EDASeq: Exploratory Data Analysis and Normalization for RNA-Seq

Davide Risso

Modified: April 25, 2015. Compiled: June 22, 2018

Contents

1 Introduction

In this document, we show how to conduct Exploratory Data Analysis (EDA) and normalization for a typical RNA-Seq experiment using the package *EDASeq*.

One can think of EDA for RNA-Seq as a two-step process: "read-level" EDA helps in discovering lanes with low sequencing depths, quality issues, and unusual nucleotide frequencies, while "gene-level" EDA can capture mislabeled lanes, issues with distributional assumptions (e.g., over-dispersion), and GC-content bias.

The package also implements both "within-lane" and "between-lane" normalization procedures, to account, respectively, for within-lane gene-specific (and possibly lane-specific) effects on read counts (e.g., related to gene length or GC-content) and for between-lane distributional differences in read counts (e.g., sequencing depths).

To illustrate the functionality of the *EDASeq* package, we make use of the *Saccharomyces cerevisiae* RNA-Seq data from Lee et al. [1]. Briefly, a wild-type strain and three mutant strains were sequenced using the Solexa 1G Genome Analyzer. For each strain, there are four technical replicate lanes from the same library preparation. The reads were aligned using *Bowtie* [2], with unique mapping and allowing up to two mismatches.

The *leeBamViews* package provides a subset of the aligned reads in BAM format. In particular, only the reads mapped between bases 800,000 and 900,000 of chromosome XIII are considered. We use these reads to illustrate read-level EDA.

The *yeastRNASeq* package contains gene-level read counts for four lanes: two replicates of the wild-type strain ("wt") and two replicates of one of the mutant strains ("mut"). We use these data to illustrate gene-level EDA.

```
require(EDASeq)
require(yeastRNASeq)
require(leeBamViews)
```

2 Reading in unaligned and aligned read data

Unaligned reads. Unaligned (unmapped) reads stored in FASTQ format may be managed via the class *FastqFileList* imported from *ShortRead*. Information related to the libraries sequenced in each lane can be stored in the `elementMetadata` slot of the *FastqFileList* object.

```
files <- list.files(file.path(system.file(package = "yeastRNASeq"),
                              "reads"), pattern = "fastq", full.names = TRUE)
names(files) <- gsub("\\.fastq.*", "", basename(files))
met <- DataFrame(conditions=c(rep("mut",2), rep("wt",2)),
                 row.names=names(files))
fastq <- FastqFileList(files)
elementMetadata(fastq) <- met
fastq
```

```
## FastqFileList of length 4
## names(4): mut_1_f mut_2_f wt_1_f wt_2_f
```

Aligned reads. The package can deal with aligned (mapped) reads in BAM format, using the class *BamFileList* from *Rsamtools*. Again, the `elementMetadata` slot can be used to store lane-level sample information.

```
files <- list.files(file.path(system.file(package = "leeBamViews"), "bam"),
                    pattern = "bam$", full.names = TRUE)
names(files) <- gsub("\\.bam", "", basename(files))
```

```
gt <- gsub(".*/", "", files)
gt <- gsub("_.*", "", gt)
lane <- gsub(".*(.)$", "\\1", gt)
geno <- gsub(".$", "", gt)

pd <- DataFrame(geno=geno, lane=lane,
                row.names=paste(geno,lane,sep="."))

bfs <- BamFileList(files)
elementMetadata(bfs) <- pd
bfs
```

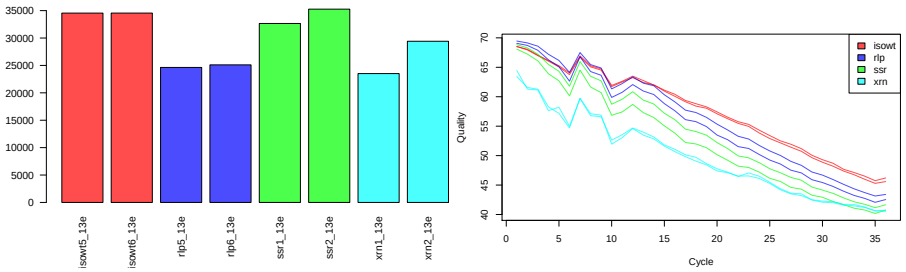
```
## BamFileList of length 8
## names(8): isowt5_13e isowt6_13e ... xrn1_13e xrn2_13e
```

