpress.pri.bms.com/CGI/project_summary.cgi?project=20135

Use function Xpress2DGEO from the Xpress2R package to retrieve the counts and annotation.

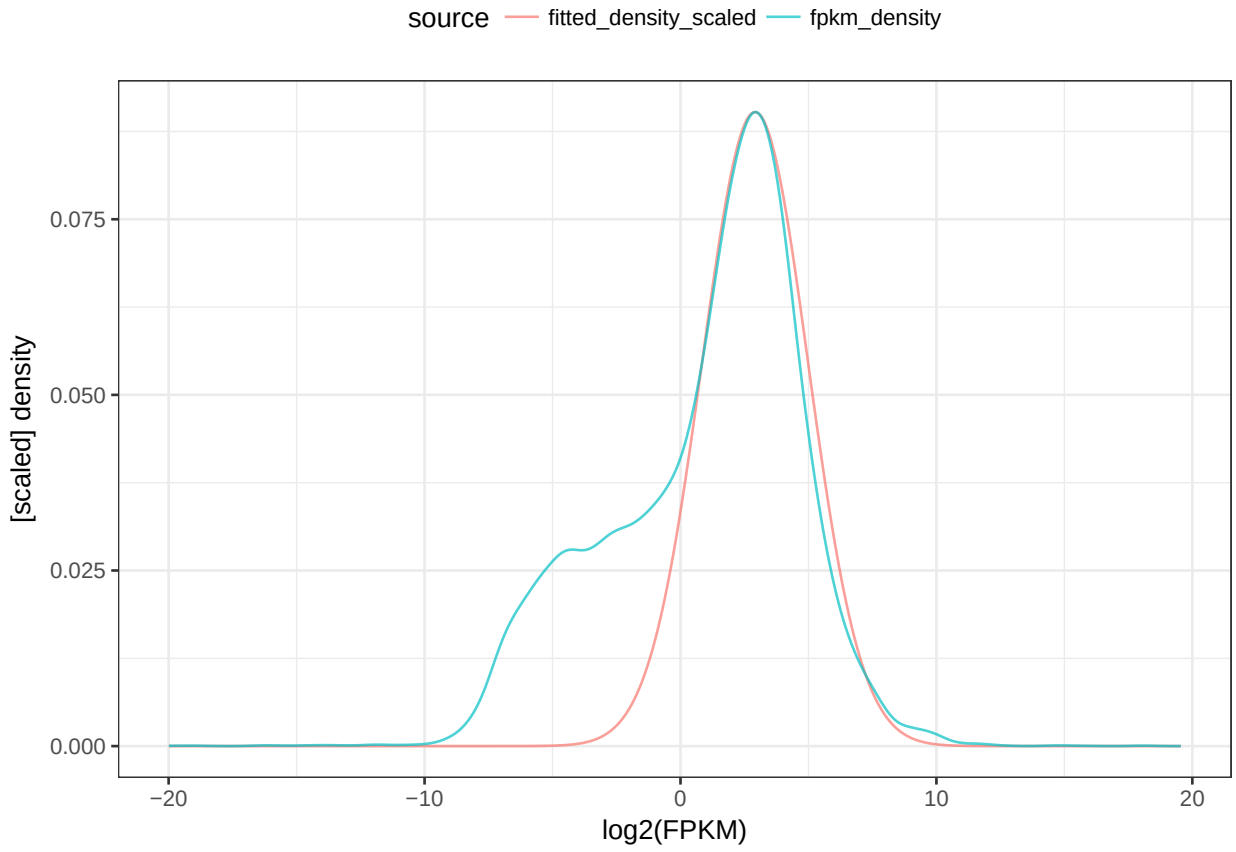
```
library(Xpress2R)
dgeObj20135 = Xpress2DGEO(20135)
```

4 zFPKM analysis

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 and 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 on the BMS biogit.

This shows how to calculate zFPKM. zFPKM >= -3 is the recommended threshold to use as a cutoff for "present" genes. To rely on zFPKM you need to 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 <- getItem(dgeObj, "counts")
geneAnno <- getItem(dgeObj, "geneData")
FPKM <- convertCounts(counts, unit="FPKM",
                      geneLength=geneAnno$ExonLength) %>%
    as.data.frame
library(zFPKM)
zFPKM <- zFPKMTransformDF(FPKM[,1, drop=FALSE])
```



5 Filter out low intensity genes

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

Although zFPKM is more unbiased (in terms of genelength) than the mincount method of low intensity filtering, I have a preference for the FPK filtering methods. FPK is also unbiased with respect to genelength and has the further advantage that is can be used to evaluate the "biological" noise floor by calculating the FPK value for intergenic DNA. Reads mapping to intergenic DNA are thought to arise by either low-level stochastic non-promoter driven transcription or DNA contamination. This represents a baseline below which you cannot infer mRNA expression.

I typically use a FPK + mincount filter to define detected/non detected for individual samples. 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 here just use the simplistic method of requiring XX% of

samples to pass the intensity threshold.

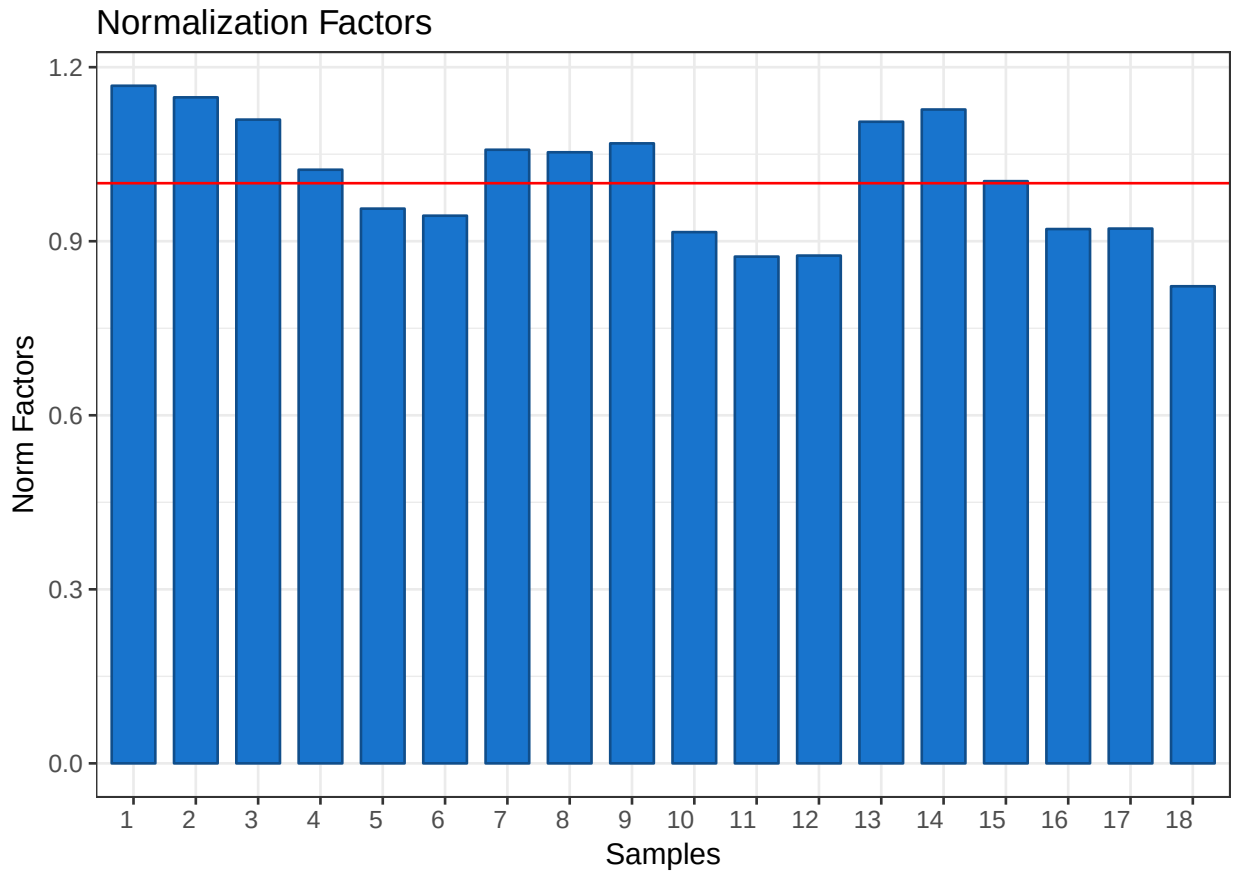
Dimensions before filtering:
27453, 18