3 Read-level EDA

Numbers of unaligned and aligned reads. One important check for quality control is to look at the total number of reads produced in each lane, the number and the percentage of reads mapped to a reference genome. A low total number of reads might be a symptom of low quality of the input RNA, while a low mapping percentage might indicate poor quality of the reads (low complexity), problems with the reference genome, or mislabeled lanes.

```
colors <- c(rep(rgb(1,0,0,alpha=0.7),2),
            rep(rgb(0,0,1,alpha=0.7),2),
            rep(rgb(0,1,0,alpha=0.7),2),
            rep(rgb(0,1,1,alpha=0.7),2))
barplot(bfs,las=2,col=colors)

plotQuality(bfs,col=colors,lty=1)
legend("topright",unique(elementMetadata(bfs)[,1]), fill=unique(colors))
```



(a) Number of mapped reads **(b)** Mean per-base quality of mapped reads

Figure 1: Per-lane number of mapped reads and quality scores

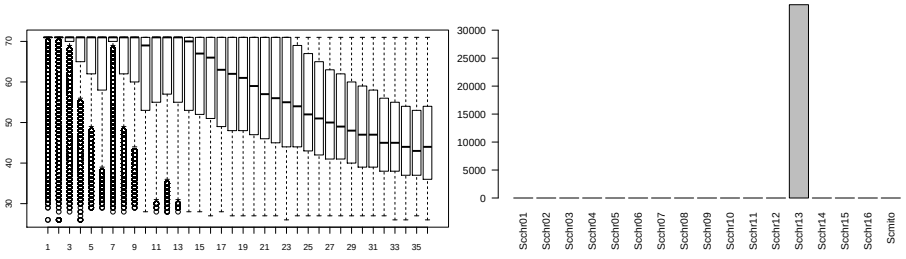
Figure 1a, produced using the `barplot` method for the *BamFileList* class, displays the number of mapped reads for the subset of the yeast dataset included in the package *leeBamViews*. Unfortunately, *leeBamViews* does not provide unaligned reads, but barplots of the total number of reads can be obtained using the `barplot` method for the

FastqFileList class. Analogously, one can plot the percentage of mapped reads with the `plot` method with signature `c(x="BamFileList", y="FastqFileList")`. See the manual pages for details.

Read quality scores. As an additional quality check, one can plot the mean per-base (i.e., per-cycle) quality of the unmapped or mapped reads in every lane (Figure 1b).

Individual lane summaries. If one is interested in looking more thoroughly at one lane, it is possible to display the per-base distribution of quality scores for each lane (Figure 2a) and the number of mapped reads stratified by chromosome (Figure 2b) or strand. As expected, all the reads are mapped to chromosome XIII.

```
plotQuality(bfs[[1]],cex.axis=.8)
barplot(bfs[[1]],las=2)
```



(a) Per-base quality of mapped reads **(b)** Number of mapped reads per-chromosome

Figure 2: Quality scores and number of mapped reads for lane 'isowt5_13e.'

Read nucleotide distributions. A potential source of bias is related to the sequence composition of the reads. The function `plotNtFrequency` plots the per-base nucleotide frequencies for all the reads in a given lane (Figure 3).

```
plotNtFrequency(bfs[[1]])
```

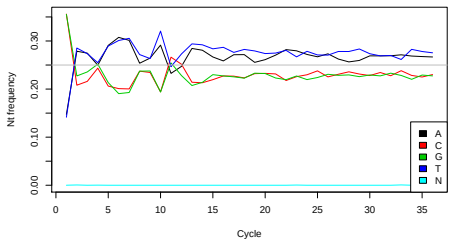


Figure 3: Per-base nucleotide frequencies of mapped reads for lane 'isowt5_13e.'

4 Gene-level EDA

Examining statistics and quality metrics at a read level can help in discovering problematic libraries or systematic biases in one or more lanes. Nevertheless, some biases can be difficult to detect at this scale and gene-level EDA is equally important.