```
fracThreshold <- 0.3

#low expression filter
counts <- getItem(dgeObj, "counts")
genelength <-getItem(dgeObj, "geneData")$ExonLength
fpk <- convertCounts(counts,
                     unit="FPK",
                     geneLength=genelength)

#keep FPK >=5 in 75% of samples
idxfpk <- fpk >= 5.0
frac <- rowSums(fpk) / ncol(fpk)
idx <- frac >= fracThreshold
dgeObj <- subset(dgeObj, idx, 1:ncol(dgeObj))

#overlay a mincount filter
counts <- getItem(dgeObj, "counts")
genelength <-getItem(dgeObj, "geneData")$ExonLength
idxmin <- counts >= 10
frac <- rowSums(idxmin)/ncol(idxmin)
idx <- frac >= fracThreshold
dgeObj <- subset(dgeObj, idx)
```

Dimensions after filtering:
14498, 18

6 EdgeR Normalization

```
dgeObj <- runEdgeRNorm(dgeObj)
```



7 Define the model

Provide a formula and construct the design matrix.

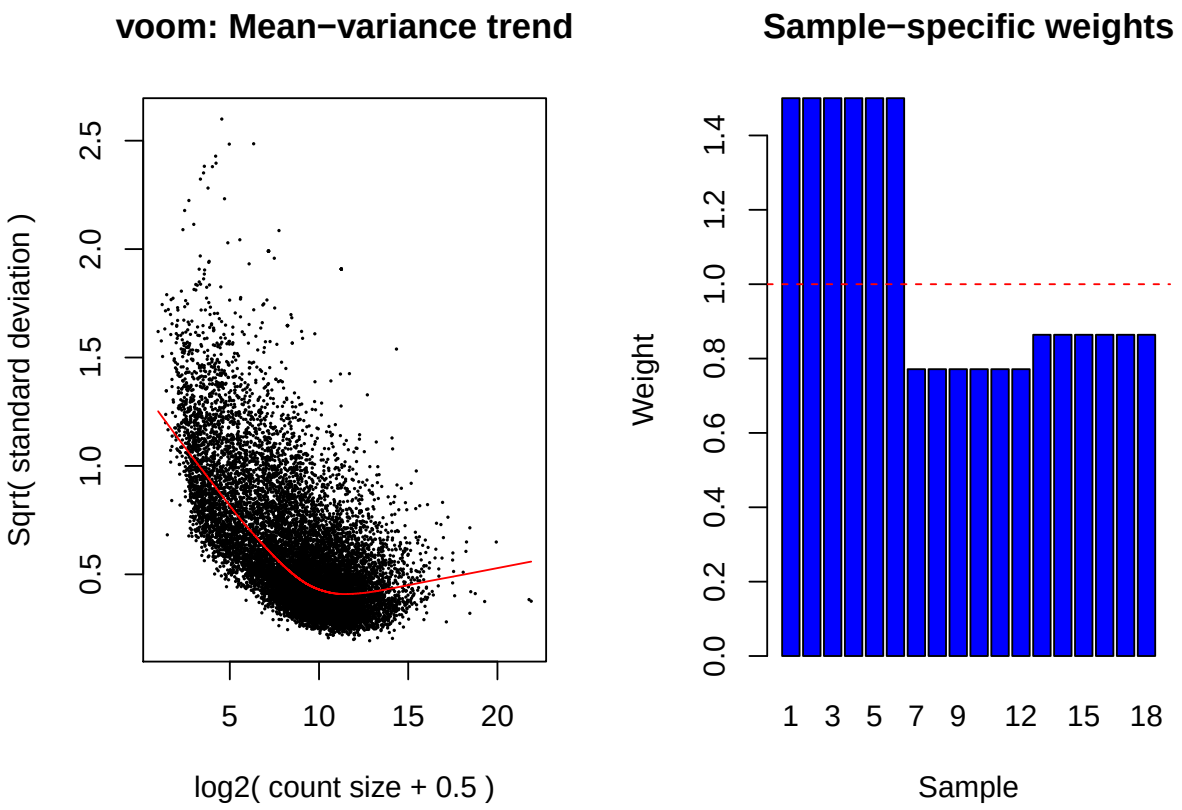
```
#define a formula and construct a design matrix
design <- getItem(dgeObj, "design")
design$ReplicateGroup %<>% as.factor
design$ReplicateGroup %<>% relevel("Normal_control")
formula <- '~ 0 + ReplicateGroup'
#build the designMatrix and add some attributes
designMatrix <- model.matrix (as.formula(formula), design)
#give this design a name
designMatrixName <- "Treatment"
#capture the formula as an attribute
designMatrix <- setAttributes(designMatrix, list(formula=formula))
#save the designMatrix
dgeObj <- addItem(dgeObj, item=designMatrix,
                  itemName=designMatrixName,
                  itemType="designMatrix",
                  parent="design")
```

8 QC: Dispersion Plot