Classes and methods for gene-level counts. There are several Bioconductor packages for aggregating reads over genes (or other genomic regions, such as, transcripts and exons) given a particular genome annotation, e.g., *IRanges*, *ShortRead*, *Genominator*, *Rsubread*. See their respective vignettes for details.

Here, we consider this step done and load the object `geneLevelData` from *yeastRNASeq*, which provides gene-level counts for 2 wild-type and 2 mutant lanes from the yeast dataset of Lee et al. [1] (see the *Genominator* vignette for an example on the same dataset).

```
data(geneLevelData)
head(geneLevelData)
```

```
##         mut_1 mut_2 wt_1 wt_2
## YHR055C     0     0    0    0
## YPR161C    38    39   35   34
## YOL138C    31    33   40   26
## YDR395W    55    52   47   47
## YGR129W    29    26    5    5
## YPR165W   189   180  151  180
```

Since it is useful to explore biases related to length and GC-content, the *EDASeq* package provides, for illustration purposes, length and GC-content for *S. cerevisiae* genes (based on SGD annotation, version r64 [3]). Functionality for automated retrieval of gene length and GC-content is introduced in the last section of the vignette.

```
data(yeastGC)
head(yeastGC)
```

```
##   YAL001C   YAL002W   YAL003W   YAL004W   YAL005C   YAL007C
## 0.3712317 0.3717647 0.4460548 0.4490741 0.4406428 0.3703704
```

```
data(yeastLength)
head(yeastLength)
```

```
## YAL001C YAL002W YAL003W YAL004W YAL005C YAL007C
##    3483    3825     621     648    1929     648
```

First, we filter the non-expressed genes, i.e., we consider only the genes with an average read count greater than 10 across the four lanes and for which we have length and GC-content information.

```
filter <- apply(geneLevelData,1,function(x) mean(x)>10)
table(filter)
```

```
## filter
## FALSE  TRUE
##  1988  5077
```

```
common <- intersect(names(yeastGC),
                    rownames(geneLevelData[filter,]))
length(common)
```

```
## [1] 4994
```

This leaves us with 4994 genes.

The *EDASeq* package provides the *SeqExpressionSet* class to store gene counts, (lane-level) information on the sequenced libraries, and (gene-level) feature information. We use the data frame `met` created in Section 2 for the lane-level data. As for the feature data, we use gene length and GC-content.

```
feature <- data.frame(gc=yeastGC,length=yeastLength)
data <- newSeqExpressionSet(counts=as.matrix(geneLevelData[common,]),
                            featureData=feature[common,],
                            phenoData=data.frame(
                              conditions=c(rep("mut",2),rep("wt",2)),
                              row.names=colnames(geneLevelData)))
data
```

```
## SeqExpressionSet (storageMode: lockedEnvironment)
## assayData: 4994 features, 4 samples
##   element names: counts, normalizedCounts, offset
## protocolData: none
## phenoData
##   sampleNames: mut_1 mut_2 wt_1 wt_2
##   varLabels: conditions
##   varMetadata: labelDescription
## featureData
##   featureNames: YAL001C YAL002W ... YPR201W (4994
##     total)
##   fvarLabels: gc length
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

Note that the row names of `counts` and `featureData` must be the same; likewise for the row names of `phenoData` and the column names of `counts`. As in the *CountDataSet* class, the expression values can be accessed with `counts`, the lane information with `pData`, and the feature information with `fData`.

```
head(counts(data))
```

```
##         mut_1 mut_2 wt_1 wt_2
## YAL001C    80    83   27   40
## YAL002W    33    38   53   66
## YAL003W  1887  1912  270  270
## YAL004W    90   110  276  295
## YAL005C   325   316  874  935
## YAL007C    27    30   19   24
```

```
pData(data)
```

```
##       conditions
## mut_1        mut
## mut_2        mut
## wt_1          wt
## wt_2          wt
```

```
head(fData(data))
```

```
##                 gc length
## YAL001C 0.3712317   3483
## YAL002W 0.3717647   3825
## YAL003W 0.4460548    621
## YAL004W 0.4490741    648
## YAL005C 0.4406428   1929
## YAL007C 0.3703704    648
```

The *SeqExpressionSet* class has two additional slots: `normalizedCounts` and `offset` (matrices of the same dimension as `counts`), which may be used to store a matrix of normalized counts and of normalization offsets, respectively, to be used for subsequent analyses (see Section 5 and the *edgeR* vignette for details on the role of offsets). If not specified, the offset is initialized as a matrix of zeros.

```
head(offst(data))
```

```
##         mut_1 mut_2 wt_1 wt_2
## YAL001C     0     0    0    0
## YAL002W     0     0    0    0
## YAL003W     0     0    0    0
## YAL004W     0     0    0    0
## YAL005C     0     0    0    0
## YAL007C     0     0    0    0
```

Between-lane distribution of gene-level counts. One of the main considerations when dealing with gene-level counts is the difference in count distributions between lanes. The `boxplot` method provides an easy way to produce boxplots of the logarithms of the gene counts in each lane (Figure 4).

```
boxplot(data,col=colors[1:4])
```

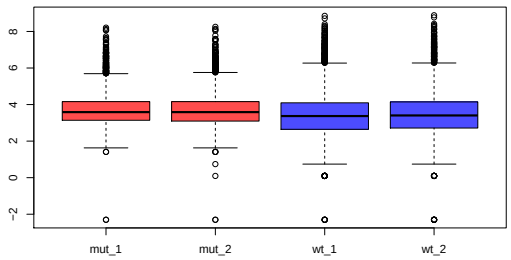


Figure 4: Between-lane distribution of gene-level counts (log)

The `MDPlot` method produces a mean-difference plot (MD-plot) of read counts for two lanes (Figure 5).

```
MDPlot(data,c(1,3))
```

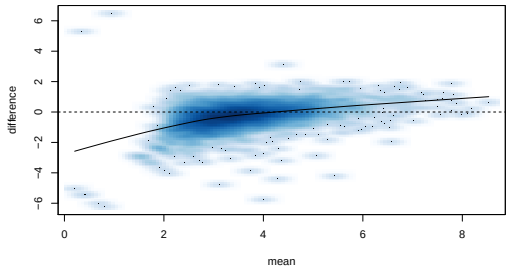
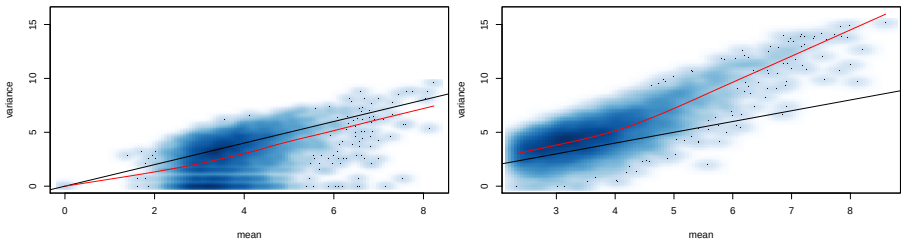


Figure 5: Mean-difference plot of the gene-level counts (log) of lanes 'mut_1' and 'wt_1.'

Over-dispersion. Although the Poisson distribution is a natural and simple way to model count data, it has the limitation of assuming equality of the mean and variance. For this reason, the negative binomial distribution has been proposed as an alternative when the data show over-dispersion. The function `meanVarPlot` can be used to check whether the count data are over-dispersed (for the Poisson distribution, one would expect the points in Figures 6 to be evenly scattered around the black line).

```
meanVarPlot(data[,1:2], log=TRUE, ylim=c(0,16))
meanVarPlot(data, log=TRUE, ylim=c(0,16))
```



(a) Mutant lanes **(b)** All four lanes

Figure 6: Mean-variance relationship for the two mutant lanes and all four lanes: the black line corresponds to the Poisson distribution (variance equal to the mean), while the red curve is a lowess fit

Note that the mean-variance relationship should be examined within replicate lanes only (i.e., conditional on variables expected to contribute to differential expression). For the yeast dataset, it is not surprising to see no evidence of over-dispersion for the two mutant technical replicate lanes (Figure 6a); likewise for the two wild-type lanes. However, one expects over-dispersion in the presence of biological variability, as seen in Figure 6b when considering at once all four mutant and wild-type lanes [4–6].

Gene-specific effects on read counts. Several authors have reported selection biases related to sequence features such as gene length, GC-content, and mappability [4, 7–9].

In Figure 7, obtained using `biasPlot`, one can see the dependence of gene-level counts on GC-content. The same plot could be created for gene length or mappability instead of GC-content.