```
#this plot takes a few minutes
#dispersion plot
dispPlot <- plotDisp(getItem(dgeObj, "DGEList"), designMatrix)
dispPlot + baseTheme(18)
```

9 Run Voom and fit the model (lmfit)

```
#QW and Var.design and dupCor
# block <- c(1,2,3,1,2,3,4,5,6,4,5,6,7,8,9,7,8,9)
# vd <- model.matrix(as.formula("~ Treatment"), design)
# dgeObj <- runVoom(dgeObj, designMatrixName,
#                qualityWeights = TRUE,
#                var.design=vd,
#                dupcorBlock=block)

#Omit duplicate correlation to run faster as an example
#QW and Var.design
block <- c(1,2,3,1,2,3,4,5,6,4,5,6,7,8,9,7,8,9)
#the vd design matrix is used to block the quality weight determination
vd <- model.matrix(as.formula("~ Disease.Status"), design)
dgeObj <- runVoom(dgeObj, designMatrixName,
              qualityWeights = TRUE,
              var.design=vd)
```

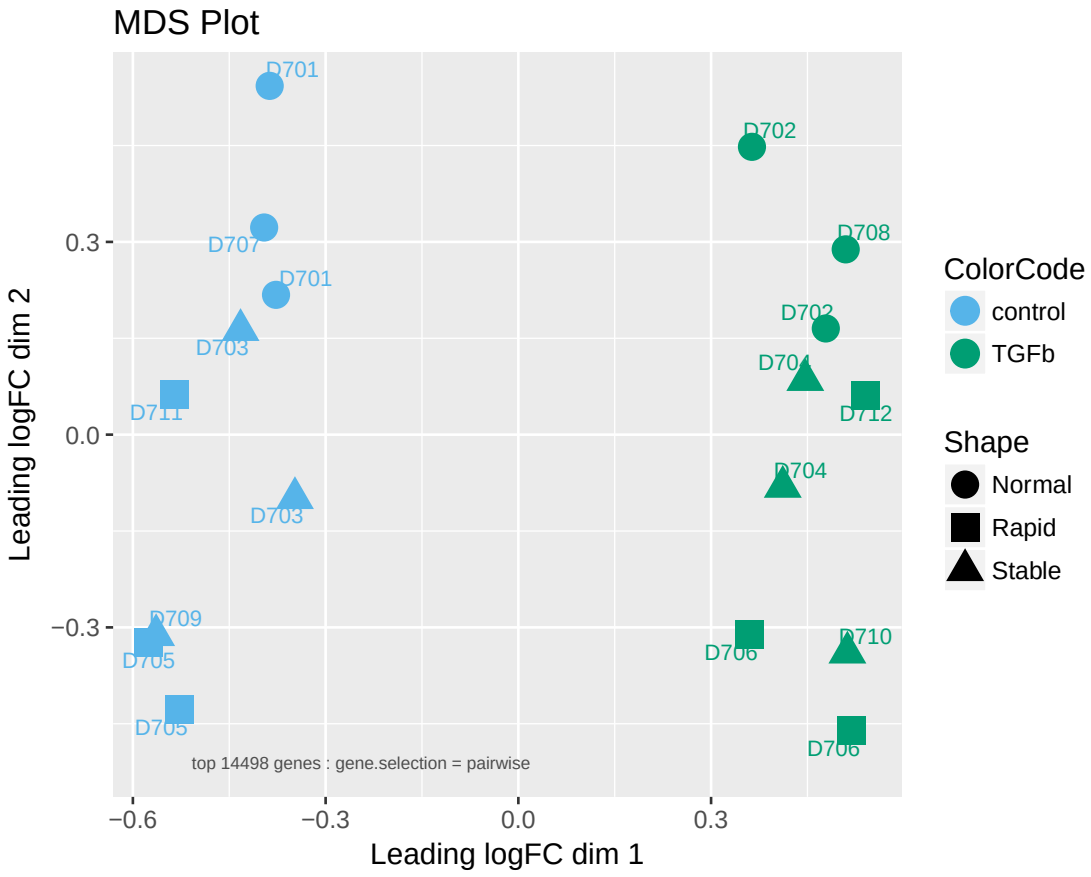


I'm getting a warning here I don't understand: coming from appendAttributes. However, the resulting DGEobj looks correct.

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

10 Data Exploration: MDS plot