```
biasPlot(data, "gc", log=TRUE, ylim=c(1,5))
```

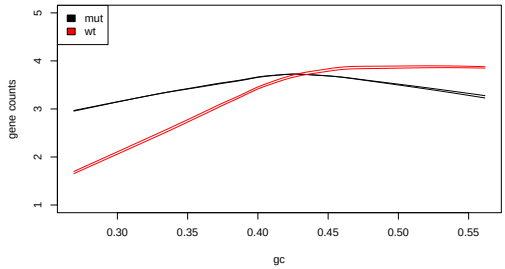


Figure 7: Lowess regression of the gene-level counts (log) on GC-content for each lane, color-coded by experimental condition

To show that GC-content dependence can bias differential expression analysis, one can produce stratified boxplots of the log-fold-change of read counts from two lanes using the `biasBoxplot` method (Figure 8). Again, the same type of plots can be created for gene length or mappability.

```
lfc <- log(counts(data)[,3]+0.1) - log(counts(data)[,1]+0.1)
biasBoxplot(lfc, fData(data)$gc)
```

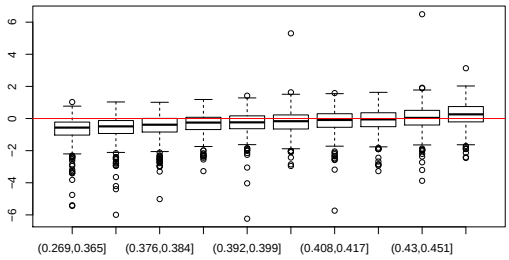


Figure 8: Boxplots of the log-fold-change between 'mut_1' and 'wt_1' lanes stratified by GC-content

5 Normalization

Following Risso et al. [8], we consider two main types of effects on gene-level counts: (1) within-lane gene-specific (and possibly lane-specific) effects, e.g., related to gene length or GC-content, and (2) effects related to between-lane distributional differences, e.g., sequencing depth. Accordingly, `withinLaneNormalization` and `betweenLaneNormalization` adjust for the first and second type of effects, respectively. We recommend to normalize for within-lane effects prior to between-lane normalization.

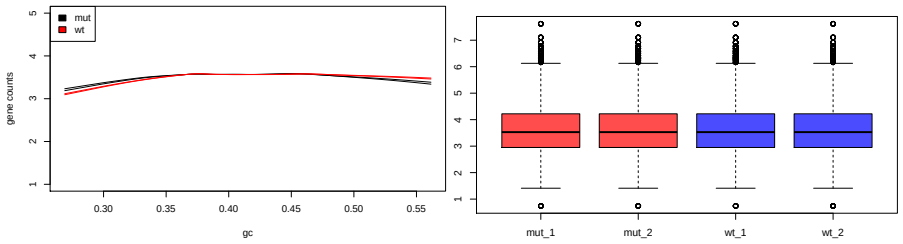
We implemented four within-lane normalization methods, namely: loess robust local regression of read counts (log) on a gene feature such as GC-content (`loess`), global-scaling between feature strata using the median (`median`), global-scaling between feature strata using the upper-quartile (`upper`), and full-quantile normalization between feature strata (`full`). For a discussion of these methods in context of GC-content normalization see Risso et al. [8].

```
dataWithin <- withinLaneNormalization(data,"gc", which="full")
dataNorm <- betweenLaneNormalization(dataWithin, which="full")
```

Regarding between-lane normalization, the package implements three of the methods introduced in Bullard et al. [4]: global-scaling using the median (`median`), global-scaling using the upper-quartile (`upper`), and full-quantile normalization (`full`).

Figure 9 shows how after full-quantile within- and between-lane normalization, the GC-content bias is reduced and the distribution of the counts is the same in each lane.

```
biasPlot(dataNorm, "gc", log=TRUE, ylim=c(1,5))
boxplot(dataNorm, col=colors)
```



(a) GC-content **(b)** Count distribution

Figure 9: Full-quantile within- and between-lane normalization
(a) Lowess regression of normalized gene-level counts (log) on GC-content for each lane. (b) Between-lane distribution of normalized gene-level counts (log).

Offset. Some authors have argued that it is better to leave the count data unchanged to preserve their sampling properties and instead use an offset for normalization purposes in the statistical model for read counts [5, 6, 9]. This can be achieved easily using the argument `offset` in both normalization functions.

```
dataOffset <- withinLaneNormalization(data,"gc",
                                      which="full",offset=TRUE)
dataOffset <- betweenLaneNormalization(dataOffset,
                                       which="full",offset=TRUE)
```

Note that the `dataOffset` object will have both normalized counts and offset stored in their respective slots.

6 Differential expression analysis

One of the main applications of RNA-Seq is differential expression analysis. The normalized counts (or the original counts and the offset) obtained using the *EDASeq* package can be supplied to packages such as *edgeR* [5] or *DESeq* [6] to find differentially expressed genes. This section should be considered only as an illustration of the compatibility of the results of *EDASeq* with two of the most widely used packages for differential expression; our aim is not to compare differential expression strategies (e.g., normalized counts vs. offset).

6.1 edgeR

We can perform a differential expression analysis with *edgeR* based on the original counts by passing an offset to the generalized linear model. For simplicity, we estimate a common dispersion parameter for all genes. See the *edgeR* vignette for details about how to perform a differential expression analysis using a gene-specific dispersion or more complex designs.

```
library(edgeR)
design <- model.matrix(~conditions, data=pData(dataOffset))
disp <- estimateGLMCommonDisp(counts(dataOffset),
                              design, offset=-offst(dataOffset))

fit <- glmFit(counts(dataOffset), design, disp, offset=-offst(dataOffset))

lrt <- glmLRT(fit, coef=2)
topTags(lrt)
```

```
## Coefficient:  conditionswt
##                logFC   logCPM       LR        PValue
## YGL088W   -5.553631 27.34850 923.9754 6.024807e-203
## YPL198W   -7.425858 26.51938 899.4352 1.302001e-197
## YMR013W-A -6.236487 26.16157 775.4217 1.191074e-170
## YHR156C   -8.132766 25.10965 647.5159 7.750968e-143
## YLR106C   -4.162281 25.93346 612.7063 2.884816e-135
## YGL076C   -4.206669 26.10030 585.3729 2.543202e-129
## YOR040W   -4.974040 25.46016 567.4563 2.007391e-125
## YBL004W   -5.261041 25.03250 556.0820 5.983025e-123
## YAL003W   -2.991170 27.49022 517.4098 1.549064e-114
## YJL047C-A -4.346547 24.96907 416.7858  1.221958e-92
##                       FDR
## YGL088W   3.008789e-199
## YPL198W   3.251095e-194
## YMR013W-A 1.982740e-167
## YHR156C   9.677083e-140
## YLR106C   2.881354e-132
## YGL076C   2.116792e-126
## YOR040W   1.432130e-122
## YBL004W   3.734903e-120
## YAL003W   8.595587e-112
## YJL047C-A  6.102458e-90
```

6.2 DESeq

We can perform a differential expression analysis with *DESeq* based on the normalized counts by using the coerce method from the *SeqExpressionSet* class to the *CountDataSet* class of *DESeq*. When working with data that have been normalized for both within- and between-lane effects, we force the size factors to be one, since differences

in lane sequencing depths have already been accounted for in our between-lane normalization. One could also consider only within-lane normalization and account for differences in sequencing depth by estimating the size factors using *DESeq*.