```
#use color and shape with labels and labelSize
m <- ggplotMDS(dgeObj, colorBy = design$Treatment,
               shapeBy = design$Disease.Status, symSize =5,
               labels=design$VendorBarcode,
               labelSize=3)
print(m[[1]])
```



11 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.

```
#runContrast testing
contrastList  <- list(TGF_Norm = "ReplicateGroupNormal_TGFb - ReplicateGroupNormal_control",
                      TGF_Stable = "ReplicateGroupStable_TGFb - ReplicateGroupStable_control",
                      TGF_Rapid = "ReplicateGroupRapid_TGFb - ReplicateGroupRapid_control"
)
# library(assertthat)
dgeObj <- runContrasts(dgeObj,
                                designMatrixName="Treatment",
                                contrastList=contrastList,
                                runTopTreat=T)
saveRDS(dgeObj, "../DGEobj.RDS")
```

12 Check for Surrogate Variables (unaccounted for variation)

If SVA find surrogate variables, you cbind the design table for the surrogate variables to your experiments design matrix and then re-run voom/lmfit.

12.1 runSVA

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

```
## No significant surrogate variables
```

SVA found 0 surrogate variables in this dataset. We'll now re-run runVoom with the new designMatrix.

12.2 Re-Run voom/lmfit

If SVA had found surrogate variables, a new design matrix with a "_sva" suffix was created and added to the DGEobj and you would rerun the runVoom function with the new design matrix:

```
block <- c(1,2,3,1,2,3,4,5,6,4,5,6,7,8,9,7,8,9)
vd <- model.matrix(as.formula("~ Treatment"), design)
dgeObj_sva <- runVoom(dgeObj_sva, designMatrixName="Treatment_sva",
              qualityWeights = TRUE,
              var.design=vd)
```

12.3 Re-run Contrasts

After you re-ran runVoom you would also re-run runContrasts as follows:

```
#runContrast testing
contrastList  <- list(TGF_Norm = "ReplicateGroupNormal_TGFb - ReplicateGroupNormal_control",
                      TGF_Stable = "ReplicateGroupStable_TGFb - ReplicateGroupStable_control",
                      TGF_Rapid = "ReplicateGroupRapid_TGFb - ReplicateGroupRapid_control"
)
# library(assertthat)
dgeObj <- runContrasts(dgeObj_sva,
                                designMatrixName="Treatment_sva",
                                contrastList=contrastList,
                                runTopTreat=T)
saveRDS(dgeObj, "../DGEobj.RDS")
```

13 Alternative FDR scores

topTable provides a BH FDR value (adj.P.Val). We provide two functions (runQvalue and runIHW) that provide alternative FDR measures. See the help for each of those functions for details of how each method determines the FDR value.

Here we extract all topTable dataframes as a list. Then we loop through and add the specified FDR measures as additional columns in the topTable dataframes and return the original list of dataframes.

13.1 runQvalue

```
#Qvalue test
#extract a topTable List
contrastList <- getType(dgeObj, type="topTable", parent="Treatment_fit_cf")
```

```
contrastList2 <- runQvalue(contrastList)

#now put the contrasts back in the DGEobj
DgeObj_q <- addItems(dgeObj,
                     itemList=contrastList2,
                     itemTypes=as.list(rep("topTable", length(contrastList2))),
                     overwrite=TRUE)
```

13.2 runIHW

```
#IHW test
#extract a topTable List
contrastList <- getType(dgeObj, type="topTable", parent="Treatment_fit_cf")
IHWresult <- runIHW(contrastList)
contrastList2 <- IHWresult[[1]]