```
library(DESeq)
counts <- as(dataNorm, "CountDataSet")
sizeFactors(counts) <- rep(1,4)
counts <- estimateDispersions(counts)
res <- nbinomTest(counts, "wt", "mut")
head(res)
```

```
##        id baseMean baseMeanA baseMeanB foldChange
## 1 YAL001C    92.00      58.0     126.0  2.1724138
## 2 YAL002W    73.75     111.0      36.5  0.3288288
## 3 YAL003W   565.00     126.0    1004.0  7.9682540
## 4 YAL004W   100.00     133.0      67.0  0.5037594
## 5 YAL005C   259.25     330.5     188.0  0.5688351
## 6 YAL007C    32.25      37.0      27.5  0.7432432
##   log2FoldChange         pval         padj
## 1      1.1192989 3.415271e-04 3.732136e-03
## 2     -1.6045913 2.106666e-06 4.826006e-05
## 3      2.9942636 3.882861e-31 1.615917e-28
## 4     -0.9891932 1.181444e-03 1.022553e-02
## 5     -0.8139176 1.418425e-03 1.178638e-02
## 6     -0.4280937 3.346353e-01 5.776740e-01
```

7 Definitions and conventions

7.1 Rounding

After either within-lane or between-lane normalization, the expression values are not counts anymore. However, their distribution still shows some typical features of counts distribution (e.g., the variance depends on the mean). Hence, for most applications, it is useful to round the normalized values to recover count-like values, which we refer to as "pseudo-counts".

By default, both `withinLaneNormalization` and `betweenLaneNormalization` round the normalized values to the closest integer. This behavior can be changed by specifying `round=FALSE`. This gives the user more flexibility and assures that rounding approximations do not affect subsequent computations (e.g., recovering the offset from the normalized counts).

7.2 Zero counts

To avoid problems in the computation of logarithms (e.g. in log-fold-changes), we add a small positive constant (namely 0.1) to the counts. For instance, the log-fold-change between y_1 and y_2 is defined as

$$\frac{\log(y_1 + 0.1)}{\log(y_2 + 0.1)}.$$

7.3 Offset

We define an offset in the normalization as

$$o = \log(y_{norm} + 0.1) - \log(y_{raw} + 0.1),$$

where y_{norm} and y_{raw} are the normalized and raw counts, respectively.

One can easily recover the normalized data from the raw counts and offset, as shown here:

```
dataNorm <- betweenLaneNormalization(data, round=FALSE, offset=TRUE)

norm1 <- normCounts(dataNorm)
norm2 <- exp(log(counts(dataNorm) + 0.1 ) + offst(dataNorm)) - 0.1

head(norm1 - norm2)
```

```
##                  mut_1         mut_2          wt_1
## YAL001C -1.421085e-14 -1.421085e-14  0.000000e+00
## YAL002W  3.552714e-15 -7.105427e-15 -2.131628e-14
## YAL003W  4.547474e-13 -2.273737e-13  1.136868e-13
## YAL004W -2.842171e-14 -1.421085e-14  5.684342e-14
## YAL005C -1.136868e-13  5.684342e-14 -3.410605e-13
## YAL007C  3.552714e-15  3.552714e-15 -3.552714e-15
##                  wt_2
## YAL001C  0.000000e+00
## YAL002W  0.000000e+00
## YAL003W -5.684342e-14
## YAL004W -5.684342e-14
## YAL005C -1.136868e-13
## YAL007C  3.552714e-15
```

Note that the small constant added in the definition of offset does not matter when pseudo-counts are considered, i.e.,

```
head(round(normCounts(dataNorm)) - round(counts(dataNorm) * exp(offst(dataNorm))))
```

```
##         mut_1 mut_2 wt_1 wt_2
## YAL001C     0     0    0    0
## YAL002W     0     0    0    0
## YAL003W     0     0    0    0
## YAL004W     0     0    0    0
## YAL005C     0     0    0    0
## YAL007C     0     0    0    0
```

We defined the offset as the log-ratio between normalized and raw counts. However, the *edgeR* functions expect as offset argument the log-ratio between raw and normalized counts. One must use `-offst(offsetData)` as the offset argument of *edgeR* (see Section 6.1).

8 Retrieving gene length and GC-content

Two essential features the gene-level EDA normalizes for are gene length and GC-content. As users might wish to automatically retrieve this information, we provide the function `getGeneLengthAndGCContent`. Given selected ENTREZ or ENSEMBL gene IDs and the organism under investigation, this can be done either based on BioMart (default) or using BioC annotation utilities.