#now put the contrasts back in the DGEobj
DgeObj_ihw <- addItems(dgeObj,
                       itemList=contrastList2,
                       itemTypes=as.list(rep("topTable", length(contrastList2))),
                       overwrite=TRUE)
```

14 Printing a DGEobj

The print method has been adapted to print out an informative table of information that is useful to inspect the contents of a DGEobj and identify the names of individual items.

Simple print:

```
library(knitr)
print(dgeObj)
```

```
        ItemName           ItemType BaseType            Parent
```

1 counts_orig counts_orig meta
2 counts counts assay counts_orig 3 design_orig design_orig meta
4 design design col design_orig 5 geneData_orig geneData_orig meta
6 geneData geneData row geneData_orig 7 granges_orig granges_orig meta geneData_orig 8 granges granges row geneData 9 DGEList DGEList assay counts 10 Treatment designMatrix col design 11 Treatment_Elist Elist assay DGEList 12 Treatment_fit fit row Treatment_Elist 13 Treatment_fit_cm contrastMatrix meta Treatment_fit 14 Treatment_fit_cf contrast_fit row Treatment_fit 15 TGF_Norm topTable row Treatment_fit_cf 16 TGF_Stable topTable row Treatment_fit_cf 17 TGF_Rapid topTable row Treatment_fit_cf 18 Treatment_fit_cft contrast_fit_treat meta Treatment_fit 19 TGF_Norm_treat topTreat meta Treatment_fit_cft 20 TGF_Stable_treat topTreat meta Treatment_fit_cft 21 TGF_Rapid_treat topTreat meta Treatment_fit_cft DateCreated 1 2017-03-24 10:16:43 2 2017-03-24 10:16:43 3 2017-03-24 10:16:43 4 2017-03-24 10:16:43 5 2017-03-24 10:16:43 6 2017-03-24 10:16:43 7 2017-03-24 10:16:43 8 2017-03-24 10:16:43 9 2017-03-24 10:16:46 10 2017-03-24 10:16:46 11 2017-03-24 10:16:55 12 2017-03-24 10:16:55 13 2017-03-24 10:16:58 14 2017-03-24 10:16:58 15 2017-03-24 10:16:58 16 2017-03-24 10:16:58 17 2017-03-24 10:16:58 18 2017-03-24 10:16:58 19 2017-03-24 10:16:58 20 2017-03-24 10:16:58 21 2017-03-24 10:16:58

Verbose print:

```
#results='asis' argument makes the kable output of the DGEobj print methods
#format nicely as expected.

print(dgeObj, verbose=TRUE)
```

```
        ItemName           ItemType BaseType            Parent
```

1 counts_orig counts_orig meta
2 counts counts assay counts_orig 3 design_orig design_orig meta
4 design design col design_orig 5 geneData_orig geneData_orig meta
6 geneData geneData row geneData_orig 7 granges_orig granges_orig meta geneData_orig 8 granges granges row geneData 9 DGEList DGEList assay counts 10 Treatment designMatrix col design 11 Treatment_Elist Elist assay DGEList 12 Treatment_fit fit row Treatment_Elist 13 Treatment_fit_cm contrastMatrix meta Treatment_fit 14 Treatment_fit_cf contrast_fit row Treatment_fit 15 TGF_Norm topTable row Treatment_fit_cf 16 TGF_Stable topTable row Treatment_fit_cf 17 TGF_Rapid topTable row Treatment_fit_cf 18 Treatment_fit_cft contrast_fit_treat meta Treatment_fit 19 TGF_Norm_treat topTreat meta Treatment_fit_cft 20 TGF_Stable_treat topTreat meta Treatment_fit_cft 21 TGF_Rapid_treat topTreat meta Treatment_fit_cft DateCreated 1 2017-03-24 10:16:43 2 2017-03-24 10:16:43 3 2017-03-24 10:16:43 4 2017-03-24 10:16:43 5 2017-03-24 10:16:43 6 2017-03-24 10:16:43 7 2017-03-24 10:16:43 8 2017-03-24 10:16:43 9 2017-03-24 10:16:46 10 2017-03-24 10:16:46 11 2017-03-24 10:16:55 12 2017-03-24 10:16:55 13 2017-03-24 10:16:58 14 2017-03-24 10:16:58 15 2017-03-24 10:16:58 16 2017-03-24 10:16:58 17 2017-03-24 10:16:58 18 2017-03-24 10:16:58 19 2017-03-24 10:16:58 20 2017-03-24 10:16:58 21 2017-03-24 10:16:58 FunArgs 1 initDGEobj(countData, seqData, designData, level) 2 initDGEobj(countData, seqData, designData, level) 3 initDGEobj(countData, seqData, designData, level) 4 initDGEobj(countData, seqData, designData, level) 5 initDGEobj(countData, seqData, designData, level) 6 initDGEobj(countData, seqData, designData, level) 7 initDGEobj(countData, seqData, designData, level) 8 initDGEobj(countData, seqData, designData, level) 9 runEdgeRNorm(dgeObj) 10 addItem(dgeObj, designMatrix, designMatrixName, designMatrix, design) 11 runVoom(dgeObj, designMatrixName, TRUE, vd) 12 runVoom(dgeObj, designMatrixName, TRUE, vd) 13 runContrasts(dgeObj, Treatment, contrastList, T) 14 runContrasts(dgeObj, Treatment, contrastList, T) 15 runContrasts(dgeObj, Treatment, contrastList, T) 16 runContrasts(dgeObj, Treatment, contrastList, T) 17 runContrasts(dgeObj, Treatment, contrastList, T) 18 runContrasts(dgeObj, Treatment, contrastList, T) 19 runContrasts(dgeObj, Treatment, contrastList, T) 20 runContrasts(dgeObj, Treatment, contrastList, T) 21 runContrasts(dgeObj, Treatment, contrastList, T)

15 Session Info

Time required to process this report: *31.98197 secs*

R Session Info

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