```
getGeneLengthAndGCContent(id=c("ENSG00000012048", "ENSG00000139618"), org="hsa")
```

```
##                  length        gc
## ENSG00000012048   9187 0.4607598
## ENSG00000139618  12273 0.3644586
```

Accordingly, we can retrieve the precalculated yeast data that has been used throughout the vignette via

```
fData(data) <- getGeneLengthAndGCContent(featureNames(data),
                                         org="sacCer3", mode="org.db")
```

9 SessionInfo

```
toLatex(sessionInfo())
```

- R version 3.5.0 (2018-04-23), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.4 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BSgenome 1.48.0, Biobase 2.40.0, BiocGenerics 0.26.0, BiocParallel 1.14.1, Biostrings 2.48.0, DESeq 1.32.0, DelayedArray 0.6.1, EDASeq 2.14.1, GenomeInfoDb 1.16.0, GenomicAlignments 1.16.0, GenomicRanges 1.32.3, IRanges 2.14.10, Rsamtools 1.32.0, S4Vectors 0.18.3, ShortRead 1.38.0, SummarizedExperiment 1.10.1, XVector 0.20.0, edgeR 3.22.3, knitr 1.20, lattice 0.20-35, leeBamViews 1.16.0, limma 3.36.2, locfit 1.5-9.1, matrixStats 0.53.1, rtracklayer 1.40.3, yeastRNASeq 0.18.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.42.1, BiocStyle 2.8.2, DBI 1.0.0, GenomeInfoDbData 1.1.0, GenomicFeatures 1.32.0, KernSmooth 2.23-15, Matrix 1.2-14, R.methodsS3 1.7.1, R.oo 1.22.0,

R.utils 2.6.0, R6 2.2.2, RColorBrewer 1.1-2, RCurl 1.95-4.10, RSQLite 2.1.1, Rcpp 0.12.17, XML 3.98-1.11, annotate 1.58.0, aroma.light 3.10.0, assertthat 0.2.0, backports 1.1.2, biomaRt 2.36.1, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, compiler 3.5.0, crayon 1.3.4, curl 3.2, digest 0.6.15, evaluate 0.10.1, genefilter 1.62.0, geneplotter 1.58.0, grid 3.5.0, highr 0.7, hms 0.4.2, htmltools 0.3.6, httr 1.3.1, hwriter 1.3.2, latticeExtra 0.6-28, magrittr 1.5, memoise 1.1.0, pkgconfig 2.0.1, prettyunits 1.0.2, progress 1.2.0, rlang 0.2.1, rmarkdown 1.10, rprojroot 1.3-2, splines 3.5.0, stringi 1.2.3, stringr 1.3.1, survival 2.42-3, tools 3.5.0, xtable 1.8-2, yaml 2.1.19, zlibbioc 1.26.0

References

[1] A. Lee, K.D. Hansen, J. Bullard, S. Dudoit, and G. Sherlock. Novel low abundance and transient RNAs in yeast revealed by tiling microarrays and ultra high-throughput sequencing are not conserved across closely related yeast species. *PLoS Genet*, 4 (12):e1000299, 2008.

[2] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[3] Saccharomyces Genome Database. http://www.yeastgenome.org, r64.

[4] J.H. Bullard, E. Purdom, K.D. Hansen, and S. Dudoit. Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC bioinformatics*, 11(1):94, 2010.

[5] M.D. Robinson, D.J. McCarthy, and G.K. Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26 (1):139, 2010.

[6] S. Anders and W. Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11(10):R106, 2010.

[7] A. Oshlack and M.J. Wakefield. Transcript length bias in RNA-seq data confounds systems biology. *Biology Direct*, 4(1):14, 2009.

[8] D. Risso, K. Schwartz, G. Sherlock, and S. Dudoit. GC-Content Normalization for RNA-Seq Data. Technical report #291, University of California, Berkeley, Division of Biostatistics, 2011. http://www.bepress.com/ucbbiostat/paper291/.

[9] K.D. Hansen, R.A. Irizarry, and Z. Wu. Removing technical variability in RNA-Seq data using conditional quantile normalization. Technical report #227, Johns Hopkins University, Dept. of Biostatistics Working Papers, 2011. http://www.bepress.com/jhubiostat/paper227/.