```
## R version 3.3.3 (2017-03-06)
## Platform: i386-w64-mingw32/i386 (32-bit)
## Running under: Windows 7 (build 7601) Service Pack 1
##
## 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] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.15.1         zFPKM_0.901          DGE.Tools2_0.9.3
## [4] DGEobj_0.9.1         magrittr_1.5         edgeR_3.16.5
## [7] limma_3.30.12        dplyr_0.5.0          BiocInstaller_1.24.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10               xml2_1.1.1
##  [3] XVector_0.14.0             GenomicRanges_1.26.3
##  [5] IRanges_2.8.1              splines_3.3.3
##  [7] BiocGenerics_0.20.0        zlibbioc_1.20.0
##  [9] statmod_1.4.29             xtable_1.8-2
## [11] colorspace_1.3-2           lattice_0.20-34
## [13] R6_2.2.0                   IHW_1.2.0
## [15] parallel_3.3.3             R.oo_1.21.0
## [17] genefilter_1.56.0          htmltools_0.3.5
## [19] assertthat_0.1             rprojroot_1.2
## [21] digest_0.6.12              tibble_1.2
## [23] Matrix_1.2-8               lpsymphony_1.2.0
## [25] ggrepel_0.6.5              RCurl_1.95-4.8
## [27] slam_0.1-40                rmarkdown_1.3
## [29] scales_0.4.1               backports_1.0.5
## [31] gdtools_0.1.4              R.methodsS3_1.7.1
## [33] stats4_3.3.3               locfit_1.5-9.1
## [35] lubridate_1.6.0            annotate_1.52.1
## [37] AnnotationDbi_1.36.2       munsell_0.4.3
## [39] fdrtool_1.2.15             GenomeInfoDb_1.10.3
## [41] plyr_1.8.4                 stringr_1.2.0
## [43] tools_3.3.3                grid_3.3.3
```

```
## [45] SummarizedExperiment_1.4.0 nlme_3.1-131
## [47] Biobase_2.34.0             gtable_0.2.0
## [49] mgcv_1.8-17                rvg_0.1.3
## [51] ggiraph_0.3.2              DBI_0.6
## [53] lazyeval_0.2.0             yaml_2.1.14
## [55] survival_2.41-2            gridExtra_2.2.1
## [57] sva_3.22.0                 purrr_0.2.2
## [59] ggplot2_2.2.1              officer_0.1.1
## [61] tidyr_0.6.1                reshape2_1.4.2
## [63] bitops_1.0-6               R.utils_2.5.0
## [65] S4Vectors_0.12.1           htmlwidgets_0.8
## [67] base64enc_0.1-3            qvalue_2.6.0
## [69] evaluate_0.10              RSQLite_1.1-2
## [71] memoise_1.0.0              labeling_0.3
## [73] stringi_1.1.2              XML_3.98-1.5
```